Blender 3D: Designing Objects

Table of Contents

Blender 3D: Designing Objects

Blender 3D: Designing Objects

Build your very own stunning characters in Blender from scratch

A course in three modules

BIRMINGHAM - MUMBAI

Blender 3D: Designing Objects

Published on: September 2016

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78712-719-7

www.packtpub.com

Credits

Authors

Romain Caudron

Pierre-Armand Nicq

Enrico Valenza

Reviewers

Fernando Castilhos Melo

Luo Congyi

Waqas Abdul Majeed

Reynante M. Martinez

Jordan Matelsky

Ian Smithers

Romain Caudron

John E. Herreño

Sanu Vamanchery Mana

Content Development Editor

Parshva Sheth

Graphics

Jason Monteiro

Production Coordinator

Aparna Bhagat

Preface

Blender is a powerful tool, stable, with an integral workflow that will allow you to understand your learning of 3D creation with serenity. Today, it is considered to be one of the most complete 3D packages on the market and it is free and open source! It is very efficient for many types of productions, such as 3D animated or live action films, architecture, research, or even game creation with its integrated game engine and its use of the Python language. Moreover, Blender has an active community that contributes to expanding its functionalities. Today, it is used in many professional products and by many companies.

What this learning path covers

Module 1, *Blender 3D By Example*,

Through this module, you will create many types of concert projects using a step-by-step approach. You will start by getting to know the modeling tools available in Blender as you create a 3D robot toy. Then, you will discover more advanced techniques such as sculpting and re-topology by creating a funny alien character. After that, you will create a full haunted house scene.

For the last project, you will create a short film featuring a rat cowboy shooting cheese in a rat trap! This will be a more complex project in which you learn how to rig, animate, compose advanced material, composite, and edit a full sequence.

Each project in this module will give you more practice and increase your knowledge of the Blender tools, and Blender game engine. By the end of this book, you will master a workflow that you will be able to apply to your own creations.

Module 2, *Blender 3D Cookbook*,

This module will take you on a journey to understand the workflow normally used to create characters, from the modeling to the rendering stages using the tools of the last official release of Blender exclusively.

This module helps you create a character mesh and sculpt features, using tools and techniques such as the Skin modifier and polygon merging. You will also get a detailed, step-by-step overview of how to rig and skin your character for animation, how to paint textures and create shaders, and how to perform rendering and compositing. With the help of this book, you will be making production-quality 3D models and characters quickly and efficiently, which will be ready to be added to your very own animated feature or game.

Module 3, *Blender Cycles: Materials and Textures Cookbook - Third Edition*

This module will teach you how to utilize the power of the Blender series to create a wide variety of materials, textures, and effects with the Cycles rendering engine. You will learn about node-based shader creation, and master cycles through step-by-step, recipe-based advice. With this book, you will start small by rendering the textures of stones and water, then scale things up to massive landscapes of

mountains and oceans. You will then learn how to create the look of different artificial materials such as plastic, carpenter wood, and metal, and utilize volumetric shaders to create the effects of smoke, clouds, and subsurface scattering effects of skin. You will also learn how illumination works in Cycles, improvising the quality of the final render, and how to avoid the presence of noise and fireflies. By the end, you will know how to create an impressive library of realistic-looking materials and textures.

What you need for this learning path

The only software strictly needed to put into practice the content of this course is the last official Blender release. You just have to download it from http://www.blender.org/download/get-blender; some Python script may be necessary in some recipes, but for the most part, they should all be included in the Blender package. Eventually, you can quite surely find any missing add-on at http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts.

Any particular texture needed for the exercises in the book is provided as a free download on the Packt Publishing website itself. Not essential, but handy to have is a 2D image editor, in case you want to adapt your own textures to replace the provided ones; I suggest you try Gimp, an open source image editor that you can download from http://www.gimp.org; any other one you prefer is perfect anyway.

Who this learning path is for

If you are a graphic designer and are looking for a tool to meet your requirements in designing, especially with regards to 3D designing, this course is for you. This course will make use of Blender to meet your design needs.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail <feedback@packtpub.com>, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at http://www.packtpub.com. If you purchased this course elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at https://github.com/PacktPublishing/Blender-for-Designers/tree/master. We also have other code bundles from our rich catalog of books, videos, and courses available at https://github.com/PacktPublishing/. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting http://www.packtpub.com/submit-errata, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to https://www.packtpub.com/books/content/support and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at <questions@packtpub.com>, and we will do our best to address the problem.

Part 1. Module 1

Blender 3D By Example

Design a complete workflow with Blender to create stunning 3D scenes and films step-by-step!

Chapter 1. Straight into Blender!

Welcome to the first chapter, in which you will start getting familiar with Blender.

Here, navigation within the interface will be presented. Its approach is atypical in comparison to other 3D software, such as Autodesk Maya® or Autodesk 3DS Max®, but once you get used to this, it will be extremely effective.

If you have had the opportunity to use Blender before, it is important to note that the interface went through changes during the evolution of the software (especially since version 2.5).

We will give you an idea of the possibilities that this wonderful free and open source software gives by presenting different workflows. You will learn some vocabulary and key concepts of 3D creation so that you will not to get lost during your learning.

Finally, you will have a brief introduction to the projects that we will carry out throughout this module.

Let's dive into the third dimension! The following topics will be covered in this chapter:

- Learning some theory and vocabulary
- Navigating the 3D viewport
- How to set up preferences
- Using keyboard shortcuts to save time

An overview of the 3D workflow

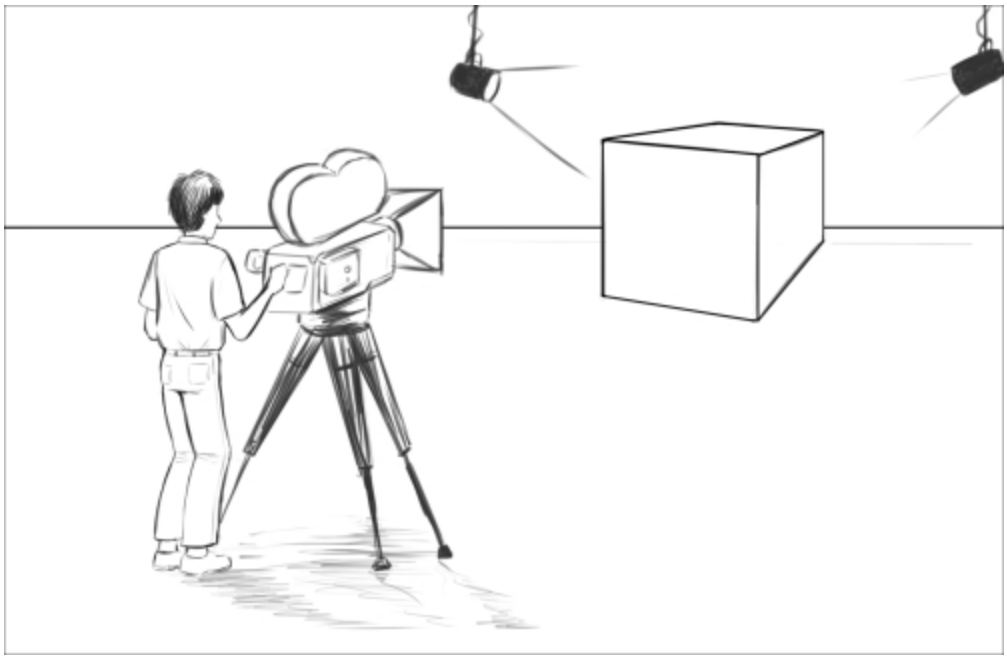
Before learning how to navigate the Blender interface, we will give you a short introduction to the 3D workflow.

The anatomy of a 3D scene

To start learning about Blender, you need to understand some basic concepts. Don't worry, there is no need to have special knowledge in mathematics or programming to create beautiful 3D objects; it only requires curiosity. Some artistic notions are a plus.

All 3D elements, which you will handle, will evolve in to a scene. There is a three-dimensional space with a coordinate system composed of three axes. In Blender, the *x* axis shows the width, *y* axis shows the depth, and the *z* axis shows the height. Some softwares use a different approach and reverses the *y* and *z* axes. These axes are color-coded, we advise you to remember them: the *x* axis in red, the *y* axis in green and the *z* axis in blue.

A scene may have the scale you want and you can adjust it according to your needs. This looks like a film set for a movie. A scene can be populated by one or more cameras, lights, models, rigs, and many other elements. You will have the control of their placement and their setup.



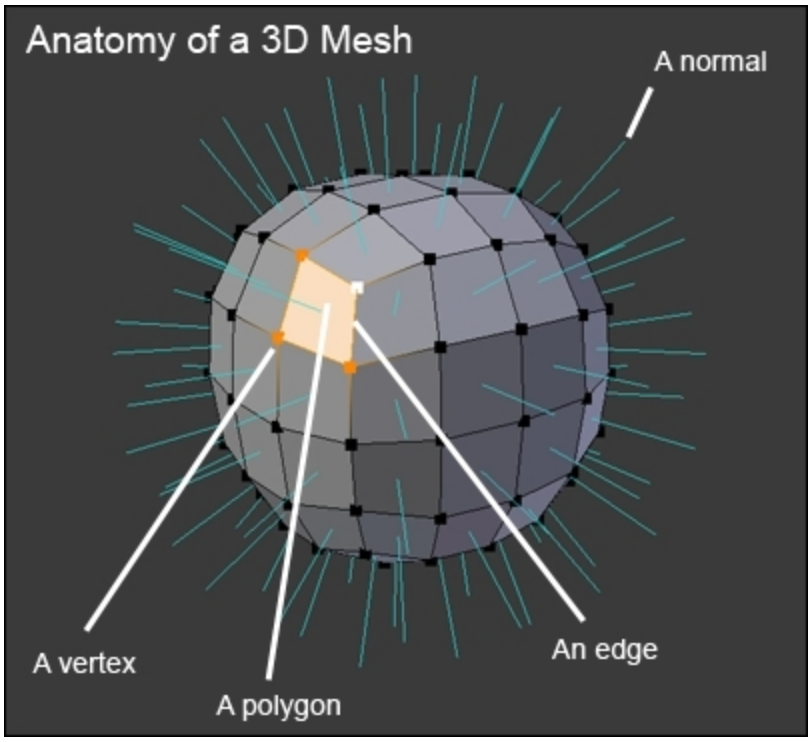
A 3D scene looks like a film set.

A mesh is made of vertices, edges, and faces. The vertices are some points in the scene space that are placed at the end of the edges. They could be thought of as 3D points in space and the edges connect them. Connected together, the edges and the vertices form a face, also called a polygon. It is a geometric plane, which has several sides as its name suggests.

In 3D software, a polygon is constituted of at least three sides. It is often essential to favor four-sided polygons during modeling for a better result. You will have an opportunity to see this in more detail later.

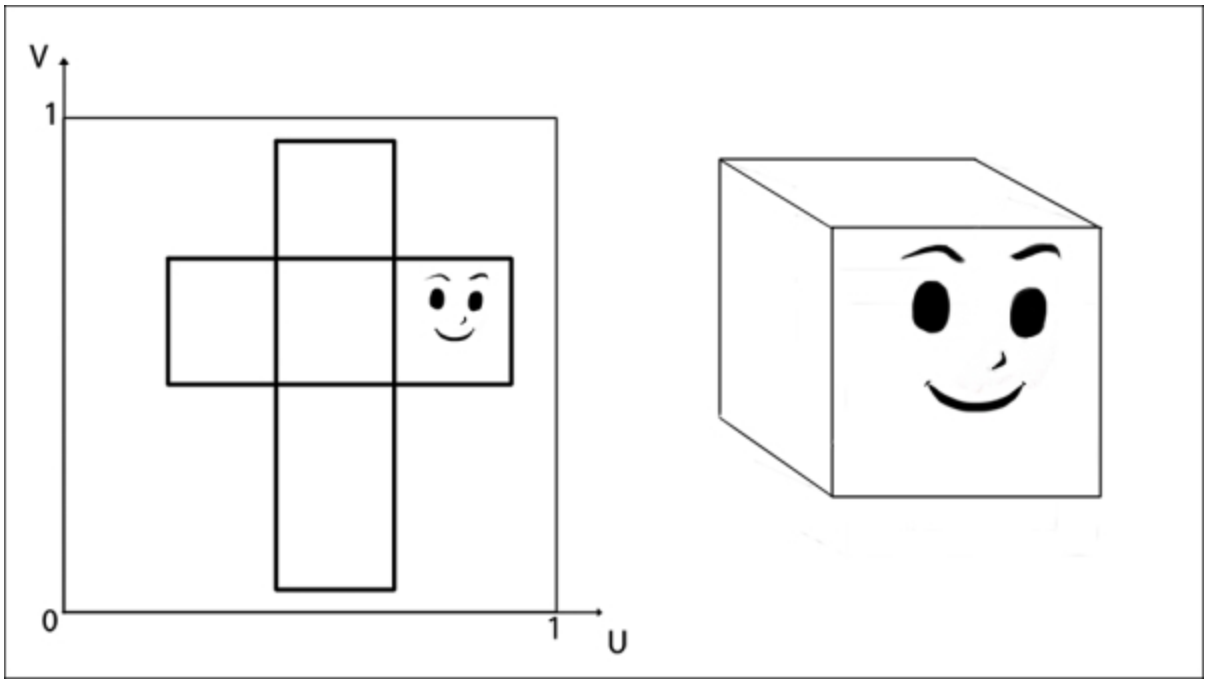
Your actors and environments will be made of polygonal objects, or more commonly called as meshes. If you have played old 3D games, you've probably noticed the very angular outline of the characters; it was, in fact, due to a low count of polygons.

We must clarify that the orientation of the faces is important for your polygon object to be illuminated. Each face has a normal. This is a perpendicular vector that indicates the direction of the polygon. In order for the surface to be seen, it is necessary that the normals point to the outside of the model. Except in special cases where the interior of a polygonal object is empty and invisible. You will be able to create your actors and environment as if you were handling virtual clay to give them the desired shape.



Anatomy of a 3D Mesh

To make your characters presentable, you will have to create their textures, which are 2D images that will be mapped to the 3D object. UV coordinates will be necessary in order to project the texture onto the mesh. Imagine an origami paper cube that you are going to unfold. This is roughly the same. These details are contained in a square space with the representation of the mesh laid flat. You can paint the texture of your model in your favorite software, even in Blender.



This is the representation of the UV mapping process. The texture on the left is projected to the 3D model on the right.

After this, you can give the illusion of life to your virtual actors by animating them. For this, you will need to place animation keys spaced on the timeline. If you change the state of the object between two keyframes, you will get the illusion of movement—animation. To move the characters, there is a very interesting process that uses a bone system, mimicking the mechanism of a real skeleton. Your polygon object will be then attached to the skeleton with a weight assigned to the vertices on each bone, so if you animate the bones, the mesh components will follow them.

Once your characters, props, or environment are ready, you will be able to choose a focal length and an adequate framework for your camera.

In order to light your scene, the choice of the render engine will be important for the kind of lamps to use, but usually there are three types of lamps as used in cinema productions. You will have to place them carefully. There are directional lights, which behave like the sun and produce hard shadows. There are omnidirectional lights, which will allow you to simulate diffuse light, illuminating everything around it and casting soft shadows. There are also spots that will simulate a conical shape. As in the film industry or other imaging creation fields, good lighting is a must-have in order to sell the final picture. Lighting is an expressive and narrative element that can magnify your models, or make them irrelevant.

Once everything is in place, you are going to make a render. You will have a choice between a still image and an animated sequence. All the given parameters with the lights and materials will be calculated by the render engine. Some render engines offer an approach based on physics with rays that are launched from the camera. Cycles is a good example of this kind of engine and succeed in producing

very realistic renders. Others will have a much simpler approach, but none less technically based on visible elements from the camera.

All of this is an overview of what you will be able to achieve while reading this module and following along with Blender.

What can you do with Blender?

In addition to being completely free and open source, Blender is a powerful tool that is stable and with an integral workflow that will allow you to understand your learning of 3D creation with ease. Software updates are very frequent; they fix bugs and, more importantly, add new features.

You will not feel alone as Blender has an active and passionate community around it. There are many sites providing tutorials, and an official documentation detailing the features of Blender.

You will be able to carry out everything you need in Blender, including things that are unusual for a 3D package such as concept art creation, sculpting, or digital postproduction, which we have not yet discussed, including compositing and video editing. This is particularly interesting in order to push the aesthetics of your future images and movies to another level.

It is also possible to make video games. Also, note that the Blender game engine is still largely unknown and underestimated. Although this aspect of the software is not as developed as other specialized game engines, it is possible to make good quality games without switching to another software.

You will realize that the possibilities are enormous, and you will be able to adjust your workflow to suit your needs and desires.

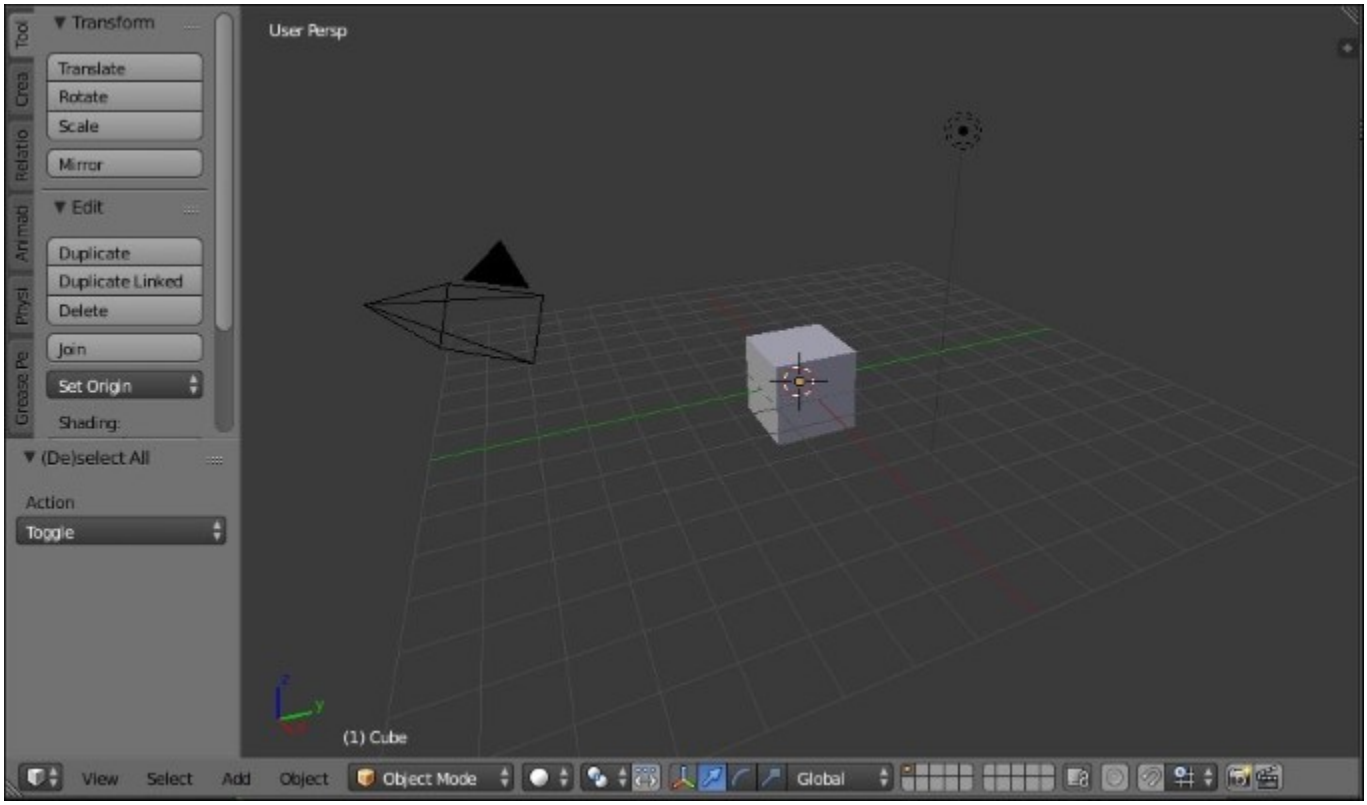
Software of this type could scare you by its unusual handling and its complexity, but you'll realize that once you have learned its basics, it is really intuitive in many ways.

Getting used to the navigation in Blender

Now that you have been introduced to the 3D workflow, you will learn how to navigate the Blender interface, starting with the 3D viewport.

An introduction to the navigation of the 3D Viewport

It is time to learn how to navigate in the Blender viewport. The viewport represents the 3D space, in which you will spend most of your time. As we previously said, it is defined by three axes (*x*, *y*, and *z*). Its main goal is to display the 3D scene from a certain point of view while you're working on it.



The Blender 3D Viewport

When you are navigating through this, it will be as if you were a movie director but with special powers that allow you to film from any point of view.

The navigation is defined by three main actions: pan, orbit, and zoom. The pan action means that you will move horizontally or vertically according to your current point of view. If we connect that to our cameraman metaphor, it's like if you were moving laterally to the left, or to the right, or moving up or down with a camera crane.

By default, in Blender the shortcut to pan around is to press the *Shift* button and the **Middle Mouse Button** (**MMB**), and drag the mouse.

The orbit action means that you will rotate around the point that you are focusing on. For instance, imagine that you are filming a romantic scene of two actors and that you rotate around them in a circular manner. In this case, the couple will be the main focus. In a 3D scene, your main focus would be a 3D character, a light, or any other 3D object.

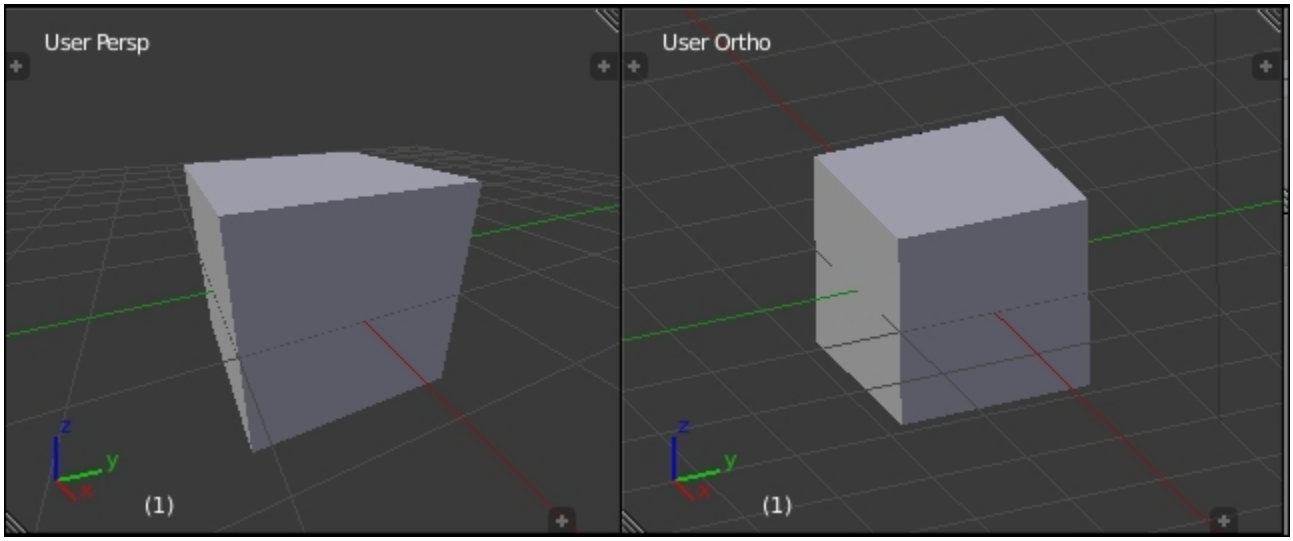
To orbit around in the Blender viewport, the default shortcut is to press the MMB and then drag the mouse.

The last action that we mentioned is zoom. The zoom action is straightforward. It is the action of moving our point of view closer to an element or further away from an element.

In Blender, you can zoom in by scrolling your mouse wheel up and zoom out by scrolling your mouse wheel down.

To gain time and precision, Blender proposes some predefined points of view. For instance, you can quickly go in a top view by pressing the numpad *7*, you can also go in a front view by pressing the numpad *1*, you can go in a side view by pressing the numpad *3*, and last but not least, the numpad *0* allows you to go in **Camera** view, which represents the final render point of the view of your scene.

You can also press the numpad *5* in order to activate or deactivate the orthographic mode. The orthographic mode removes perspective. It is very useful if you want to be precise. It feels as if you were manipulating a blueprint of the 3D scene.



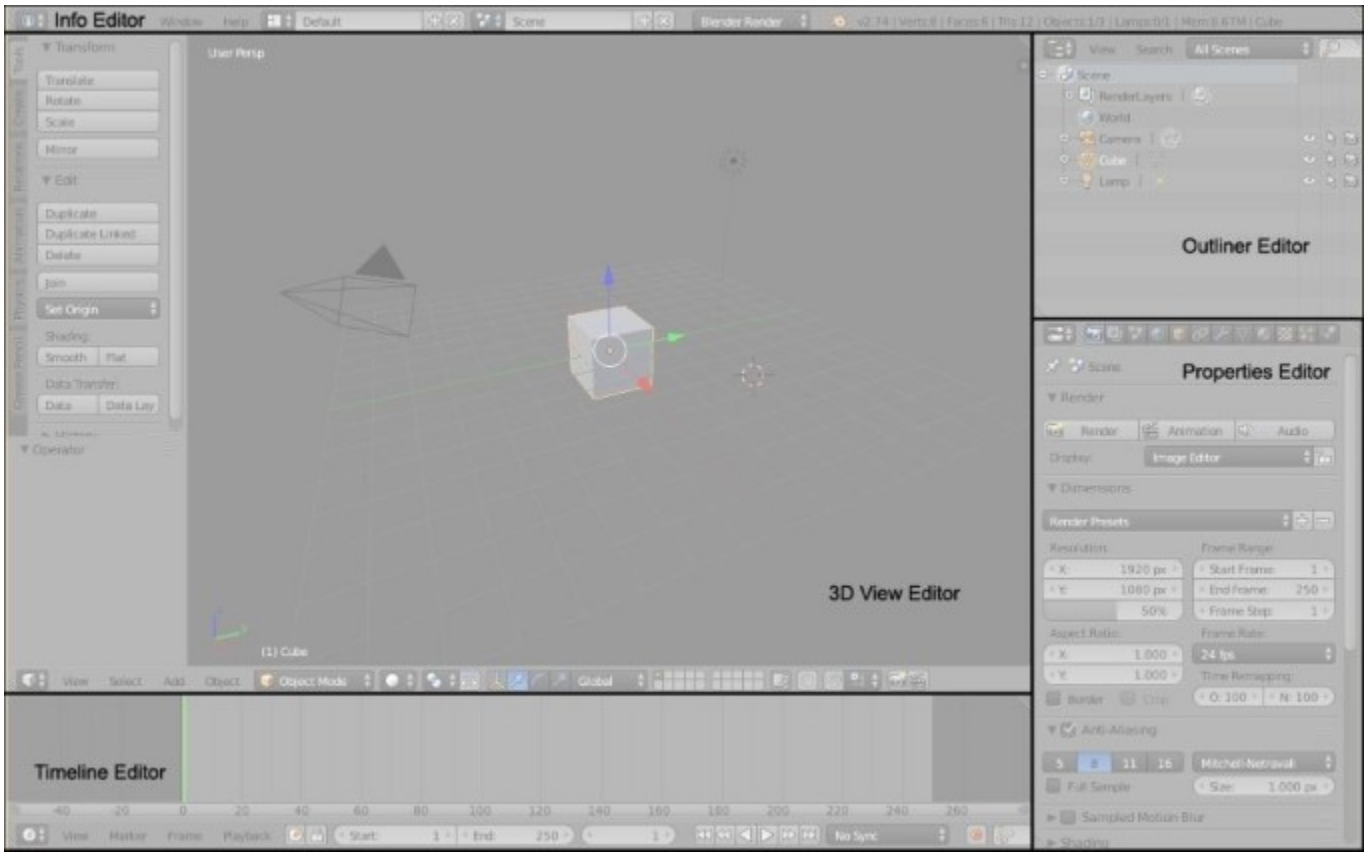
The difference between Perspective (left) and Orthographic (right)

If you are lost, you can always look at the top left corner of the viewport in order to see in which view you are, and whether the orthographic mode is on or off.

Try to learn by heart all these shortcuts; you will use them a lot. With repetition, this will become a habit.

What are editors?

In Blender, the interface is divided into subpanels that we call **editors**; even the menu bar where you save your file is an editor. Each editor gives you access to tools categorized by their functionality. You have already used an editor, the 3D view. Now it's time to learn more about the editor's anatomy.

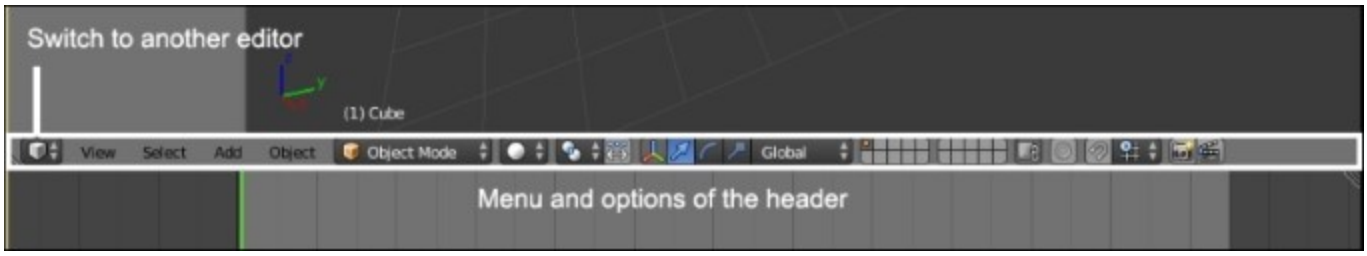


In this picture, you can see how Blender is divided into editors

The anatomy of an editor

There are 17 different editors in Blender and they all have the same base. An editor is composed of a **Header**, which is a menu that groups different options related to the editor. The first button of the header is to switch between other editors. For instance, you can replace the 3D view by the **UV/Image Editor** by clicking on it. You can easily change its place by right-clicking on it in an empty space and by choosing the **Flip to Top/Bottom** option.

The header can be hidden by selecting its top edge and by pulling it down. If you want to bring it back, press the little plus sign at the far right.



The header of the 3D viewport. The first button is for switching between editors, and also, we can choose between different options in the menu

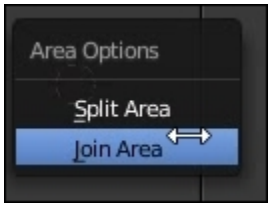
In some editors, you can get access to hidden panels that give you other options. For instance, in the 3D view you can press the *T* key or the *N* key to toggle them on or off. As in the header, if a sub panel of an editor is hidden, you can click on the little plus sign to display it again.

Split, Join, and Detach

Blender offers you the possibility of creating editors where you want. To do this, you need to right-click on the border of an editor and select **Split Area** in order to choose where to separate them.

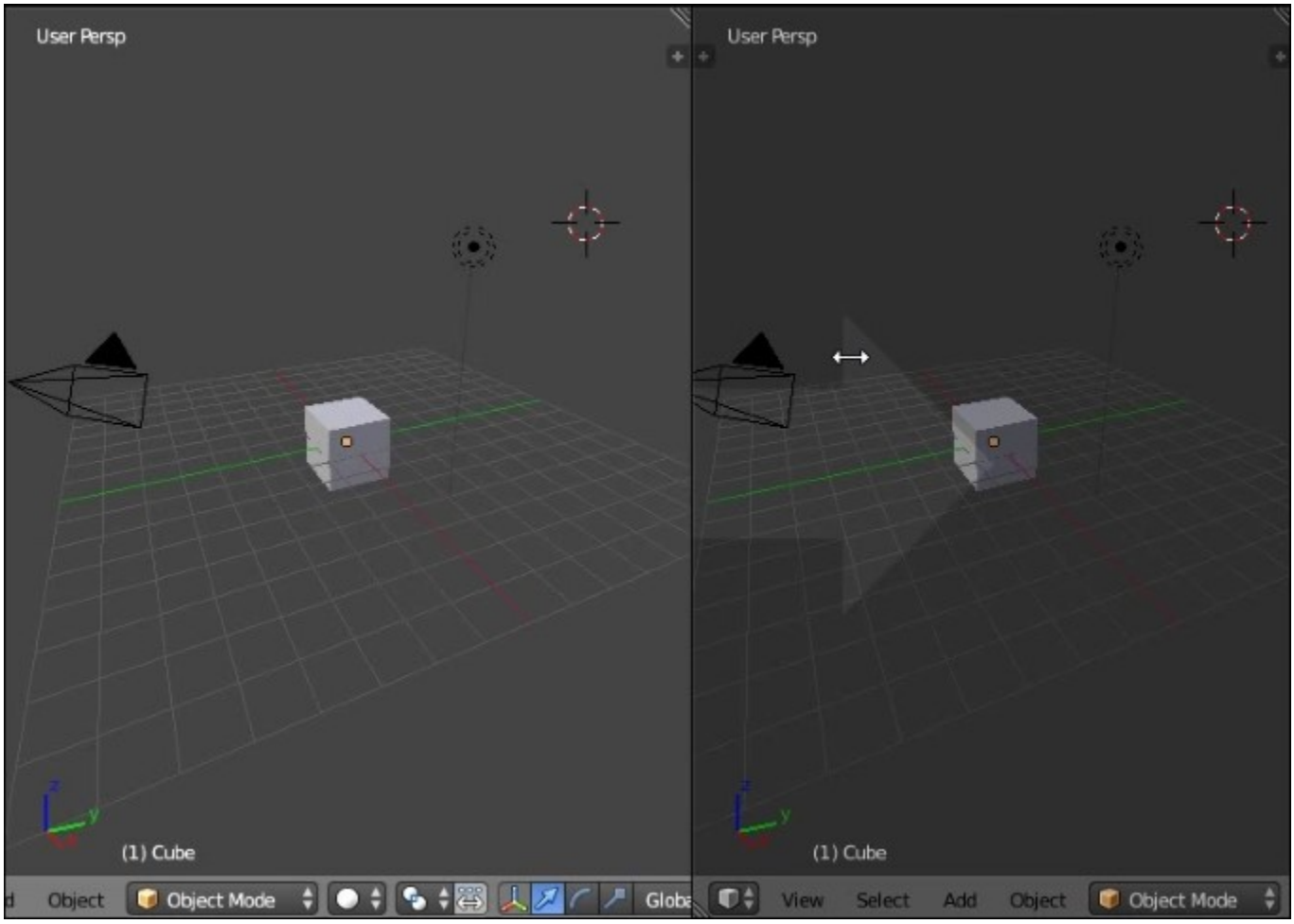
Right-click on the border of an editor to split it into two editors

The current editor will then be split in two editors. Now you can switch to any other editor that you desire by clicking on the first button of the header bar. If you want to merge two editors into one, you can right-click on the border that separates them and select the **Join Area** button. You will then have to click on the editor that you want to erase by pointing the arrow on it.



*Use the **Join Area** option to join two editors together*

You then have to choose which editor you want to remove by pointing and clicking on it.



We are going to see another method of splitting editors that is nice. You can drag the top right corner of an editor and another editor will magically appear! If you want to join back two editors together, you will have to drag the top right corner in the direction of the editor that you want to remove. The last manipulation can be tricky at first, but with a little bit of practice, you will be able to do it closed eyes!

The top right corner of an editor

If you have multiple monitors, it could be a great idea to detach some editors in a separated window. With this, you could gain space and won't be overwhelmed by a condensed interface. In order to do this, you will need to press the *Shift* key and drag the top right corner of the editor with the **Left Mouse Button** (**LMB**).

Some useful layout presets

Blender offers you many predefined layouts that depend on the context of your creation. For instance, you can select the Animation preset in order to have all the major animation tools, or you can use the UV Editing preset in order to prepare your texturing. To switch between the presets, go to the top of the interface (in the **Info** editor, near the **Help** menu) and click on the drop-down menu. If you want, you can add new presets by clicking on the plus sign or delete presets by clicking on the X button. If you want to rename a preset, simply enter a new name in the corresponding text field. The following screenshot shows the Layout presets drop-down menu:

The layout presets drop-down menu

Setting up your preferences

When we start learning new software, it's good to know how to set up your preferences. Blender has a large number of options, but we will show you just the basic ones in order to change the default navigation style or to add new tools that we call add-ons in Blender.

An introduction to the Preferences window

The preferences window can be opened by navigating to the **File** menu and selecting the **User Preferences** option. If you want, you can use the *Ctrl* + *Alt* + *U* shortcut or the *Cmd* key and comma key on a Mac system.

There are seven tabs in this window as shown here:

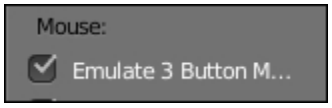
The different tabs that compose the Preferences window

A nice thing that Blender offers is the ability to change its default theme. For this, you can go to the **Themes** tab and choose between different presets or even change the aspect of each interface elements.

Another useful setting to change is the number of undo that is 32 steps, by default. To change this number, go to the **Editing** tab and under the **Undo** label, slide the **Steps** to the desired value.

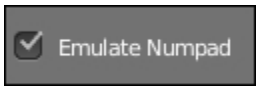
Customizing the default navigation style

We will now show you how to use a different style of navigation in the viewport. In many other 3D programs, such as Autodesk Maya®, you can use the *Alt* key in order to navigate in the 3D view. In order to activate this in Blender, navigate to the **Input** tab, and under the **Mouse** section, check the **Emulate 3 Button Mouse** option. Now if you want to use this navigation style in the viewport, you can press *Alt* and LMB to orbit around, *Ctrl* + *Alt* and the LMB to zoom, and *Alt* + *Shift* and the LMB to pan. Remember these shortcuts as they will be very useful when we enter the sculpting mode while using a pen tablet. The **Emulate 3 Button Mouse** checkbox is shown as follows:



The Emulate 3 Button Mouse will be very useful when sculpting using a pen tablet

Another useful setting is the **Emulate Numpad**. It allows you to use the numeric keys that are above the QWERTY keys in addition to the numpad keys. This is very useful for changing the views if you have a laptop without a numpad, or if you want to improve your workflow speed.



The Emulate Numpad allows you to use the numeric keys above the QWERTY keys in order to switch views or toggle the perspective on or off

Improving Blender with add-ons

If you want even more tools, you can install what is called as add-ons on your copy of Blender. **Add-ons**, also called **Plugins** or **Scripts**, are Python files with the `.py` extension. By default, Blender comes

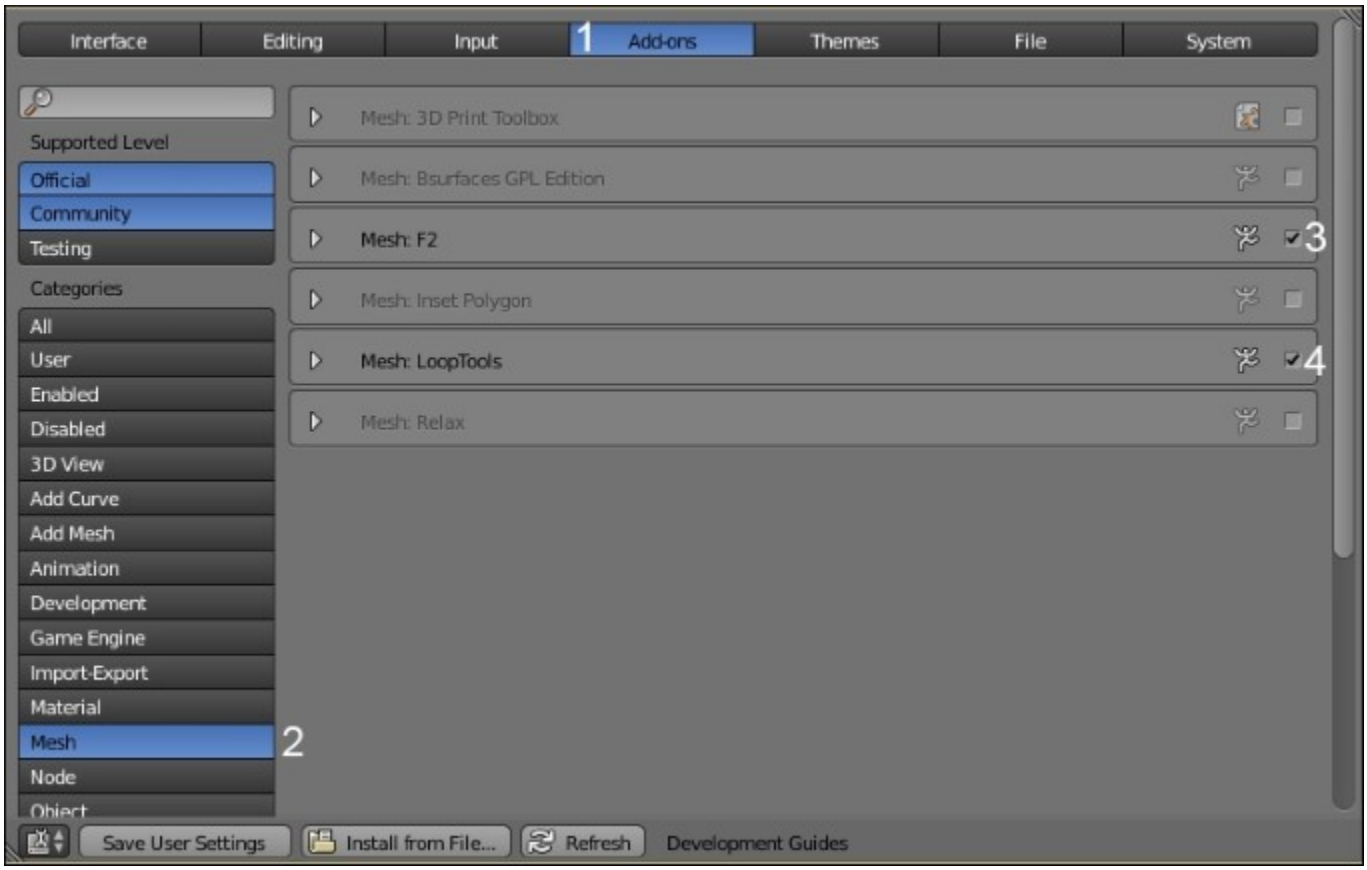
with many disabled add-ons ordered by category. We will now activate two very useful add-ons that will improve our speed while modeling. First, go to the **Add-ons** tab, and click on the **Mesh** button in the category list at the left. Here, you will see all the default mesh add-ons available. Click on the check-boxes at the left of the **Mesh: F2** and **Mesh: LoopTools** subpanels in order to activate these add-ons. If you know the name of the add-on you want to activate, you can try to find it by typing its name in the search bar. There are many websites where you can download free add-ons, starting from the official Blender website. If you want to install a script, you can click on the **Install from File** button and you will be asked to select the corresponding Python file.

Note

The official Blender Add-ons Catalog

You can find it at http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts.

The following screenshot shows the steps for activating the add-ons:



Steps for Add-ons activation

Note

Where are the add-ons on the hard-disk?

All the scripts are placed in the `add-ons` folder that is located wherever you have installed Blender on your hard disk. This folder will usually be at `Your Installation Path\Blender Foundation\Blender\2.VersionNumber\scripts\addons`.

If you find it easier, you can drop the Python files here instead of at the standard installation.

Don't forget to click on the **Save User Settings** button in order to save all your changes!

A brief introduction to the projects

You will now be introduced to the fun projects that we will do together during each of the later chapters. You will need practice to improve your skills.

The Robot Toy

In this project, you will follow step by step the modeling of a little Robot Toy, starting from a simple cube primitive. This old school mechanic robot will make you re-live your childhood. The goal of this chapter is to teach you the modeling process in Blender. You will gain a good overview of the main modeling tools, such as extrude or loop cut. On the other hand, you will discover what a good workflow is by creating your model according to a reference.

The Alien Character

This project will be exciting! We think you will have enough experience to start learning how to create your own alien character using the sculpting tools of Blender. During the project, you will encounter a new modeling process by creating a base mesh for sculpting. After this, you will understand how to retopologize and keep the details of that sculpt. It will be divided into two parts: the sculpting and the retopology process.

The Haunted House

The Haunted House is a nice but scary little house in the middle of the Pennsylvania...Booooohhhhhhhoooohhh! The legend says that it is haunted by thousands of spectrums. In this project, divided into three parts, you will start by modeling the house and its environment while discovering new modeling techniques, such as the array modifier. After completing the modeling, you will learn how to use the powerful Blender texturing and UV tools in order to add colors to your meshes. Finally, you will use the **Cycles** nodal editor in order to create materials with the textures previously made. After reading the corresponding chapters, you will have a good understanding of how a full 3D scene is constructed and how to organize yourself for such a big task.

The Rat Cowboy

The Rat Cowboy and the story of the holes in the cheese will be your first animated sequence. It will be a nice starting point to learn more about rigging and animation. The Rat will face a piece of cheese pinched under a rat trap, and he will unsheathe his gun to shoot the cheese. The Gruyère cheese is born. In order to produce a polished final shot, you will learn some compositing tricks and how to render the sequence with Cycles.

Summary

In this chapter, you have learned the steps behind 3D creations. You know what a mesh is and what it is composed of. Then you have been introduced to navigation in Blender by manipulating the 3D viewport and going through the user preference menu. In the later sections, you configured some preferences and extended Blender by activating some add-ons.

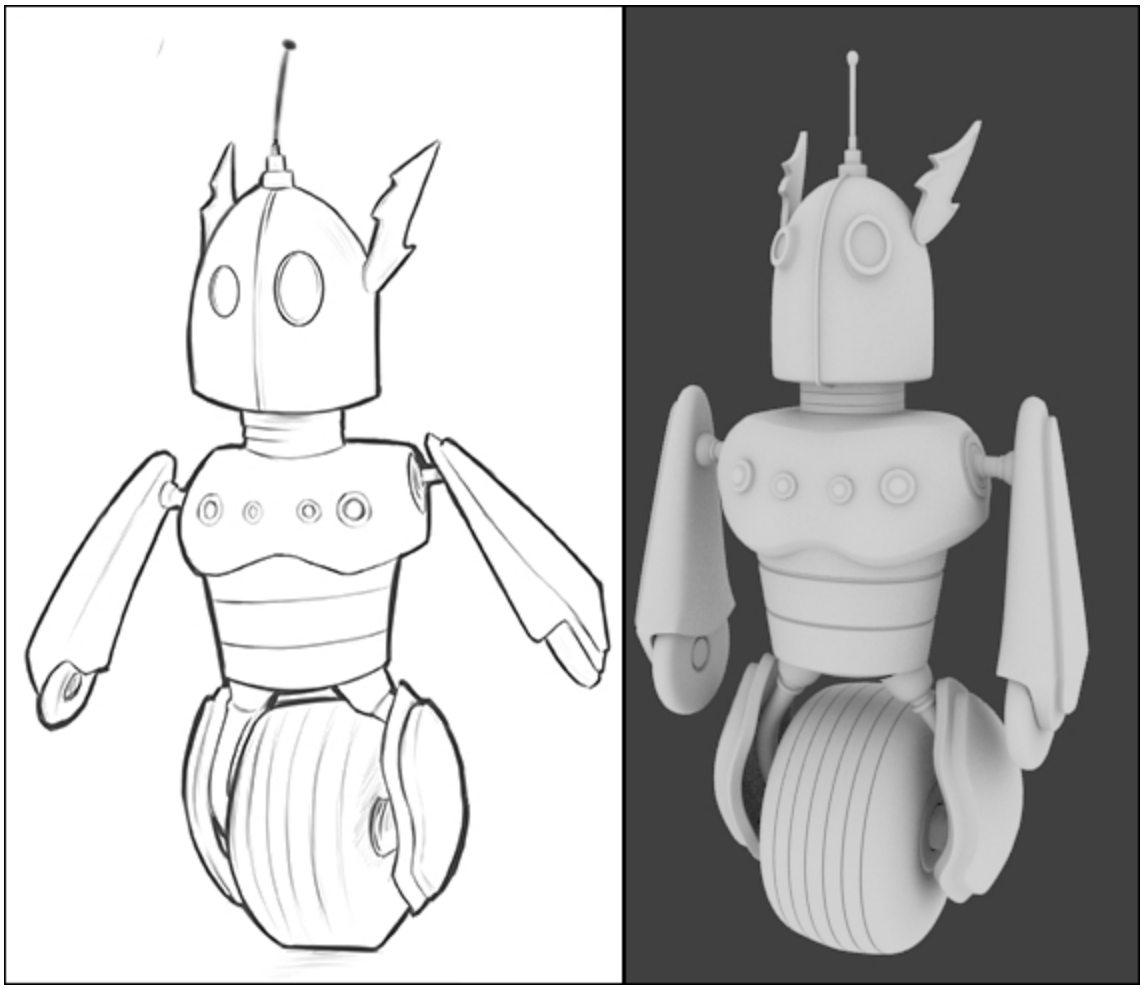
You are now ready to start the 3D modeling of our Robot Toy project.

Chapter 2. Robot Toy – Modeling of an Object

In this chapter, we will start our first project in order to discover the fundamental modeling tools of Blender. We will create a little robot that is inspired by vintage toys with a drawing image reference. You will learn polygonal modeling workflow, which will be useful for your future 3D productions. The head will be created with a simple cylindrical primitive that we will modify to give it the right shape. Then, in the same way, starting from a primitive, we will model the rest of the body, always with a good topology in mind. Indeed, we are going to maximize the number of quads (polygons with four faces) and organize them so that they best fit the shape of each part. In the end, we will do a quick render with the Blender internal render engine. Without further ado, let's enter the marvelous world of 3D modeling! In this chapter, we will cover the following topics:

- Adding and editing objects
- Using the basic modeling tools
- Understanding the basic modifiers (such as mirror and subsurface)
- Modeling with a proper topology
- Creating a quick preview with Blender Internal

In the following screenshot, on the right, you can see the 3D robot modeled using a sketch, shown on the left as a reference, with Krita, which is another open source tool for 2D art:



Let's start the modeling of our robot toy

We will now start the modeling of the robot toy by adding the first object to the scene. The robot will be modeled from a simple cylinder.

Preparing the workflow by adding an image reference

In order to start the modeling of the robot, let's have a look at the following procedure:

1. We will add the robot image reference in a new **UV/Image Editor**.
2. After dividing the view and selecting the right editor (by clicking on the RMB on the edge of an editor and selecting **Split Area**), go to the **UV/Image Editor** header and select **Open Image** to choose the corresponding reference in the file browser.
3. To pan or zoom in this editor, use the same shortcuts as the 3D view. This reference will serve as a guide during the modeling process. Refer to this in order to get the main shape right, but don't rely on its details.

Adding the head primitive

When you start modeling an object, you need to start with a basic 3D shape that is close to the shape you want to model. In our case, we will use a cylindrical primitive to start modeling the head. To do this, follow these steps:

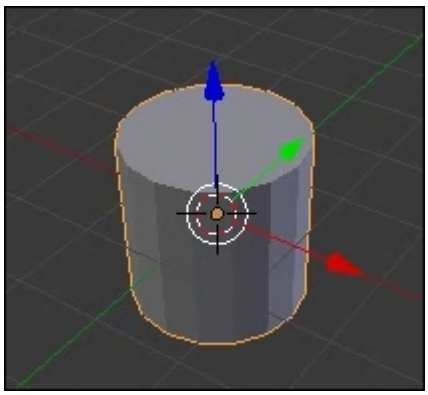
1. First we will need to remove the 3D cube that is placed by default in any Blender starting file. The cube is selected if it has an orange outline. If this isn't the case, you can right-click on it. This is the main selection method in Blender. If you want to select or unselect all the objects present in the 3D view, you can press the *A* (All) key.
2. You can now remove the selected cube by pressing the *X* key or the *Delete* key. It's now time to add the cylindrical primitive.
3. All the primitives are going to spawn at the 3D Cursor location. We will ensure that the cursor is at the center of the scene by pressing *Shift* + *C*.
4. We can now use the *Shift* + *A* shortcut, and select **Mesh** | **Cylinder** to create the primitive at the center of the scene.
5. Our new object has too many details, so we will decrease the number of vertices in the left 3D view panel. If you can't see this panel, press the *T* key. At the bottom of this, you can see the preferences of the currently active tool (the mesh creation, in our case), and you can change the number of vertices of our cylinder to **16**.
6. We will now set the 3D view focus on the newly created object by pressing the dot numpad key or by selecting **View** | **View Selected** in the 3D view header.

Note

About naming shortcuts

Most of the shortcuts correspond to the first letter of the tool's name. For instance, the Grab tool can be activated by the *G* key and the Scale tool can be selected by the *S* key.

If you want to explore all Blender's shortcuts, visit http://www.shortcutsheaven.com/.



The cylinder located at the cursor position (center of the world) that we will use as a base for the head of the robot.

The Edit Mode versus the Object Mode

Currently, we cannot access the components (vertices, edges, and faces) of our cylinder because we are in the **Object Mode**. This mode allows you to do basic things on objects such as moving, rotating, or scaling them. Let's perform the following set of steps:

1. If you want to edit an object, you need to use the **Edit Mode**. To switch between these modes, press the *Tab* key or go to the **Modes** drop-down menu in the 3D view header while any object is selected. In the **Edit Mode**, you can choose the type of components to select by pressing *Ctrl* + *Tab* or by selecting the component type in the 3D view header.
2. Let's go into the **Face Mode** and select the top face of the cylinder by right-clicking on it. As you can see, in Blender, faces (or polygons) are represented by a little square in the middle.
3. Now you can go into the orthographic front view (the *3* numpad key and *5* numpad key for perspective/orthographic views respectively), and use the *z* axis of the Gizmo tool to move the selected face a little bit down.

Note

In Blender, we don't encourage you to use gizmos as there is a much faster method to move, rotate, or scale your selection. To move a selection, press the *G* (Grab) key and, if you want to constrain your move to a certain axis, press the corresponding *X*, *Y*, or *Z* keys. You can even hide the Gizmo tool by pressing *Ctrl* + *Space*.

*The top face moved down in the **Edit Mode** with the z axis of the Gizmo tool or by pressing the G + Z shortcut.*

Using the basic modeling tools

In the following, you will learn the powerful usage of the main modeling tools of Blender, such as the Extrusion, Bevel, or Loop Cut tool, while creating your little robot toy.

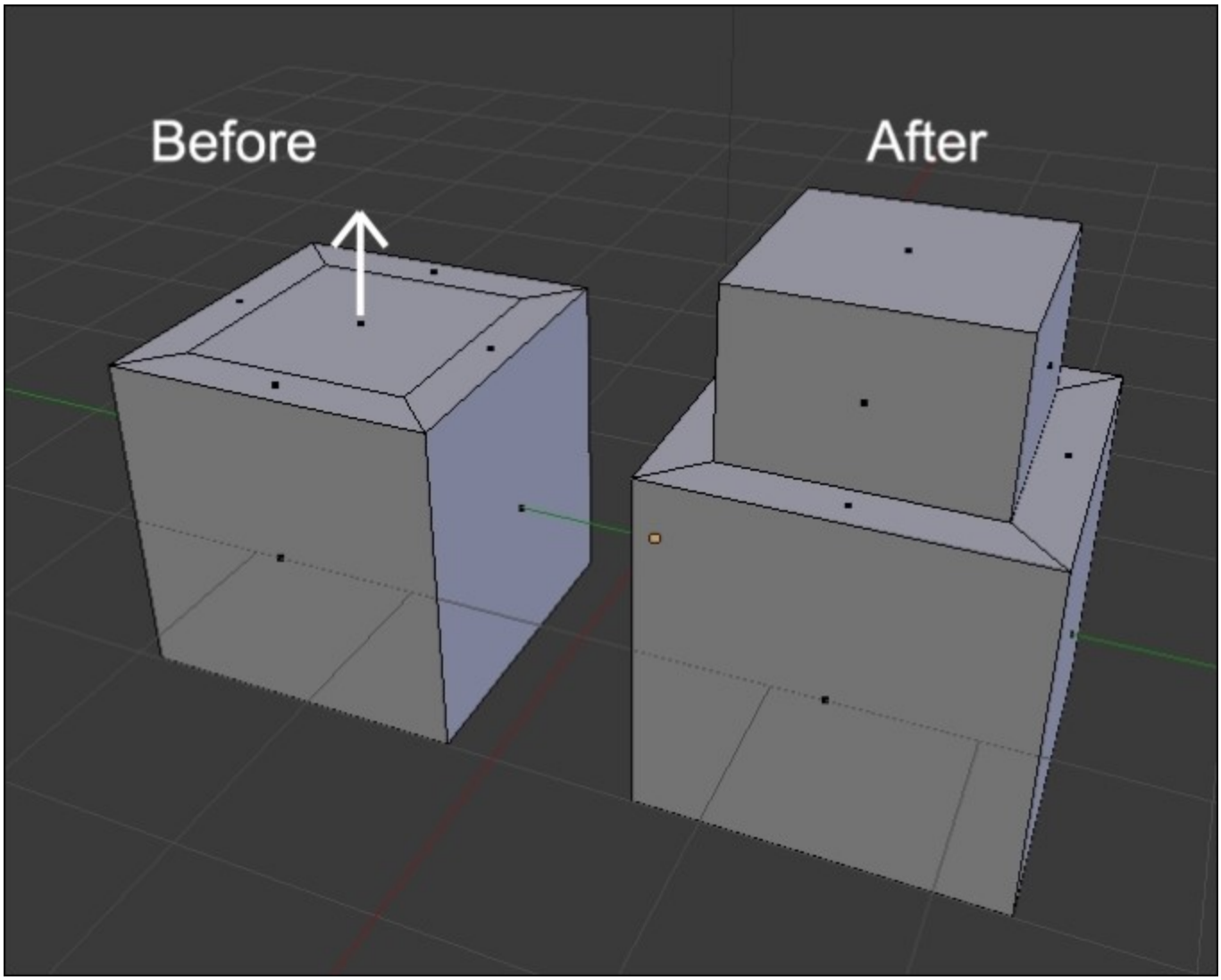
Modeling the head

We will now use the basic modeling tools in order to form the shape of the head. As you may have understood, we are going to add new geometry gradually to approximate the shape in 3D. One of the useful tools is called **Extrusion**.

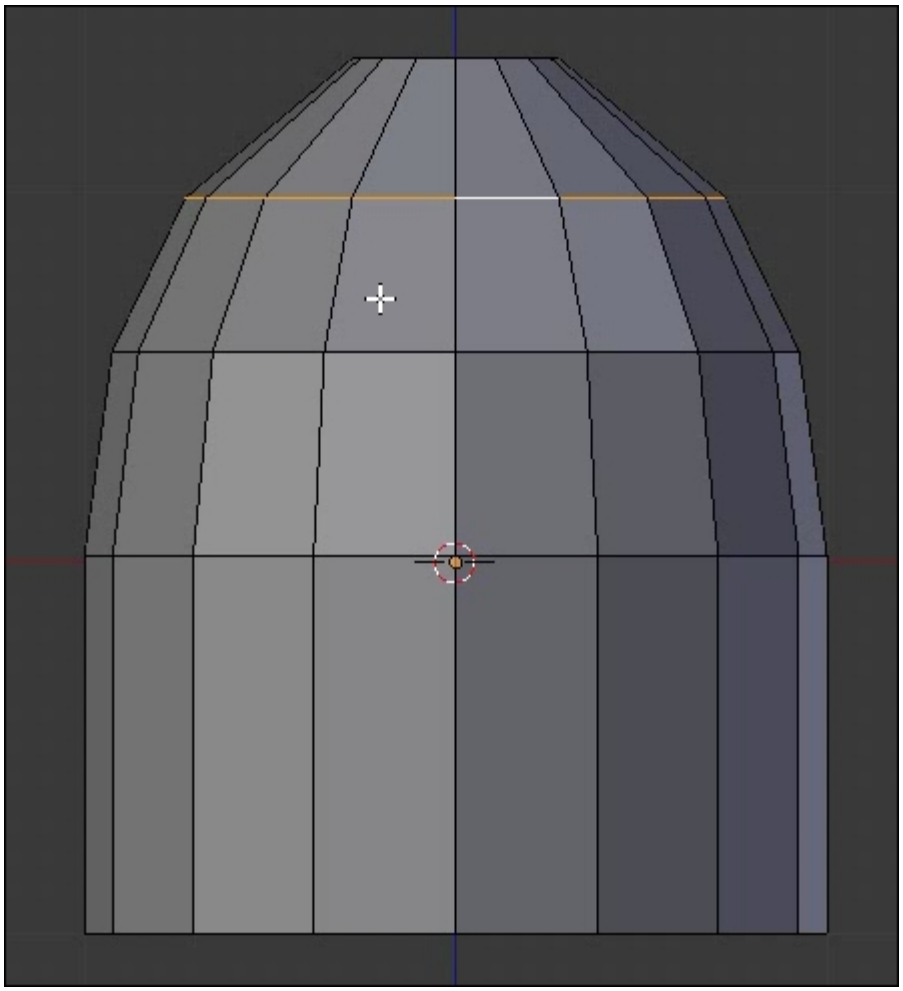
Note

What is an extrusion?

This is the process of creating new geometry by extending (and optionally transforming) selected components.



1. While the top face of the cylinder is selected in the **Edit Mode**, we will press the *E* key in order to create a new geometry from that face. Then, we will need to position and validate the extrusion.
2. We now have two choices in order to confirm the extrusion. If you want, you can move the extruded geometry, and after this, press LMB in order to validate its position. The other choice is to press RMB, in order to place the extruded geometry at the same position as that of the selected component(s). In our case, we will place the extrusion just over the selected face.
3. We can now scale our extrusion by pressing the *S* key, and repeat the process of extruding the top face and scaling it three times in order to have a bell shape. We can always go back by pressing *Ctrl* + *Z* and redo these steps.
4. We can also go into the **Edge** component mode (*Ctrl* + *Tab*), and select the edge loops that came from the different extrusions by placing the mouse pointer over them and pressing *Alt* and RMB.
5. After this, we can move them along the volume by tapping the *G* key twice.

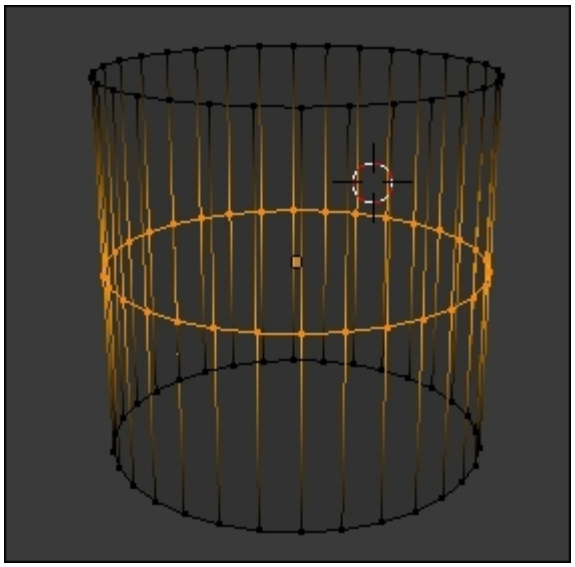


Shaping the head with extrusions.

Note

What is an Edge Loop?

An Edge Loop is a set of edges that are connected together and form a loop. You can also get face loops to follow the same principle but with faces. They are essential to construct the shape of an object.



Modeling the antenna

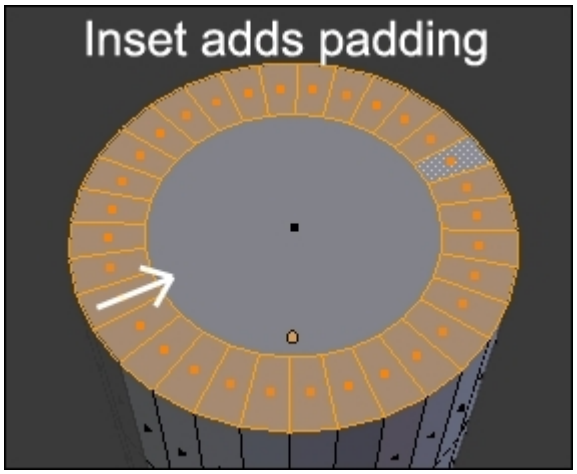
In order to create the antenna, we will start from the head and detach it later. Follow these steps to create the antenna:

1. In the head **Edit Mode**, we will select the top face and extrude it a little bit to make the base of the antenna.
2. Then we will make an inset from the top face of the base by pressing the *I* key.

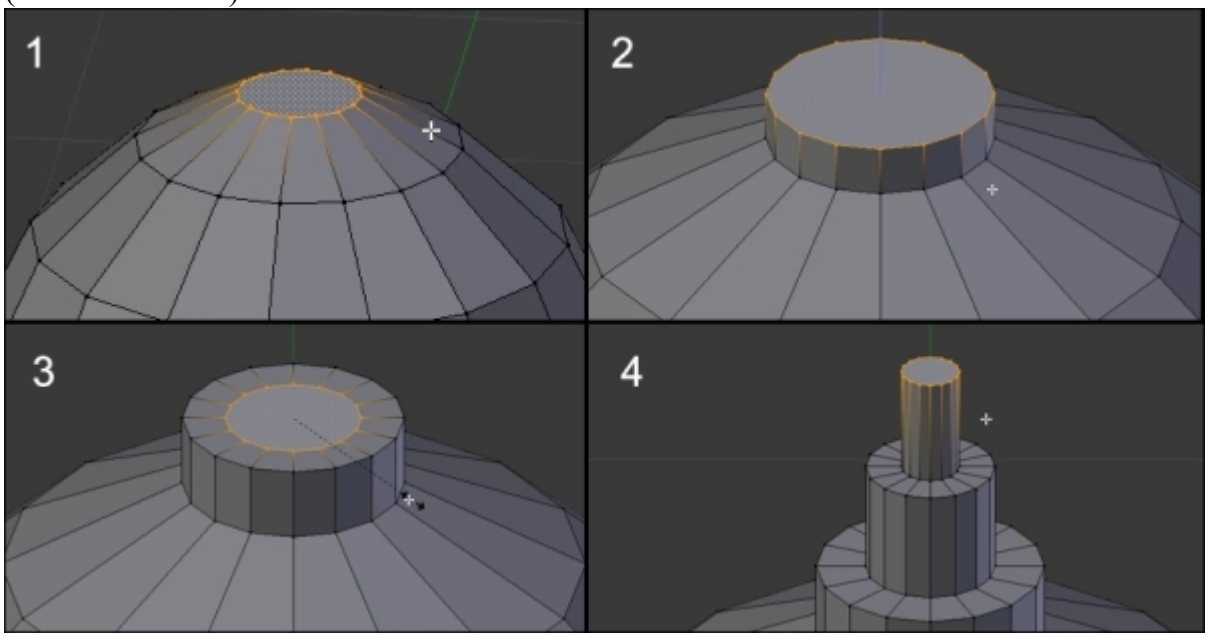
   ## Note

   What is an inset?

   An inset allows you to add some padding on a face:

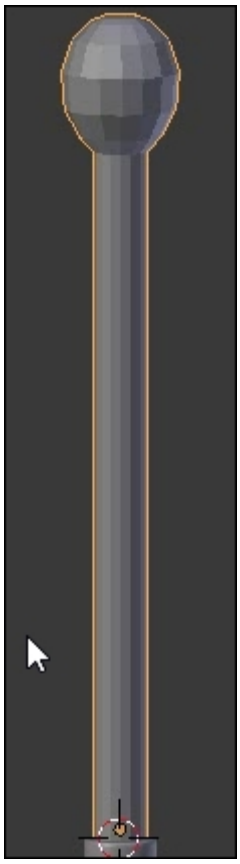


3. With the inner face of the inset selected, we are going to repeat the process one more time (extrusion + inset).



The different steps to model the base of the antenna. A succession of insets and extrusions.

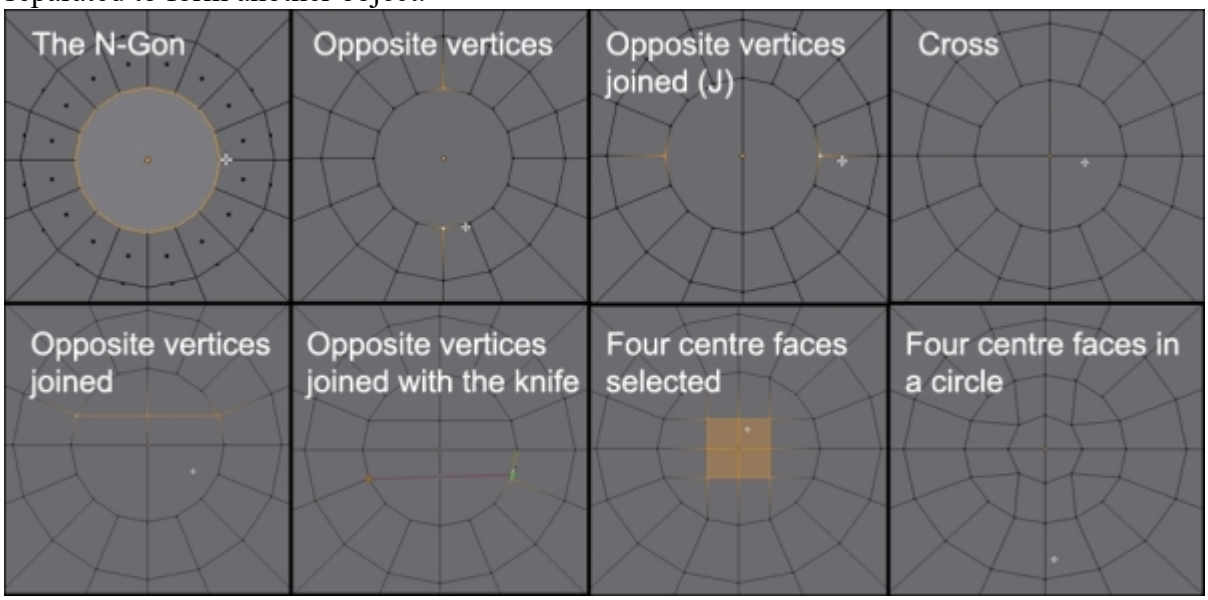
4. After this, we will add the stem of the antenna by moving the cursor to the top of the antenna's base, selecting the top face, pressing *Shift* + *S*, and selecting **Cursor to Selected**.
5. Now we can add a cylinder in the **Object Mode** that will pop up on the cursor. This cylinder will represent the stem, so scale it accordingly and end it with some extrusions that you will form in a sphere shape by scaling them and following the same process as that of head modeling.
6. You can also select the top part of the stem and use the smooth option (by pressing the *W* key and selecting **Smooth**) to relax the geometry.



The stem with the different extrusions that we have shaped like a sphere with the Smooth tool

7. We now have an **N-Gon** at the top of the stem. An N-Gon is a polygon that has more than four edges. It is considered a bad practice to have these kinds of polygons in a 3D object. We are going to solve this by going into the top view (the *7* numpad key) and by doing a small inset on the object in order to maintain the border.
8. After this, we will connect some vertices together in order to have only quads (polygons that have four sides).
9. Then, we select the two opposite vertical vertices by right-clicking on the first one and pressing *Shift* and right-clicking on the second one. Pressing *Shift* and invoking any selection method allows you to add new items to your current selection.
10. After this, we join them to a new edge with the *J* key (to connect the vertex path tool) to separate the N-Gon into two equal parts.
11. Now we ought to select the two opposite horizontal vertices and join them to form a cross. If you look closely, we haven't resolved the N-Gon problem yet, because we have four more of them.
12. As we can't leave them in the mesh, we are going to repeat the process by joining the other facing vertices in order to have only quads. If you want, you can also use the Knife tool in order to cut in the geometry by pressing *K*. With the knife we will have to click on the vertices that we want to connect together and when we finish, we can press the *Return* key in order to validate.

13. At this point, we can use the **LoopTool** add-on that we installed in the first chapter. We can select the four middle faces (in **Face Mode**) and use the **LoopTool** circle option (press *W* then select **LoopTool** | **Circle**). This allows us to form a circle with the selected components.
14. It's time to detach the antenna. In order to do this, we select the loop at the base of the antenna (press RMB and *Alt*) and press *V* to rip the loop. Blender will give us the choice of moving the ripped part, but we won't. So we cancel the move by clicking the RMB.
15. Now, we will detach the geometry of the antenna to form a new object. First we deselect all the components (*A*), then we move our mouse pointer over the antenna and press *L* to select the linked geometry.
16. After pressing the *P* key and choosing **Selection** in the pop-up menu, the selected part will be separated to form another object.



N-Gon correction with the Join tool and the Knife

17. We will have to clear three N-Gons: one at the bottom and at the top of the head, and one at the top of the base of the antenna. We have decided to resolve them with the previously explained method.

An introduction to the Subdivision Surface modifier

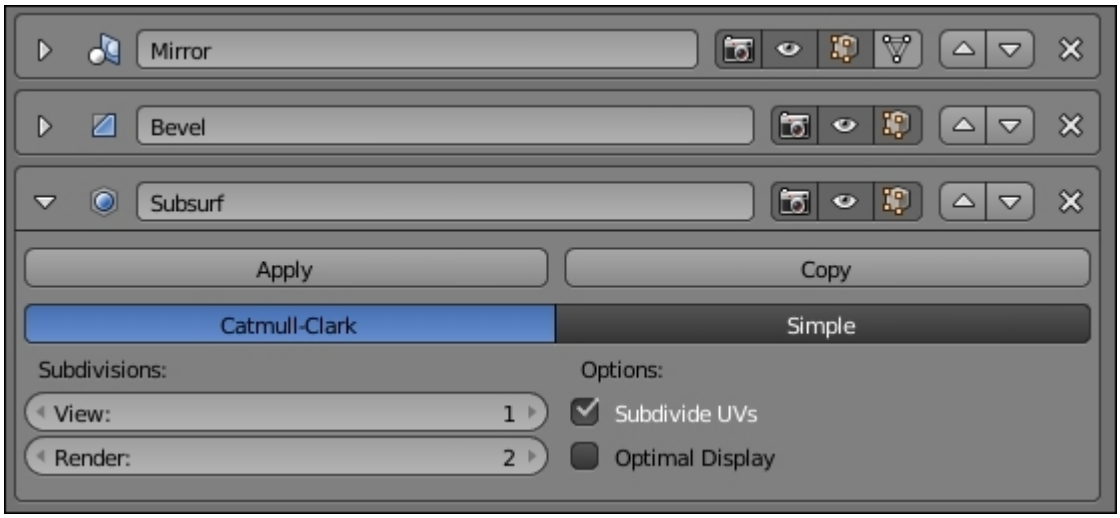
We will now smooth the geometry of the robot head and the antenna using the following steps:

1. First we go to the left 3D view panel (*T*), and with both objects selected in the **Object Mode**, we click the **Smooth** button under **Shading**. This will create a blend between the faces but not round our objects. In order to round our geometry, we will need to use a modifier called **Subdivision Surface**.
2. Let's go into the Properties editor and select the adjustable wrench. Then, we choose a **Subdivision Surface** modifier in the **Add Modifier** drop-down menu. All we need to do now is repeat the process with each object.

Note

What is a modifier?

A modifier is a tool that applies to the entire object. You can push new modifiers on the modifier stack of the object where the top modifier will take effect before the bottom one. You can also reorganize their order using the up and down arrows. You may hide a modifier using the Eye button. If you want to collapse a modifier, use the left-hand side horizontal arrow. You can also apply the behavior of the modifier with the **Apply** button. Always save your work before doing this.



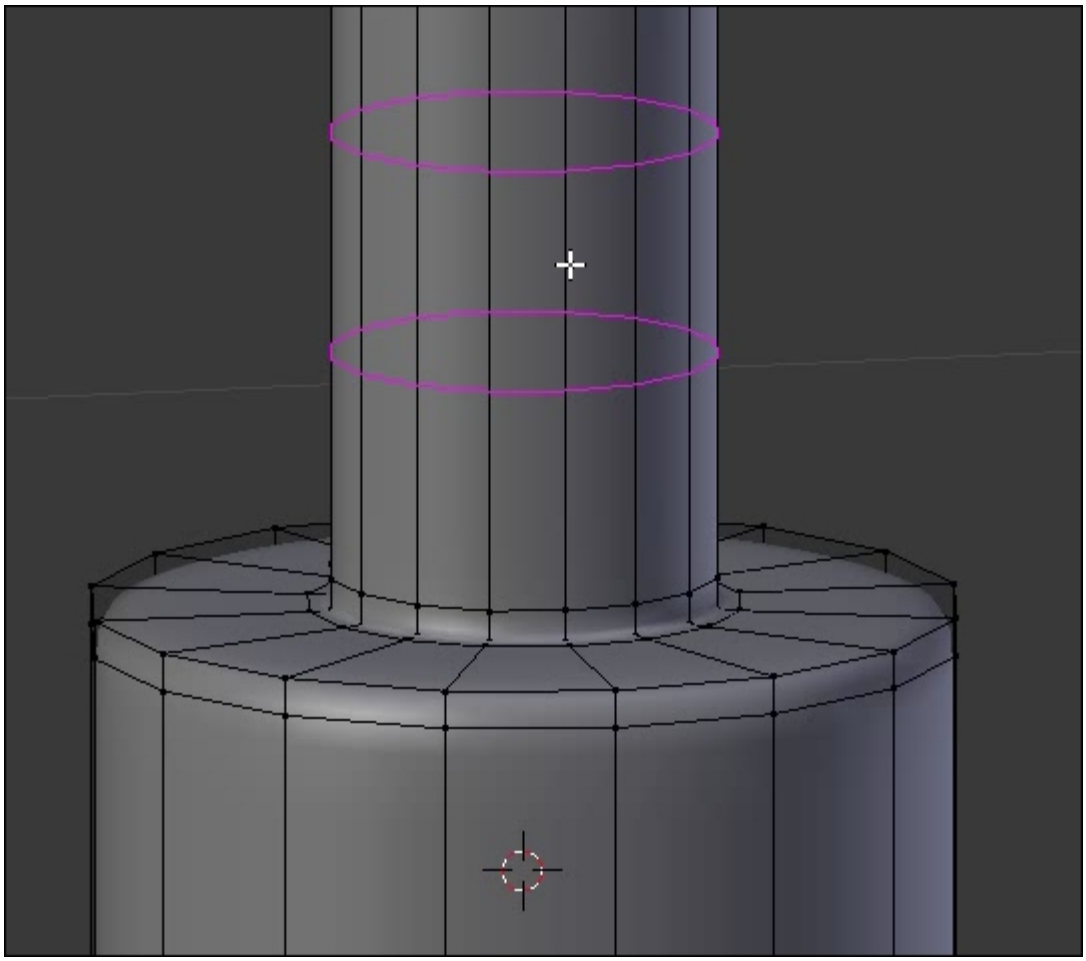
*The stack of three object modifiers. The **Subdivision Surface** applies over the Mirror and the Bevel modifier.*

3. As you may have seen, the subdivision divides all the polygons by four and tries to do an interpolation by smoothing them. If you want more divisions, you can increase the View slider under Subdivisions.
4. The shape looks better but needs to be sharp at some points. In order to do this, we will maintain a border by adding edge loops with *Ctrl* + *R*. The LoopCut tool is very useful; it allows us to add edge loops where we want and as many as we want.

Note

About the LoopCut tool

To add an edge loop, use the *Ctrl* + *R* shortcut and move your mouse cursor perpendicular to where you want to add a new edge loop. You will see a preview of the new cuts. You can add multiple loops at the same time by scrolling your mouse wheel or by pressing the + or – keys. After you have validated the cuts, you will need to position them and validate their location by left-clicking or right-clicking. The later will center the cuts.

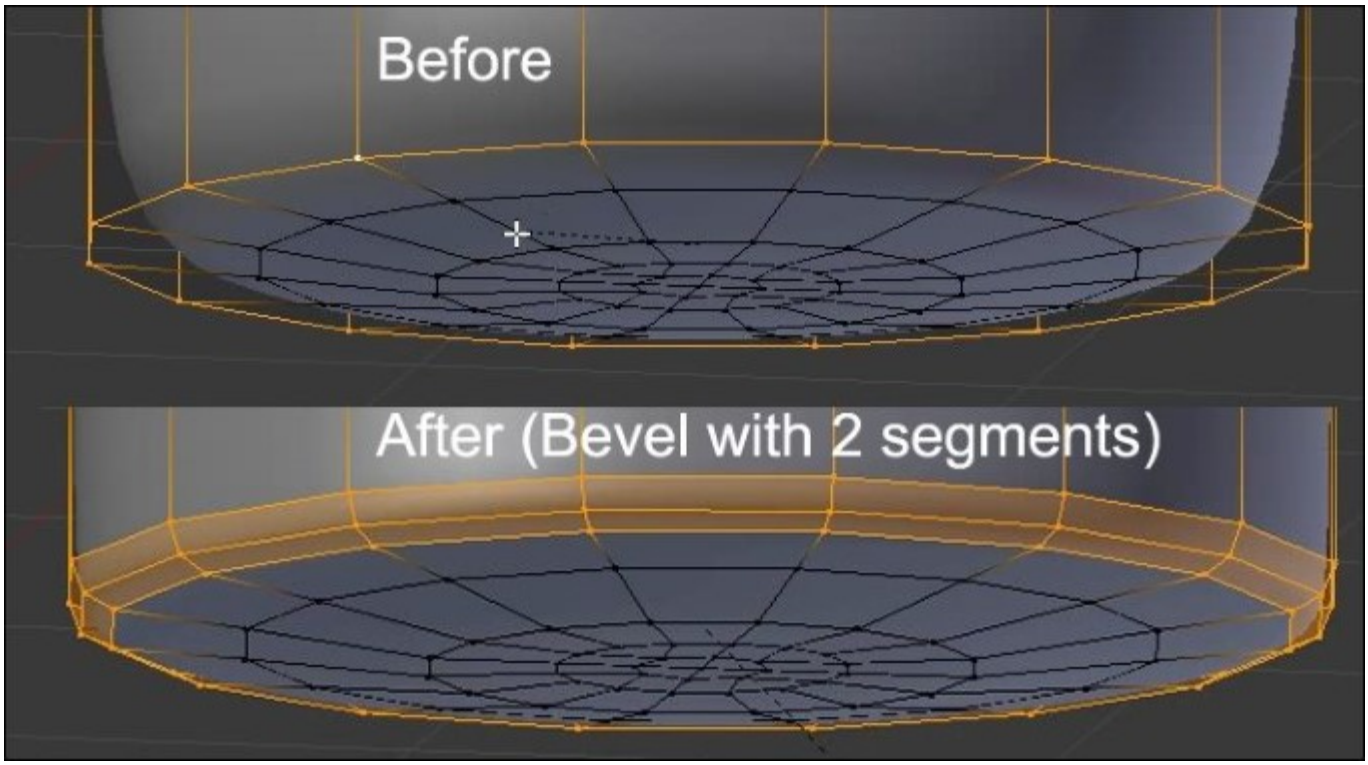


5. We can also sharpen the edges by selecting them and using the Bevel tool. Remember that the nearer the edge loops are, the sharper the result will be.

Note

About the Bevel tool

The Bevel tool allows you to split one edge into multiple edges. When you activate it with *Ctrl* + *B*, you can choose the number of splits that you want by scrolling your mouse wheel or by pressing the + or – keys. You can also decrease the speed of the tool by pressing the *Shift* key. As always, you can validate your placement by left-clicking or cancel it by right-clicking.



Improving the head shape

Let's select the head and go into the **Edit Mode**.

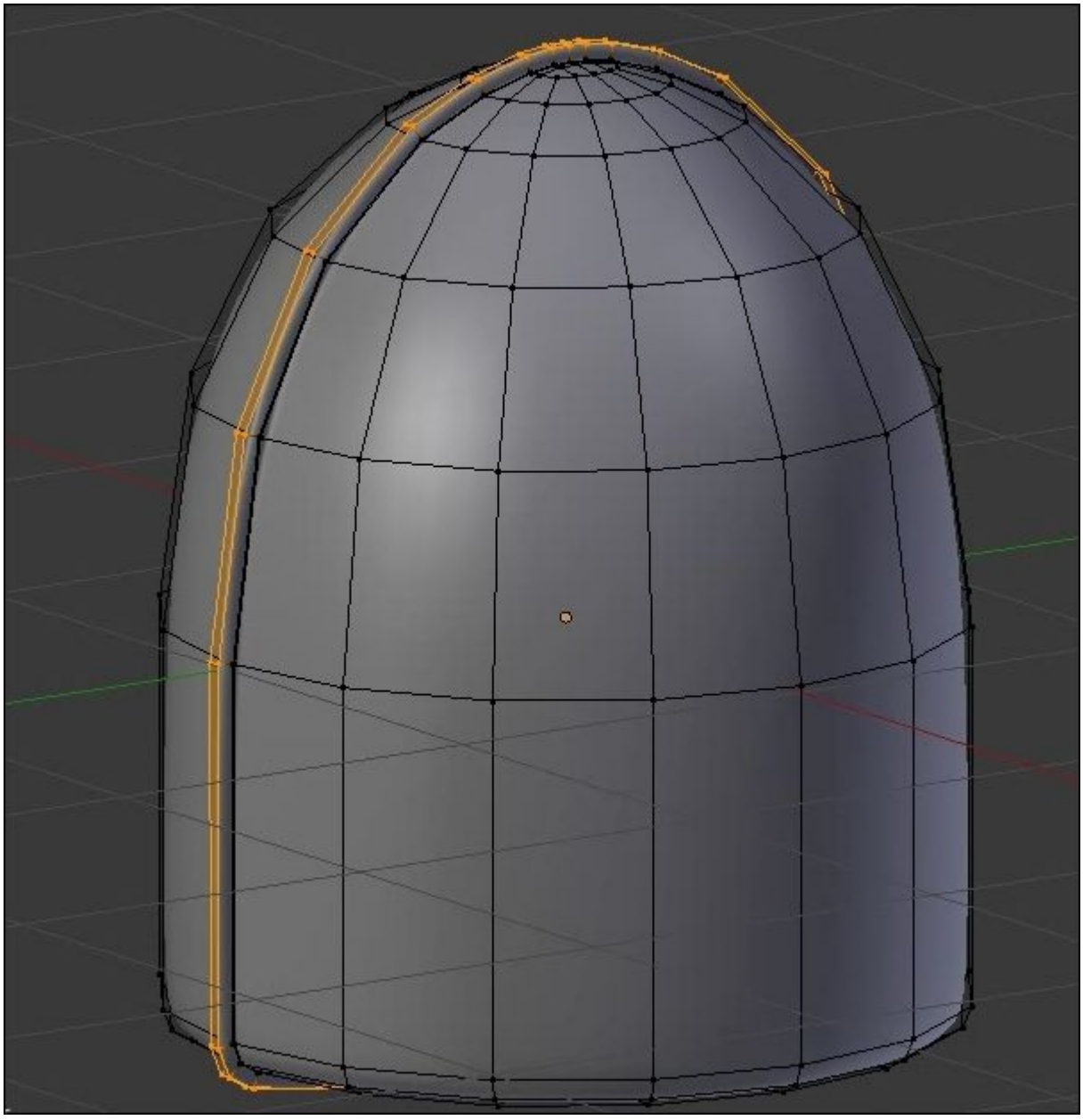
1. From the front view, we will now select the central edge loop. One way to do this is to use the wireframe mode by pressing the *Z* key. This mode allows us to see through the mesh and the selected components that are behind it.
2. We can now use the Box Selection tool with the *B* key in order to draw a rectangle area around the vertices that we want to select. If you want, you can hide the antenna and the stem by selecting them and pressing *H* (Hide). The Bevel tool will help you to create a thin base in the middle of the head. This bevel will be maintained with two new cuts. To do this, we can do an extrusion without moving it (cancel the move with RMB).
3. We can now scale the newly extruded faces on the *x* axis using the *S* + *X* shortcut. The thickness is added by selecting the inner face loop and extruding it at the same place.
4. In order to push the extrusion according to the normals, we use the *Alt* + *S* shortcut.
5. We will also have to maintain the shape of the head by adding multiple edge loops (with *Ctrl* + *R*, for instance).

Note

Save your work!

After all the work you've done, it's very important to save it! To write your blend file to your hard disk, go to the **File** menu and press the **Save** option or use the *Ctrl* + *S* shortcut. You can

now choose which directory you want to place it in. A nice trick is to press the + or – key to add or remove one unit from the name of your file.



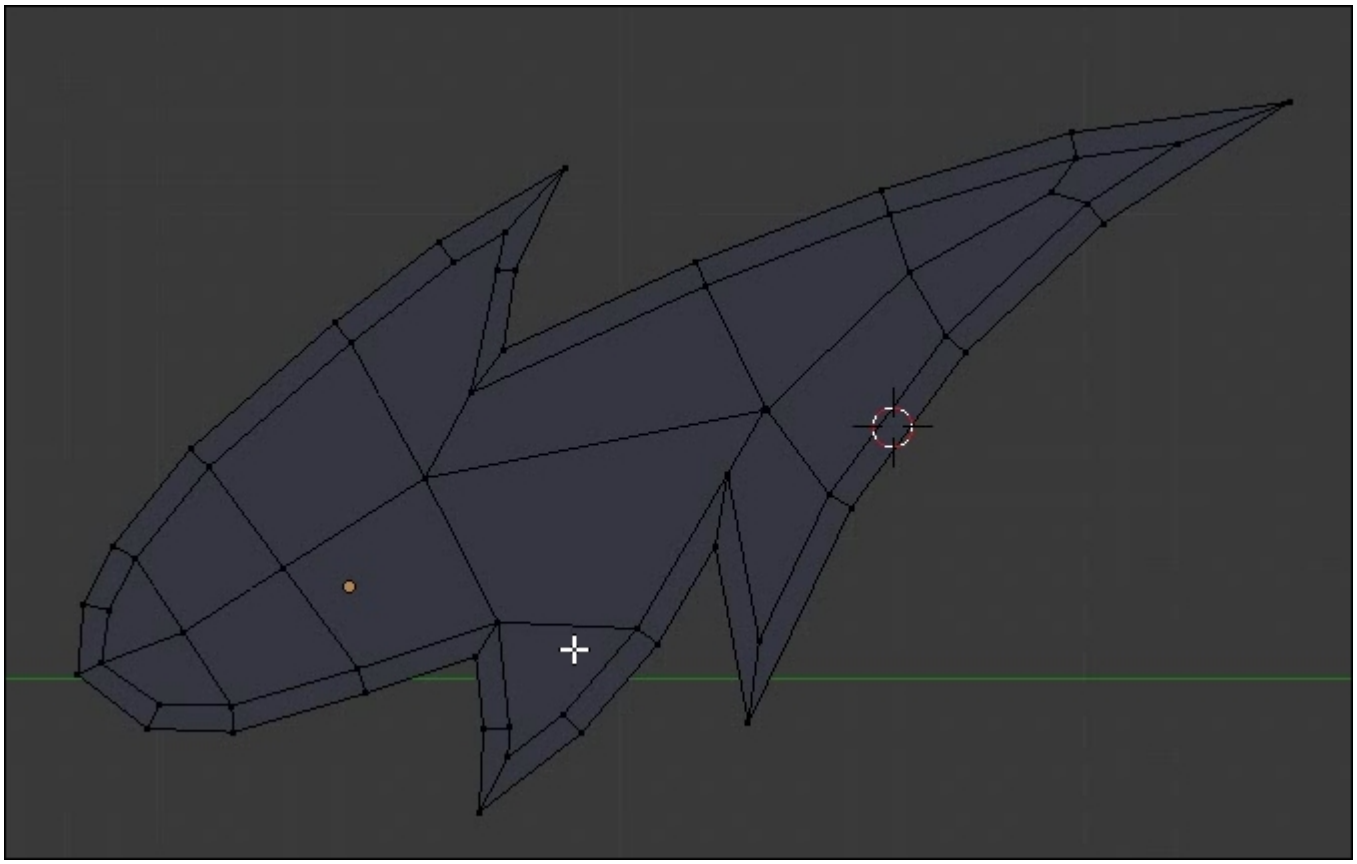
The head shape without the antenna.

6. You can unhide the antenna and the stem in the **Object Mode** by pressing *Shift* + *H*.

Modeling the thunderbolts

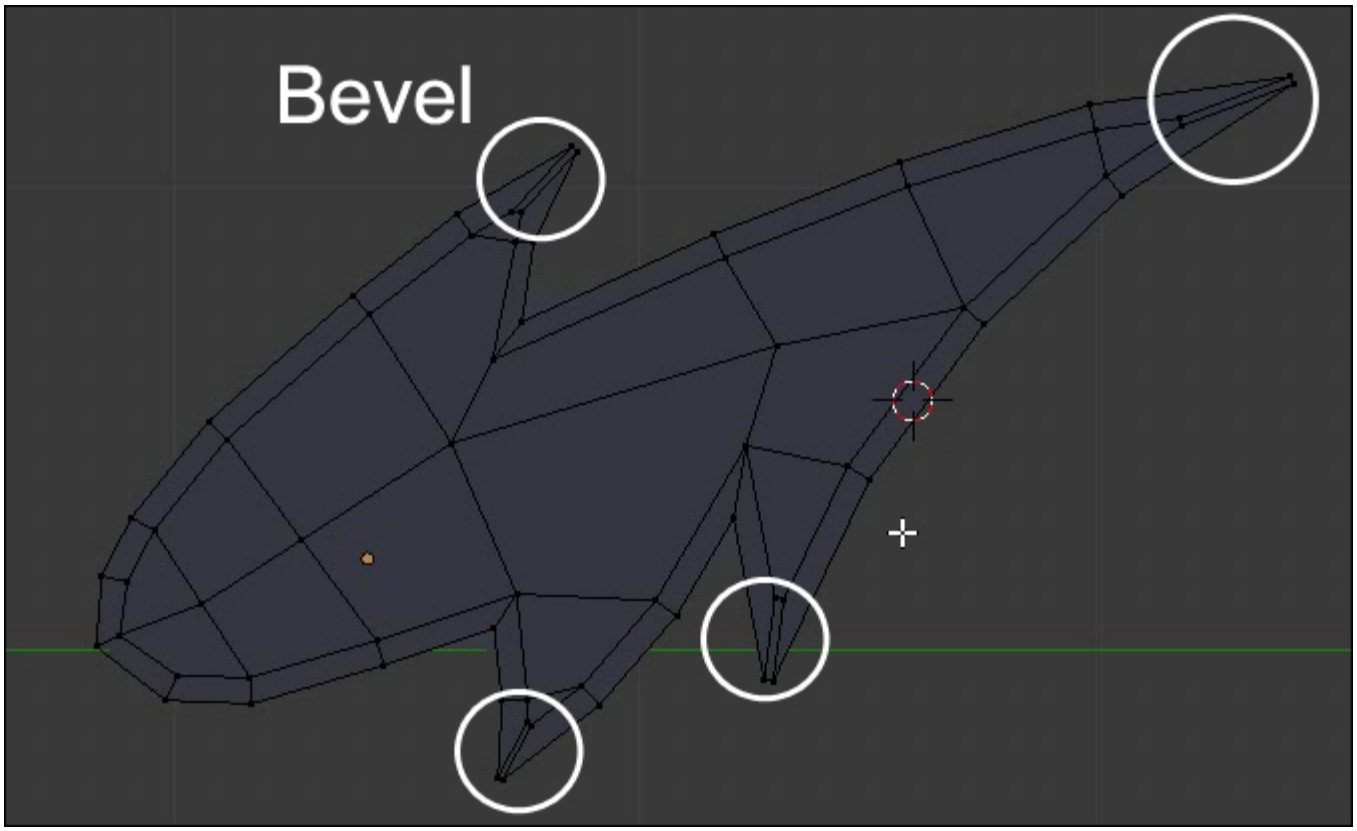
It's now time to start modeling the thunderbolts; let's have a look at the following steps:

1. We will start by going into the Side Orthographic view (the *3* numpad key) and by placing the cursor next to the head with a simple left-click.
2. Then we will add a plane and in the **Edit Mode** we will remove all the vertices with the *X* key.
3. In the **Edit Mode**, we are going create a chain of vertices that matches the thunderbolt shape of the image reference.
4. Pressing *Ctrl* and LMB, we will add new vertices and create the silhouette of the thunderbolt.
5. In order to close the shape, we select the first and last vertices and press the F key to fill them with an edge.
6. If you want to add more details to the shape, select two connected vertices and with the LoopCut tool (*Ctrl* + *R*), place a new vertex in the middle of both the connected vertices.
7. We can then select all the vertices (*A*) and fill the shape with an N-Gon (F) that we are going to resolve later.
8. We can now add an inset (I) in order to keep an outline.
9. After we've done this, we will have to clean the mesh by replacing the N-Gon with quads using the join tool (J) or the knife tool (K). If you have one triangle or N-Gon, it's not a problem for now as it could be solved later.



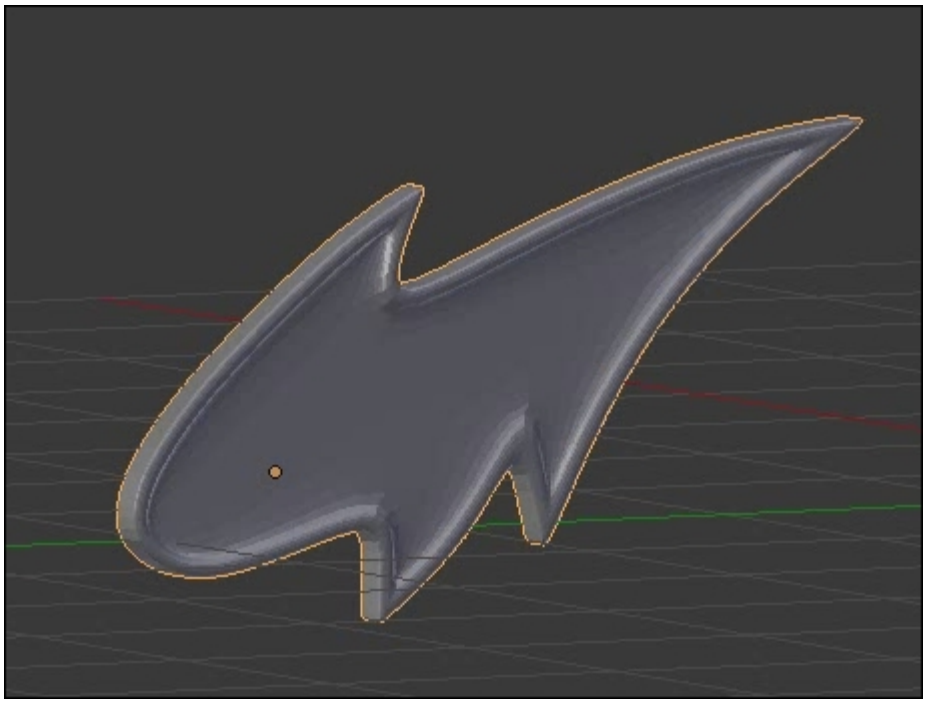
The thunderbolt shape.

10. We can now add a **Subdivision Surface** modifier in the **Object Mode**.
11. We will have to sharpen the spikes using tight bevels.
12. Of course, we will have to clean the mesh by removing the N-Gons.



Maintaining the spikes with bevels.

13. It's now time to extrude the whole thunderbolt by selecting all the faces (*A*). You may get a lighting error with black faces. It means that the normals are pointing inward and can't catch the light. You can verify this by opening the right panel of the viewport (*N*) and, under Normals, you can check the face icon. If the normals are not pointing outward, then you will need to recalculate their direction by selecting all the components and pressing the *Ctrl* + *N* shortcut.
14. We can now select the inner faces on the outside of the thunderbolt with either *Shift* + RMB or using the *C* key, which allows you to paint and select the components that you want according to the current view. With these faces selected we can create a small inner extrusion, and maintain the shape with the LoopCut tool (*Ctrl* + *R*).



The finished thunderbolt with a view 2 ***Subdivision Surface****.*

15. In order to mirror the thunderbolt on the other side of the head, we will use a **Mirror** modifier with the head as the center of a pivot. Place and rotate the thunderbolts according to the image reference.

Note

The Mirror modifier

This is an easy way to make a symmetry from your 3D model. The basic symmetry is based on the positioning of the pivot point and the *x* axis. All of these are configurable by changing the axis. It is strongly advised you use the **Clipping** option if you want to weld the components that are on the symmetry axis of your geometry. By doing this, you will avoid holes. With the **Mirror Object** option, you can choose to base the symmetry axis on the pivot point of another object in your scene.

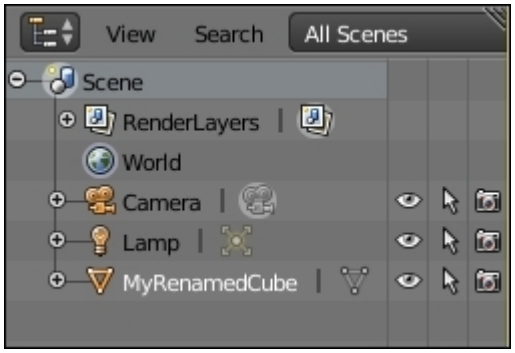
16. The last thing we may want to do at this stage is to correctly name our objects in the outline editor that is situated in the top-right corner of the interface by default.

Note

The outliner

The outliner displays a list of all the entities that make up the current scene. When you select an object in the 3D view, it will be highlighted in the outliner and conversely. You can rename any item in the list by double-clicking on its name. The outliner also gives you control of the

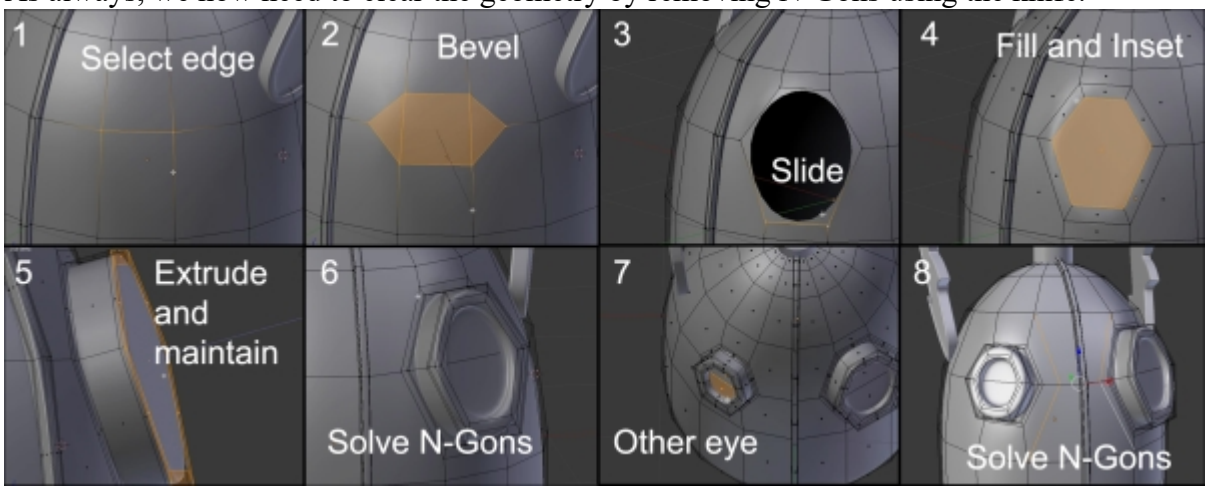
visibility of any object with the Eye icon button. The mouse cursor button can be toggled on or off to allow the selection of the corresponding object in the viewport.



Modeling the eyes

It's time to finish the head of our robot by adding a pair of eyes on it. This is done with the following steps:

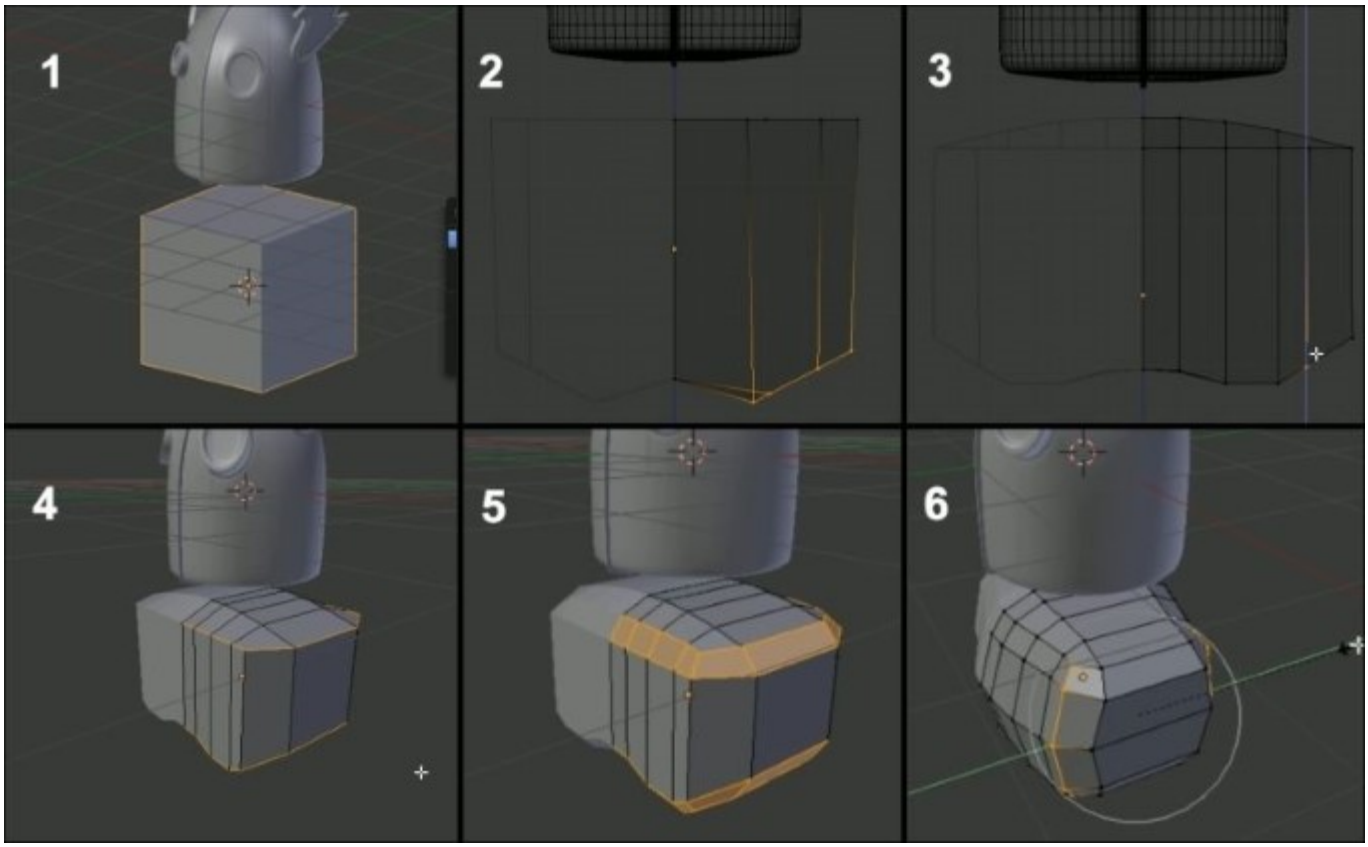
1. In the **Edit Mode** of the head, we select the edge that is roughly positioned at the eye location.
2. We use a bevel to add more geometry.
3. It is now possible to slide the top and bottom edges of the bevel according to the volume by selecting them in the **Edge Mode** and pressing G twice to form an ellipsoidal shape.
4. After this, we can use the *F* key to fill the eye. It will remove the two vertical edges that come from the bevel.
5. Now we can do a series of insets and extrusions to pop out the eye.
6. Applying the same method to the other side, we will add a little cartoon effect.
7. As always, we now need to clear the geometry by removing N-Gons using the knife.



Modeling the chest

It is time to model the chest. The following steps will help you do so:

1. Now we put the 3D cursor at the center of the space (*Shift* + *S*), then we add a box (*Shift* + *A*) for the base shape of the chest.
2. We can adjust the position under the head (*G* + *Z*) in the front view, and switch to the **Edit Mode** to start the modeling.
3. It is much faster to work with the symmetry, so we are going to cut the cube at its center with an edge loop (*Ctrl* + *R*). We select all the vertices on the left-hand side with the box select tool (*Ctrl* + *B*) in the wireframe shading mode (*Z*) and we delete them (*X*).
4. In the object mode (*Tab*), we add a mirror modifier to work with symmetry.
5. More polygons can be added to create the basic shape of the chest. We, therefore, add two vertical edge loops (*Ctrl* + *R* and scroll the mouse wheel up) from the side view and slide these on the *y* axis (*S* + *Y*).
6. Next, we move the polygon situated at the center of the chest from the side view (*G* + *X*) and add another vertical edge loop from the front view.
7. In the orthographic (*5*) front view (*1*), and with Wireframe (*Z*) Shading activated, we can easily work the shape by moving (*G*) and rotating (*R*) the selected vertices (refer to **2** and **3** in the following screenshot). Then we move the top face up a little (*G* + *Z*).
8. We can see that our central edge loop is not very well aligned in the front view. Therefore, we select and replace it by applying a zero scale on the *x* axis (*S* + *X* + *0* on the numeric keypad), which perfectly aligns our selected vertices.
9. In order to have a less angular shape, we are going to activate the edge mode and select the edges at the top and bottom of the chest (press *Shift* + *Alt* and the LMB) to finally do a bevel (*Ctrl* + *B*) (refer to **5** in the following screenshot).
10. To balance the polygon flow of the mesh, we add a horizontal edge loop to the center (*Ctrl* + *R*) that we will extend with the Scale tool on the *y* axis (*S*+*Y*) (refer to **6** in the following screenshot). The goal is to use the maximum area of the drawing as a reference. Always remember to check the silhouette of your object.
11. For a more curved shape, we will use **Proportional Editing** that can be activated by the small circle icon located at the header of the 3D view.
12. The **Proportional Editing** tool will help us to shrink the side of the bust. Be careful to check the behavior of the clipping option of the mirror modifier for the vertices located on the axis of symmetry. They don't have to be merged at the center. To smooth the model, we will add a **Subdivision Surface** modifier and will check the **Smooth Shading** option in the **Transform** panel (*T*) in order to remove the flat aspect of the polygons.



Note

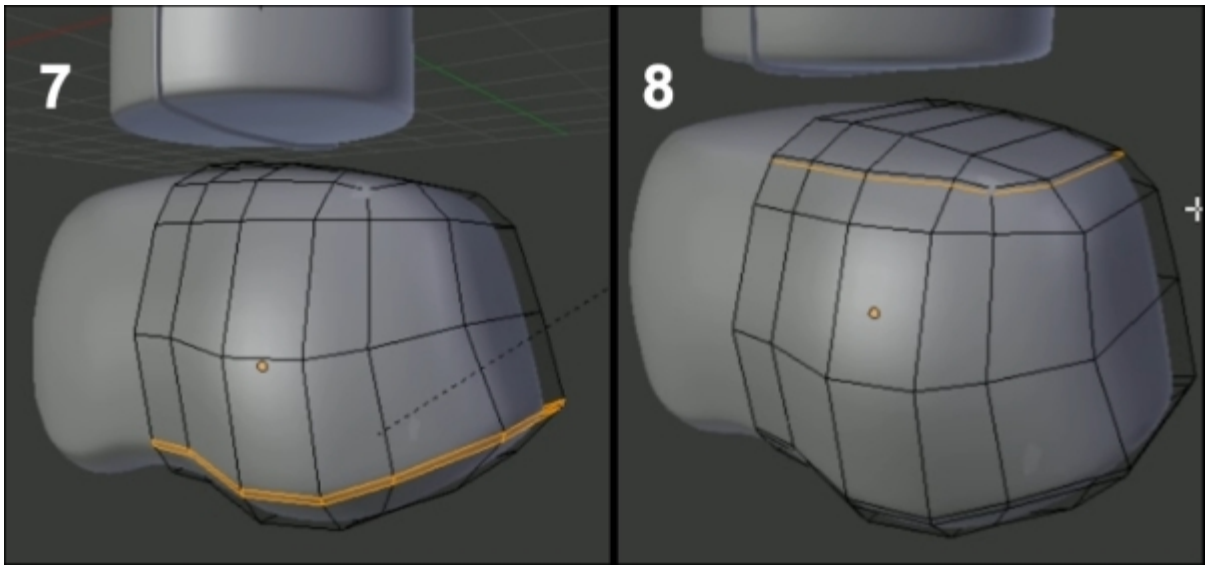
The Proportional Editing tool

This allows you to change the shape of an object in the **Edit Mode** in a smooth manner. It acts like a magnet for unselected components that are inside a circle of influence that you can adjust by scrolling the mouse wheel. This is perfect when you want to move a set of components and when the geometry is too compact.

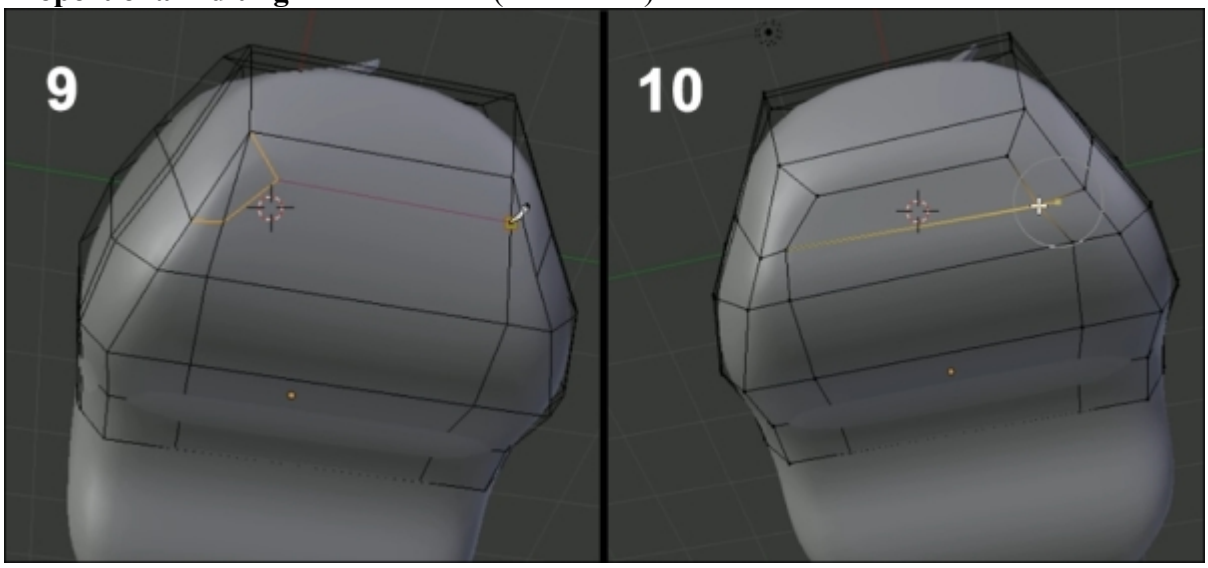
The **Connected** option allows you to limit the scope to the geometry connected to the selection.

The **Falloff** option offers a series of attenuation curve profiles.

13. With some bevels and additional edge loops, we will just sharpen some curves (refer to **7** and **8** in the following screenshot).



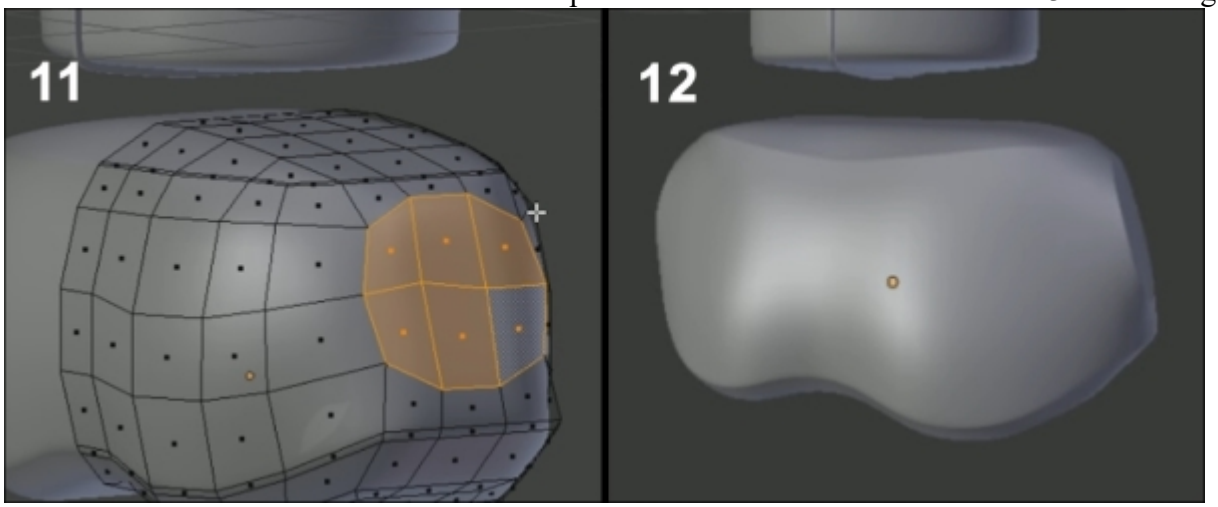
14. To flatten the top a little, we will select the highest faces and scale them on the Z axis with the **Proportional Editing** tool turned on. (S+Z and O).



15. From the bottom view, we use the Knife Topology tool (*K*) to change the organization of our vertices, edges, and faces (refer to **9** and **10** in the preceding screenshot). The process of arranging the components in the order that they best fit the shape is called "searching for a good topology". An easy way to grasp good topology is to remember that the edge loops are going to wrap around the object you want to create. Another thing that we already talked about is using only quads because they are easier to manage.
16. We form a loop to allow a better subdivision of the surface. Other edge loops can be added again horizontally and vertically to add details.
17. Then we switch to **Face Mode** and select six faces on the side of the bust in order to round them off with the LoopTools **Circle** feature (press *W* and select **LoopTool** | **Circle**) (refer to **11** in the

following screenshot). We can choose the influence of this tool at the bottom of the left panel (*T*). A value of 80 percent will best fit here.

18. Then we adjust the angle of these faces with a rotation on the X axis (*R* + *X*). To quickly select polygons, we advise you to use the Circle Select (press *C* and the LMB) tool, which is used like a brush.
19. We do a small inset (*I*) to sharpen the geometry at the location of the shoulders.
20. We continue to move some vertices (*G*) here and there to gradually round the shape, and we adjust sets of vertices by rotating them (*R*). It is important to always navigate around your object to have the correct silhouette from different points of view. This is the essence of 3D modeling.

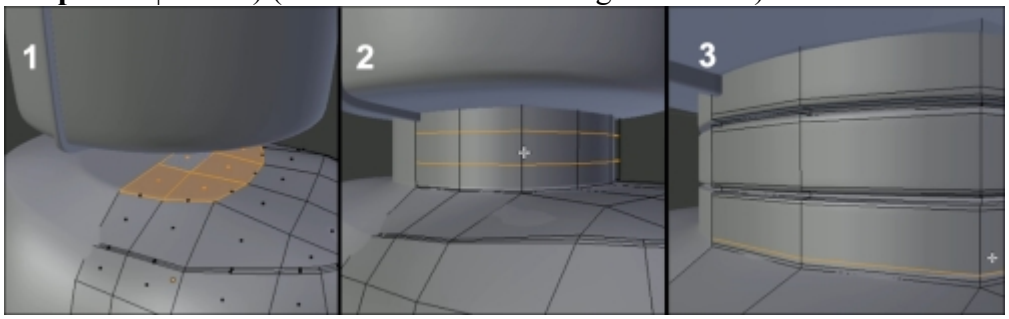


21. If you want more sharp edges, you can add a few edge loops (*Ctrl* + *R*) around them. Remember to use the smoothness parameter of the LoopCut tool (a value ranging from 0 to 1) in the left panel (*T*).

Modeling the neck

Still working on the bust in the **Edit Mode**, we will again flatten the top part of the bust and more precisely the area.

1. We start with the neck by selecting six polygons and scaling them on the *z* axis (*S* + *Z* + *0* on the numeric keypad).
2. These faces will be arranged in a circle with the LoopTool Circle (by pressing *W* and selecting **LoopTools** | **Circle**) (refer to **1** in the following screenshot).

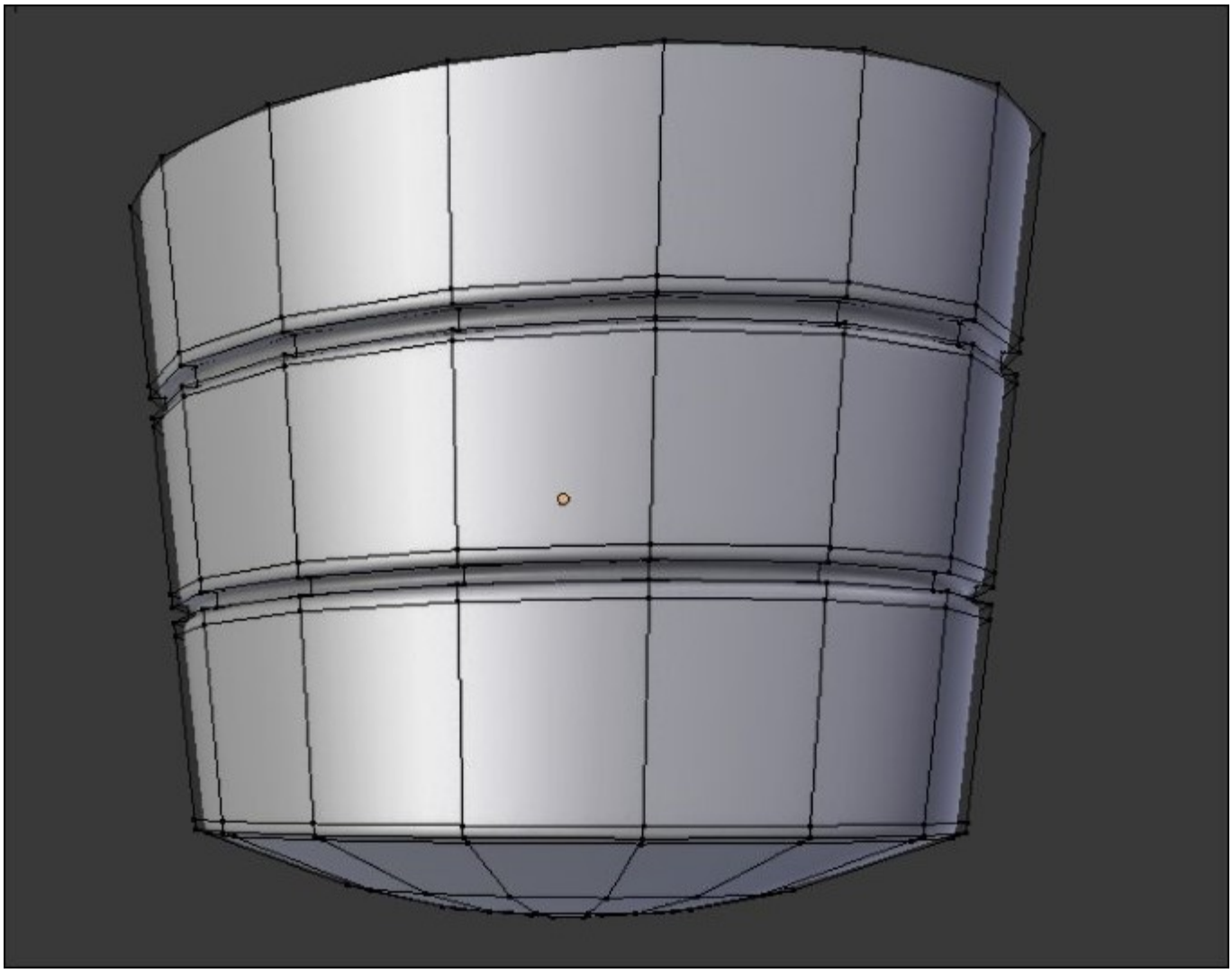


3. Then we make a very light extrusion (*E*) to hold the lower part of the neck.
4. Afterwards, we continue with another extrusion that penetrates in to the head.
5. In the Wireframe Shading mode, we remove the nonvisible face that is inside the head, which is useless.
6. Then we add two loops cuts (*Ctrl* + *R* and press MMB) (refer to **2** in the preceding screenshot) that we will divide into thinner edge loops with a slight bevel (*Ctrl* + *B*). These newly created face loops will be extruded (E) and scaled on the *x* and *y* axes (*S* + *Shift* + *Z*) (refer to **3** in the preceding screenshot).
7. As always, we will maintain the shape by adding edge loops with the LoopCut tool (*Ctrl* + *R*).
8. We end with a small extrusion (*E*) at the bottom to get the neck demarcation.

Modeling the torso

We will now work on the torso, which shares essentially the same modeling techniques as the neck. This will be done as follows:

1. We now snap the cursor on a vertex that lies on the symmetry axis of the lower part of the chest by opening the Snap floating menu with *Shift* + *S*. We select the fourth option.
2. We add a new cylinder (*Shift* + *A*) with 16 vertices (the number of vertices could be changed in the left panel by pressing *T*). By choosing this number of vertices, we get a cylinder that can be mirrored at its center and that has enough faces. We place the cylinder under the chest and we remove the top face that is not visible.
3. Next, we select the top edge loop and we change its scale on the *x* axis (*S* + *X*) and the bottom one on the *y* axis (*S* + *Y*).
4. We add two edge loops (*Ctrl* + *R*), add a bevel (*Ctrl* + *B*) to each of them, and finally, extrude (*E*) them inside (refer to the neck section).
5. We will round the lower part of the torso with a series of extrusions.



6. To get a smooth surface, we again need a **Subdivision Surface** modifier with the **Smooth Face Shading** option (Left panel: *T*).
7. We sharpen the edges with some edge loops (*Ctrl* + *R*).
8. Then, we clear our topology by solving the N-Gon with the same technique that we've used for the head.

Modeling the buttons

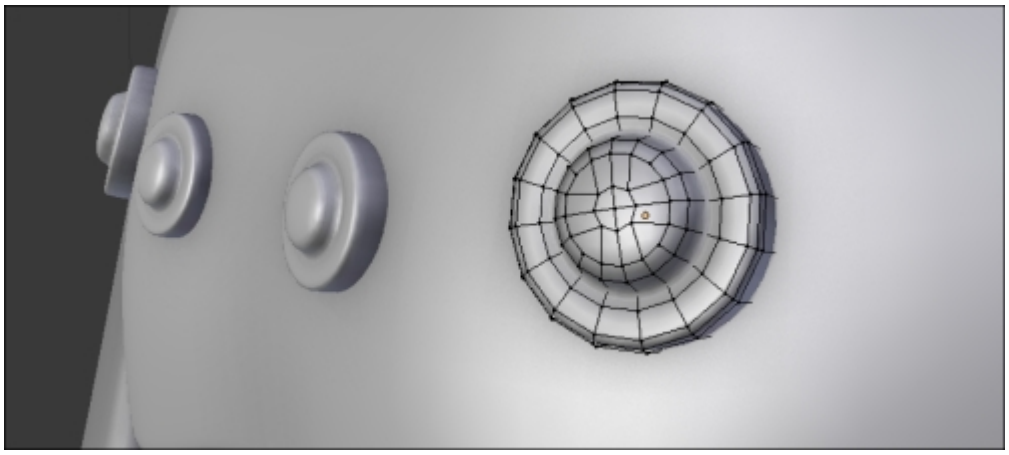
If you have followed the techniques used previously, the buttons are quite easy to make. The following steps are used to create the buttons:

1. We add a cylinder (with 16 vertices again), which we adjust in size and rotation with the Scaling (*S*) and Rotating tool (*R*). You can have a Free Rotation in all axes by pressing the R key twice. Don't forget this tool; it is very useful for aligning objects from a certain point of view!
2. We define the shape with a bevel at the top and an inward and outward extrusion.
3. We sharpen the shape with the LoopCut tool (*Ctrl* + *R*) and the Bevel tool (*Ctrl* +*B*).

4. The basic shape of one button can be achieved with only these tools (Extrusion, Bevel, and LoopCut).
5. In the **Object Mode**, we duplicate this with the Duplicate Linked tool (*Alt* + *D*).

Note

The Duplicate Object (Shift + D) and the Duplicate Linked (Alt + D) tools



These both duplicate objects or components. The Duplicate Linked tool creates an instance of the object while in the **Object Mode**. The mesh data are connected. It means that, in the **Edit Mode**, any change of geometry will be reflected on the linked objects. However, the transformations done in the **Object Mode** are not reflected. If you want to break the link between two linked objects, press *U* (in the **Object Mode**) – **Make Single User** – **Object & Data**.

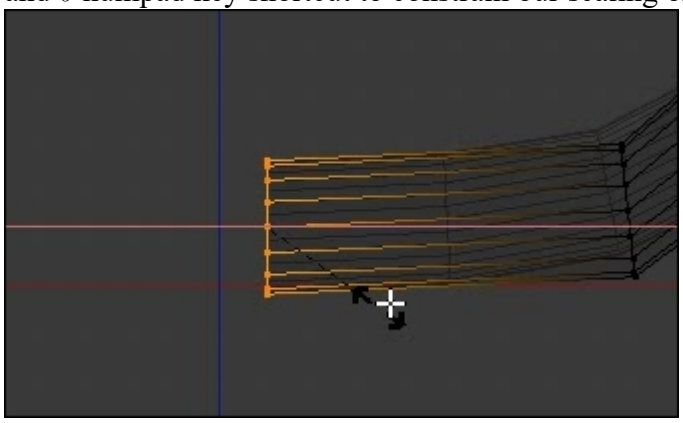
6. We will add a mirror modifier on each button. In the **Mirror Object** option, we select the torso. It will serve as the origin for the symmetry.

Modeling the fork

Now that we have finished the body, we will model the fork that covers the wheel, with the following steps:

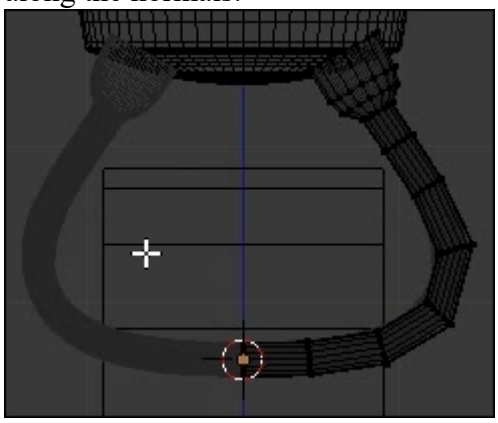
1. In order to do this, we place the cursor to the right in the front orthographic view and add a cylinder (*Shift* + *A*).
2. In the **Active tool** options, we change the **Cap Fill Type** to **Nothing**. Our cylinder will have holes at the bottom and at the top.
3. Then we scale it in the **Object Mode** and rotate it with the *R* key, always in the front view for greater precision.
4. We can now add a **Subdivision Surface** modifier and apply **Smooth Shading**.
5. As always, we will maintain the shape with edge loops in the **Edit Mode**.
6. We select the outer edge loop and, pressing *Ctrl* and LMB, we extrude the fork in a twisted arch manner. This is the same tool that was used for the thunderbolt creation.

7. The last edge loop should be flattened on the *x* axis. To do this, we select it and press the *S* + *X* and *0* numpad key shortcut to constrain our scaling on the *x* axis and give it a value of 0.



Flattening the last (inner) edge loop.

8. We will mirror the half fork to the other side using a mirror modifier. But if we do it right now, we will have a problem with pivot point placement. We have to move the pivot of our object to the same location as our body. To do this, we select the body and using the *Shift* + *S* command, we select the fourth option, **Cursor to Selected** (Note that, in any floating menu, you can choose the option that you want by typing the corresponding key on your numpad).
9. Now that the cursor is placed at the pivot point of the body, we will map the origin (also called pivot) of our fork by selecting it and going to the **Object** menu in the 3D view header and selecting **Transform** | **Origin to 3D Cursor**. You can also get a pop-up menu with the same options as that of the **Transform** menu with the *Ctrl* + *Alt* + *Shift* + *C* shortcut (one of the longest in Blender's history). Our origin is at the same location as that of the body.
10. Now, we apply the rotation by pressing *Ctrl* + *A* and selecting **Rotation**. Applying the rotation is important here because we changed it in the **Object Mode**.
11. We can now safely add a **Mirror** modifier.
12. At this point, we will add a temporary cylinder that will represent the wheel. This will help us to correctly place the fork.
13. If you want to adjust the thickness of the fork tube, use the *Alt* + *S* shortcut to push the polygons along the normals.

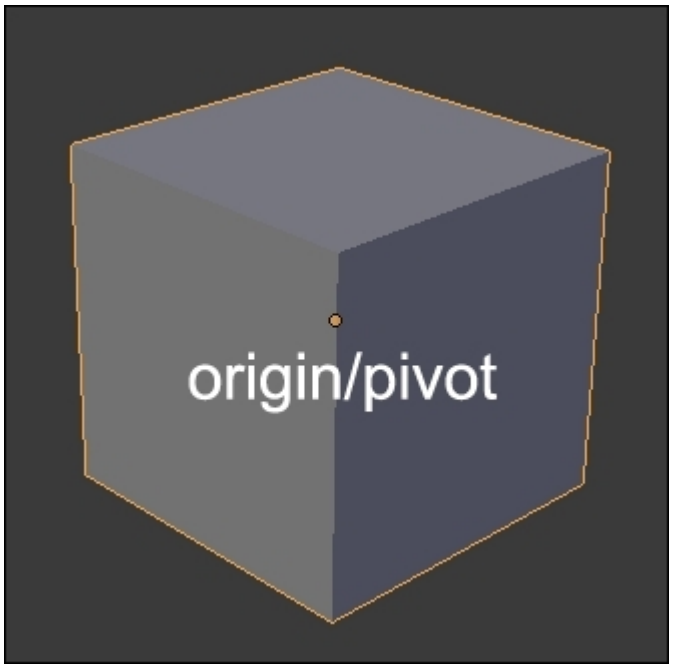


*The fork in the **Edit Mode**, with its mirror modifier and the temporary wheel.*

Note

About the origin/pivot

The origin, also called the pivot, is represented with a small origin circle in Blender. It determines the center of the mass of an object. Any rotation or scale modifications will take the origin into account by default. You can change the way these transformations work by using the **Pivot Point** drop-down menu in the 3D View header (next to the **Shading** drop-down menu).



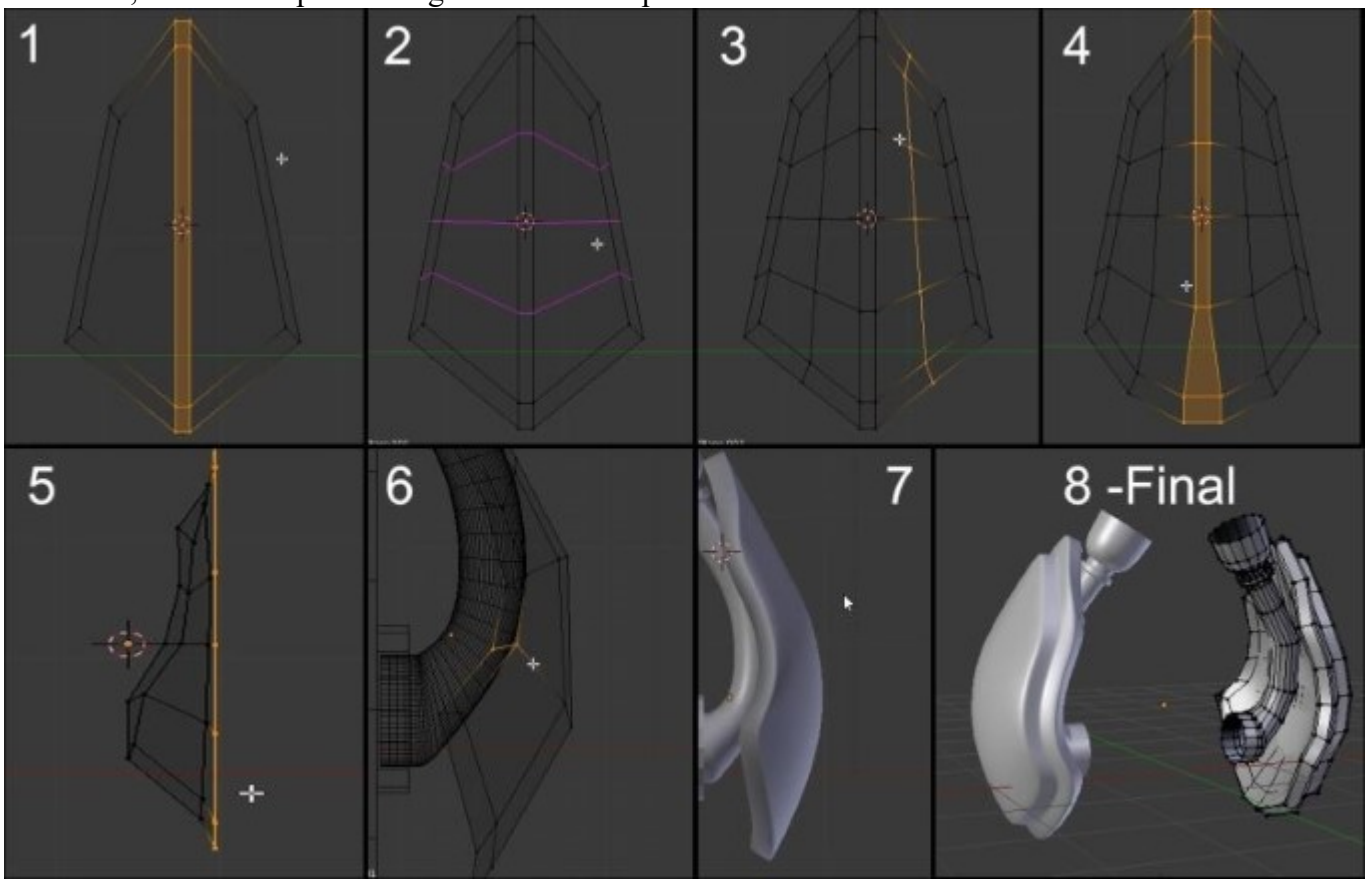
We will revisit the fork later. It's now time to create protections that cover it.

Modeling protections for the fork

In order to model this piece, we go inside the view by pressing the *3* numpad key and perform the following set of operations:

1. We add a plane, and in the **Edit Mode**, do an inset.
2. After this, we add a loop cut in the middle with *Ctrl* + *R* and scale it on the *z* axis by pressing *S* + *Z*.
3. With the two outer vertices of the top selected, we do a scale constrained on the *x* axis. This will give us a pointed shape.
4. After this, we do a bevel of the center edge loop and add loop cuts horizontally and vertically.

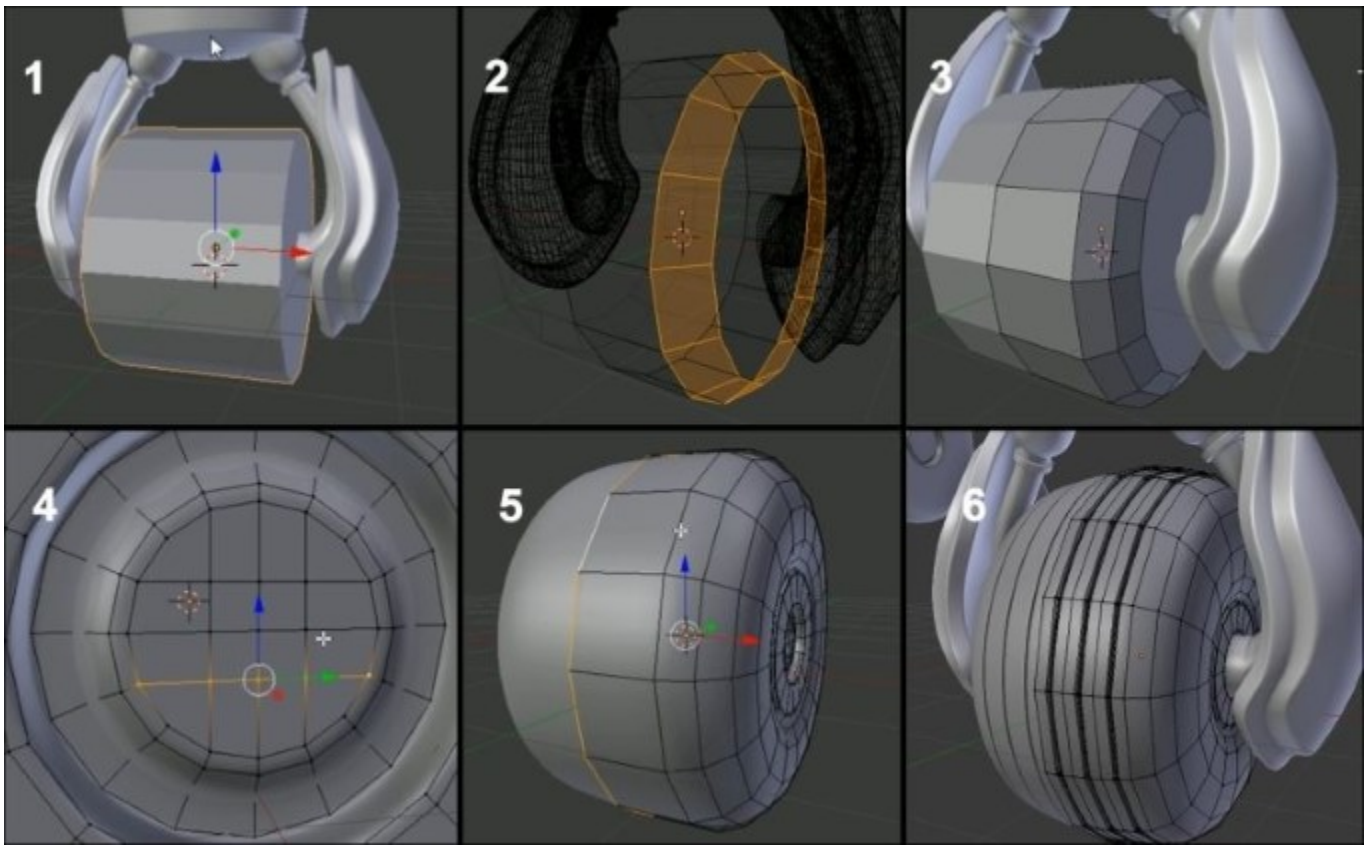
5. We will then round the lower part of the shape. While the two middle edge loops are selected, we go inside the orthographic view and, with **Proportional Editing** (*O*) using a sphere curve, we move the vertices back to round the shape.
6. We can now place the piece near the fork in the **Object Mode** and, with the wireframe shading activated (*Z*) in the **Edit Mode**, we can adapt its silhouette by following the fork shape.
7. It's now time to use **Smooth** shading and the **Subdivision Surface** modifier in the **Object Mode**. We will add a new modifier that will add thickness to the object as if we were extruding it entirely. This modifier is called the **Solidify** modifier. You can tweak its **Thickness** slider to change the amount of thickness that you want. This modifier will be placed under the **Subdivision Surface** in order to be applied to it. If you now go into the **Edit Mode**, you will see that it has added a new geometry.
8. You can maintain the newly added thickness with loop cuts.
9. We added some details on the side using the Inset tool and by extruding the created faces.
10. We will now combine protections with the fork. To do this, we first select the protection and then the fork, and by pressing *Ctrl* + *J* we will join them in one mesh. As a result, the protection is mirrored because it is inside an object that has a mirror modifier. Note that, if you reverse the order of selection, you will join the fork in the protection. That's not what we want.
11. Going back to the fork, we can add decorations to it by adding two edge loops near the top.
12. We can extrude the face loop between these edge loops and scale them according to the normals with *E* and *Alt* + *S*.
13. Of course, we can sharpen the edges with the LoopCut tool.



The process of modeling the protections and the final result with the fork.

Modeling the main wheel

We will start modeling the wheel with the temporary cylinder that we have placed in the fork section. There are many methods to do this. We will do this here with the same tools that we introduced to you before.

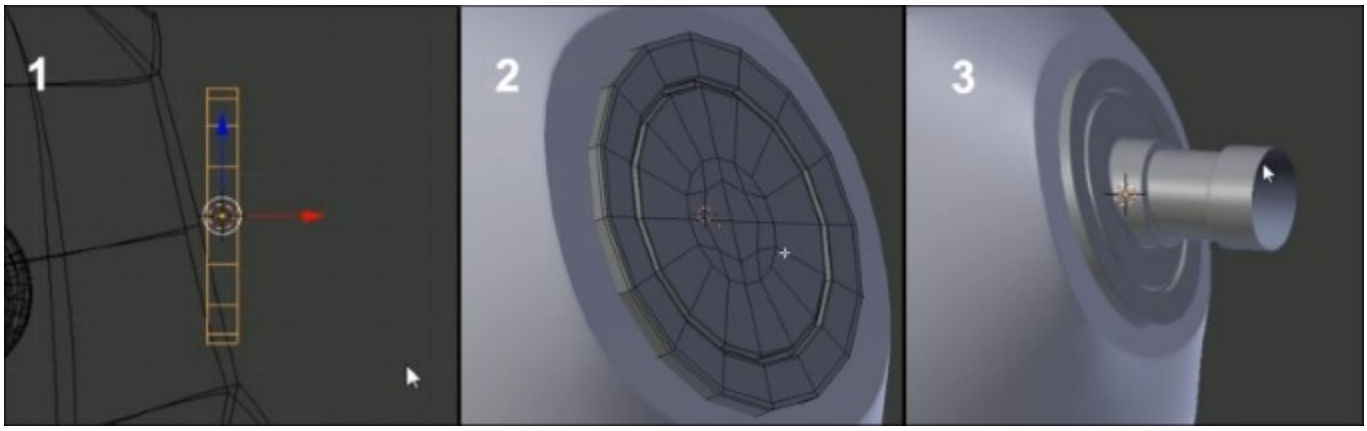


1. We will resize the wheel by enlarging it on the *y* and *z* axes (press *S* + *Shift* + *X*). When you press *Shift* and click on any axis during a transformation (rotation, scale, or grab), it will remove the constraint on that axis.
2. We then place our cylinder at the center of the forks.
3. Before you enter the **Edit Mode**, consider applying the rotation and the scale (press *Ctrl* + *A* and select **Scale and Rotate**) to avoid unpleasant surprises.
4. In the **Edit Mode**, we add an edge loop at the center (*Ctrl* + *R*) and remove the faces on the left-hand side (press **X** and select **Delete faces**).
5. Then we can add a mirror modifier with the clipping option activated.
6. In the **Edges Mode**, we add several loop cuts using a **Bevel** (press *Ctrl* + *B* and scroll the mouse wheel up) on the outer edge of the wheel (refer to **2** in the following screenshot), then a succession of insets and extrusions to form the side of the wheel. To work more easily on this, we will enter the **Local Mode** by pressing the slash key (/). This is like hiding all the other objects. If you want to leave the **Local Mode**, press the slash key again.

7. A **Subdivision Surface** can be added.
8. The N-Gon will also be transformed in quads on the side with the Vertex Connect Path (*J*) (refer to **4** in the preceding screenshot).
9. We will round the wheel by adding an edge loop to the center of it from the front view.
10. After this, we will add some grooves. For this, we will add five edge loops vertically on the front of the wheel (refer to **6** in the preceding screenshot). We will set the smoothness option to **1** in the last active tool panel in order to place the grooves without destroying the curve profile of the wheel.
11. Then we add a **Bevel** (*Ctrl* + *B*) and push its resulting faces by doing an **Extrude** (*E*) with a scale based on the normals (*Alt* + *S*).
12. Then we accentuate every stripe with the LoopCut tool (*Ctrl* + *R*).

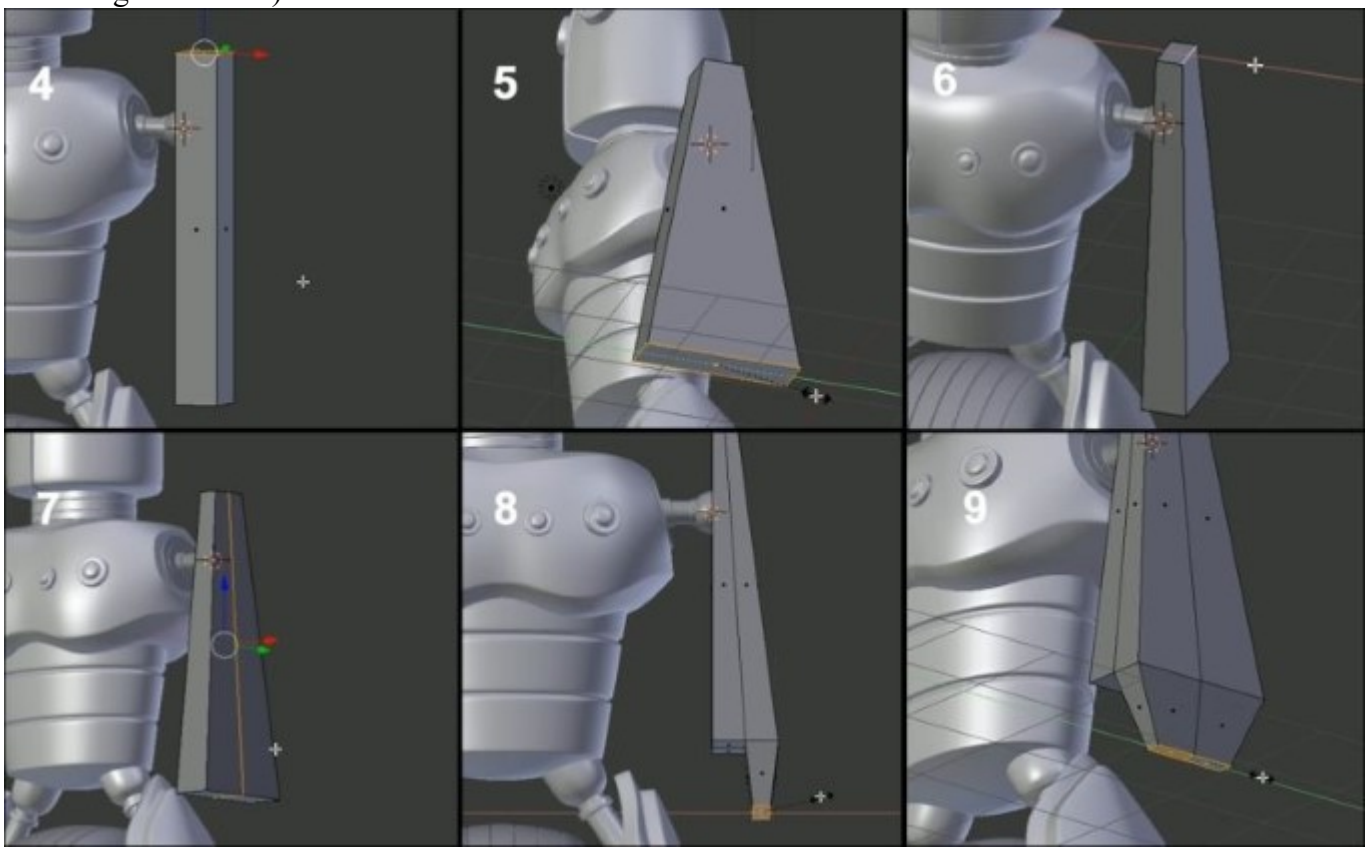
Modeling the arm

The arm is composed of four objects: the shoulder, the articulation ball, the arm, and the wheel. We will review the previous techniques by always focusing on the topology. Let's begin with the shoulder.



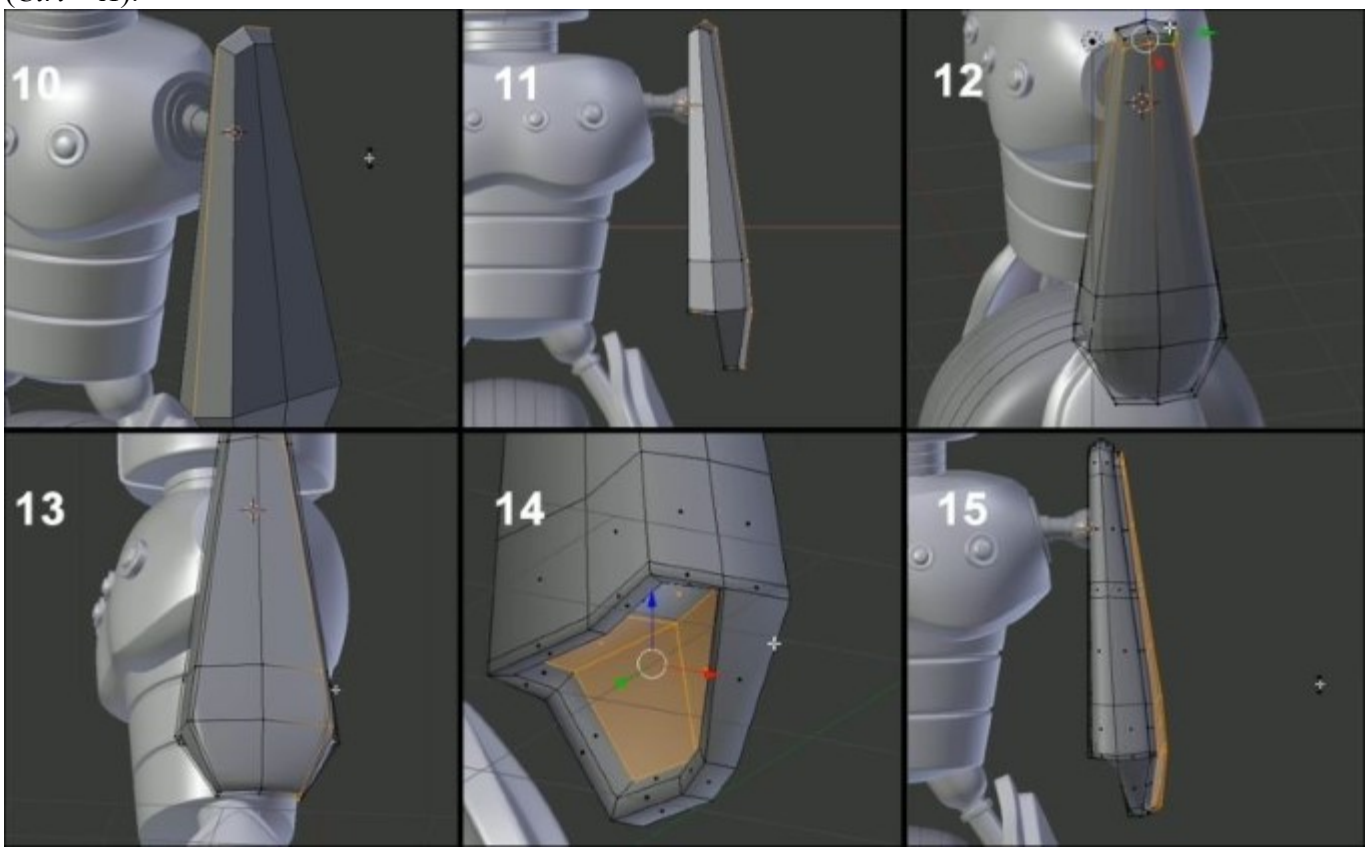
1. We will start with the arm articulation. For this, we simply add a cylinder that will be placed at the right shoulder location (refer to 1 in the preceding screenshot). It must be correctly oriented with a rotation (*R*) and flattened in the **Object Mode** with the Scale tool (*S*). When doing this, remember to constrain on the correct axis of transformation. To do these manipulations, it is better to be in orthographic view. Whenever we transform an object in the **Object Mode**, we don't forget to apply these transformations (*Ctrl* + *A*).
2. Then, we do a series of extrusions (*E*) in the **Local Mode** (*/*) and remove the face that enters the chest and thus will not be visible. Avoiding polygons that are not displayed is a good practice. Not doing this might lead to wastage of your computer resources and this is especially true for complex scenes with many polygons.
3. We can add details by digging a face loop (extrude and scale on the normals with *Alt* + *S*), which can be created with two edge loops (refer to **2** in the preceding screenshot).
4. We add a **Subdivision Subsurface** modifier and we apply **Smooth Shading**.
5. Then we extrude the tip that will hold the articulation ball. This extrusion will be flattened on the *x* axis (*S* + *X* + *0* numpad key) (refer to **3** in the preceding screenshot).

6. We add a sphere by placing the cursor in the middle of the last edge loop right at the tip of the articulation (press *Shift* + *S* and select **Cursor to Selected**).
7. The cursor is in the right place within the **Object Mode**, so we add a UV sphere (*Shift* + *A*). We lower the number of segments to **16** and the number of rings to **8** in the last active tool panel.
8. We add a **Subdivision Surface** and apply a **Smooth Shading**.
9. We can delete the two vertices that are located on either side of the sphere as they will be hidden by other objects (press *X* and select **Delete vertices**).
10. We place the cursor at the center of the sphere and add a cube.
11. We need to resize the cube and then, in the **Edit Mode**, we move the top and bottom faces on the *z* axis (*G* + *Z*) to set the height of the arm (refer to **4** in the following screenshot).
12. With the bottom face selected, we make a scale on the Y axis (refer to **5** in the following screenshot).
13. We select the external edge of the top face and slightly move it on the *x* axis (refer to **6** in the following screenshot).

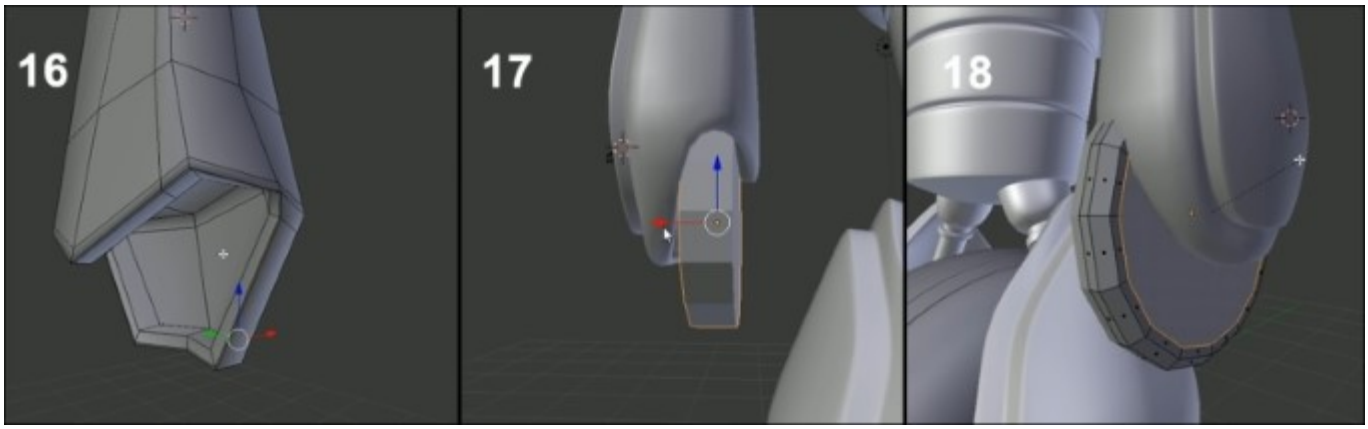


14. We add an edge loop to the center (refer to **7** in the preceding screenshot) in the side view and round the shape by slightly moving it upwards.
15. Then, we add a vertical edge loop in the front view in order to add the needed geometry for the protection of the wheel with an extrusion (refer to **8** in the preceding screenshot).
16. We now select the two right faces from the bottom view (*Ctrl* + *7*).
17. We scale the faces slightly.
18. We will round the profile of the arm by selecting the middle edge loop and by pushing it along the normals (*Alt* + *S*).

19. It's time to add our lovely **Subdivision Subsurface** modifier and remove the flat shading. As always, we will maintain the sharp angles with a couple of edge loops.
20. We will then select the four inner faces of the hand and do an inset. This will create a face loop delimiting the inner part of the hand. These inner faces will then be extruded inward in order to create the hole that will keep the wheel.
21. We will then do an inset of the external faces of the arm.
22. These faces will then be extruded to create a small thickness.
23. It's now time to add the cylinder primitive of the hand.
24. We place it at the right location and change its size. As always, we will apply the transformation (*Ctrl + A*).



25. It will be easier to model the wheel with a mirror modifier. So we will cut the cylinder into two equal parts (*Ctrl + R*), and we will remove the left side of it to add the Mirror modifier with the clipping option checked on.
26. With multiples insets and extrusions, we then construct the wheel while in the Local view (/) (refer to **18** in the following screenshot).



27. We will again use the subsurface modifier with the **Smooth Shading** option.
28. After we have shaped the silhouette of the wheel, we will add asymmetric details on its left-hand side by applying the mirror modifier. Add a hole here with your best friends: the Inset and Extrude tools.
29. We will then fix the hand to encapsulate the wheel in it with precision. For this, we will use **Proportional Editing**.
30. Maybe you've seen that there are bad tensions in the right-angle form of the hand and the arm due to the **Subdivision Surface** modifier. These kind of artifacts warns you of a bad topology. So we will, of course, find a way to correct this. To do this, we use the Knife tool (*K*) and we cut by following the hand outline (refer to **16** in the preceding screenshot).
31. We can use the Merge tool (press *Alt* + *M* and select **At center**) in order to merge the two vertices that are part of the newly created triangle.
32. We will also close the newly created N-Gon with the Vertex Connect Path tool (*J*).
33. The **Bevel** tool will be useful to maintain the inner border of the hand.
34. We will now mirror the arm, its clip, and its wheel in the **Object Mode** with the mirror modifier using the chest as the mirror object.
35. You can always push your modeling further by adjusting the transformation of each part and by adding details with the tools that we have introduced you to (such as the LoopCut tool, Extrude, and Scale along the normals).

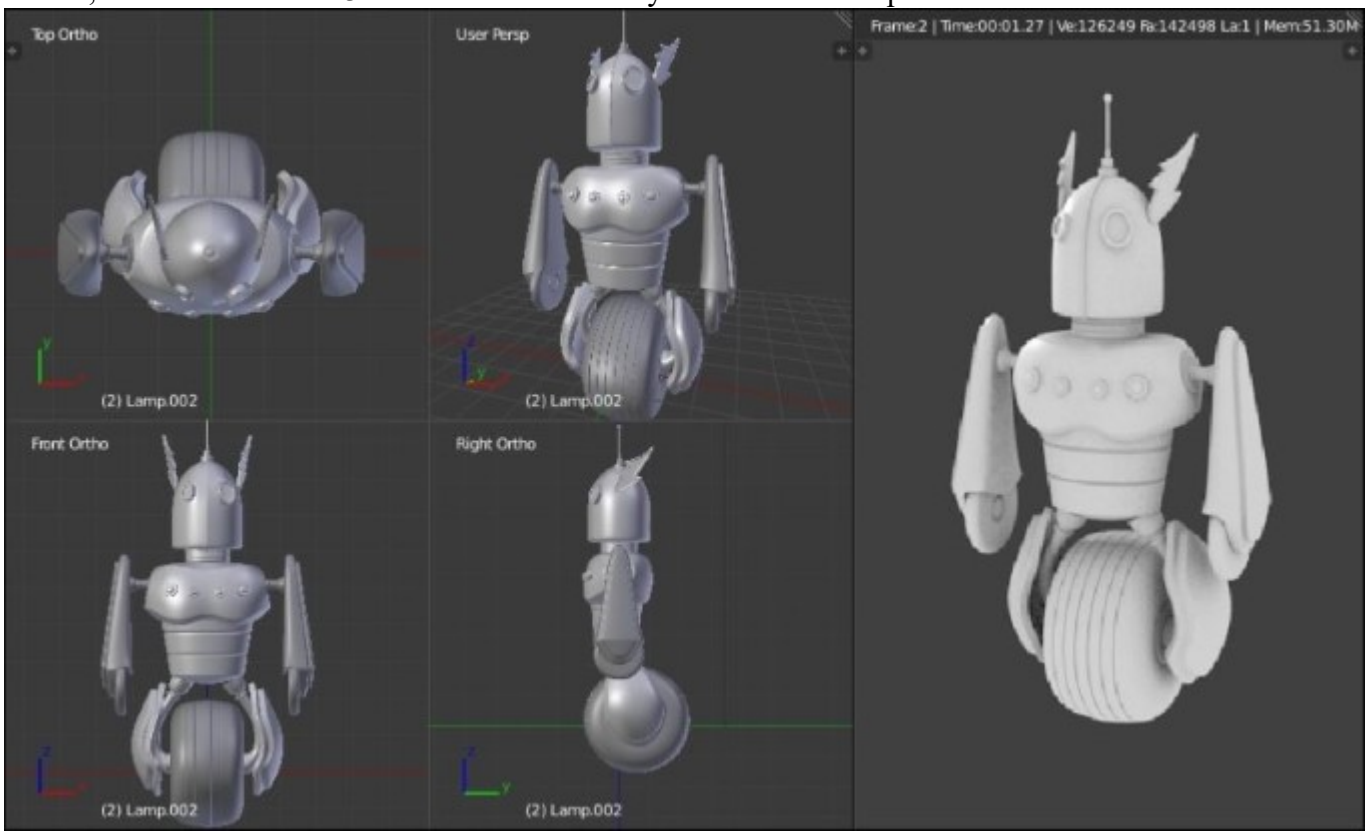
All the parts of the robot are done now! Congratulation!

Using Blender Internal to render our Robot Toy

We will now select the camera of our scene, and in a new 3D editor, we will see through it. To do so, we'll perform the following set of steps:

1. We will split the 3D view, and in the newly created editor (which should be 3D view), we will press the 0 numpad key.
2. We can move or rotate the camera like a normal object. Notice that the camera is shaped like a triangle. This represents the field of the view of your camera. If you want to place your camera while navigating around the robot, you can press the *Ctrl* + *0* shortcut.

3. Now that our camera is correctly placed (that is, focusing on our robot), we can try to render the scene. We will do a very basic render by activating **Ambient Occlusion** in the **Additive Mode**. This option is located under the World icon button in the Properties editor. You just need to check the corresponding check box.
4. In order to do a render, we will press the *F12* key or go to the Camera icon button in the Properties editor and press the big **Render** button. Blender will automatically switch your current 3D View to a **UV/Image Editor** that will show you the calculated image. You can, of course, switch it back to a 3D view as we showed you in the first chapter.



You've now completed the Robot Toy project!

Summary

In this chapter, you learned to use the main modeling tools in Blender. You will keep learning about the other ways of modeling in the next chapters. Polygons can be seen as virtual clay. There are some rules to follow, but as soon as you know them, you will be free to model everything you wish

Chapter 3. Alien Character – Base Mesh Creation and Sculpting

In this chapter, you will discover a new way of modeling 3D objects with the powerful sculpting tools of Blender.

We will start with an overview of the sculpting process including brush settings and how to optimize the viewport. We will then create a base mesh with an amazing tool that Blender offers called the Skin modifier, which follows the concept art of an alien character.

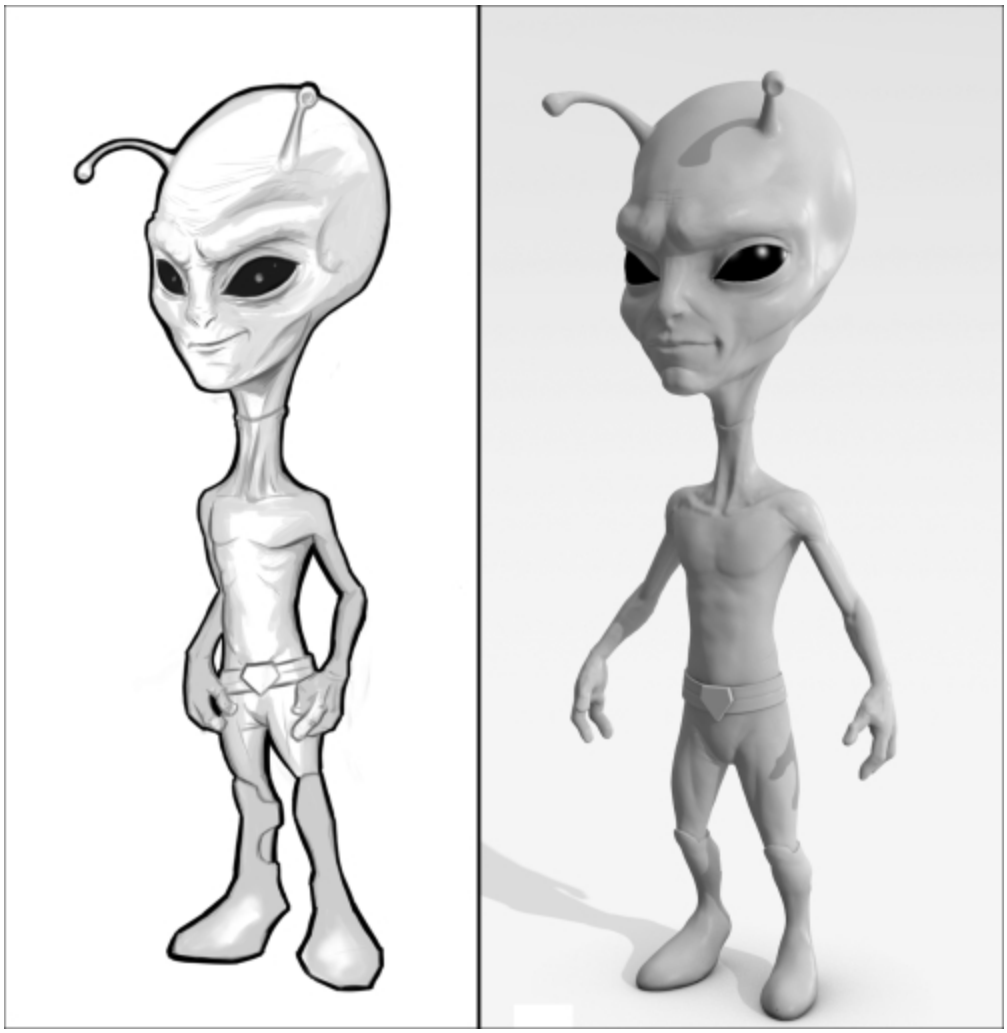
Afterwards, we will sculpt the character using the tools that we had previously introduced and learn more about their usage in the different cases that are required for our character.

As sculpting is an artistic process, you will also learn about proportions and anatomy.

Let's jump to another planet! This chapter will cover the following topics:

- Understanding the sculpting process
- Optimizing the viewport
- Learning about and using brushes
- Creating a base mesh with the Skin modifier
- Using Dyntopo
- Understanding the basics of anatomy and proportions

You will start the sculpting of the following alien character (shown on the right) using a sketch as a reference (shown on the left). This is done with **Krita** (an open source tool for 2D art).



Understanding the sculpting process

Before starting to sculpt our alien, we will take some time to understand what this means and what the advantages are of using this modeling method. We will then give an overview of the basic tools that Blender has to offer.

An introduction to sculpting

Before the introduction of sculpting in the 3D world, there was only the polygonal modeling method (the method that we've used in the second chapter) that takes more time when creating organic shapes. The goal of sculpting is to have more freedom while modeling. The process looks a little bit like real sculpting art, but in this case we sculpt over a 3D mesh (our digital clay). When sculpting in Blender, we use brushes as tools that act on the mesh. There are many brushes that have different behaviors such as digging, moving, or pinching.

Choosing sculpting over poly modeling

Sculpting allows us to think more about the shape of the object and less about its technical part, such as its topology. So, the goal of this method is to really concentrate on the design part of the object. We won't see the vertices, edges, or polygons. The technique is more efficient when the goal is to reach an organic object. When you model with the tools that we have previously shown to you (the poly modeling method) you need to keep the topology in mind while researching the shape, and it is even more complicated when you have finer details. So what if we want to have a good topology with a sculpture? We have to do a retopology, but you'll see this in the next chapter

Using a pen tablet

When we modeled the Robot Toy in the previous chapter, we used a mouse. While we are sculpting, it's pretty hard to use a mouse because it is not precise according to the process. This is why we use a pen tablet that gives the sensibility needed to get the right shape. It takes some time to get used to this, but with practice you will have more control over your sculpture.

In order to navigate with the pen tablet in the 3D viewport, go to the **User Preferences** panel (*Ctrl* + *Alt* + *U*) and check the **Emulate 3 Button mouse** option. We will now be able to use the *Alt* key to navigate. Refer to Chapter 1, *Straight into Blender!* for more precise details.

It's also a good thing to check the **Emulate Numpad** option in order to be able to switch views with the keys that are above the QWERTY keys.

A pen tablet with its stylus

The sculpt mode

In order to access all the tools needed for sculpting we need to go into the **Sculpt Mode**. The **Sculpt Mode** won't let us access the components of our mesh as in the **Edit Mode**, and we also won't be able to apply transformations on our objects as in the **Object Mode**. To switch to the **Sculpt Mode**, we select it in the drop-down menu located in the header of the viewport. As you can see, it is in the same place as the **Edit Mode** and the **Object Mode**.

Optimizing the viewport

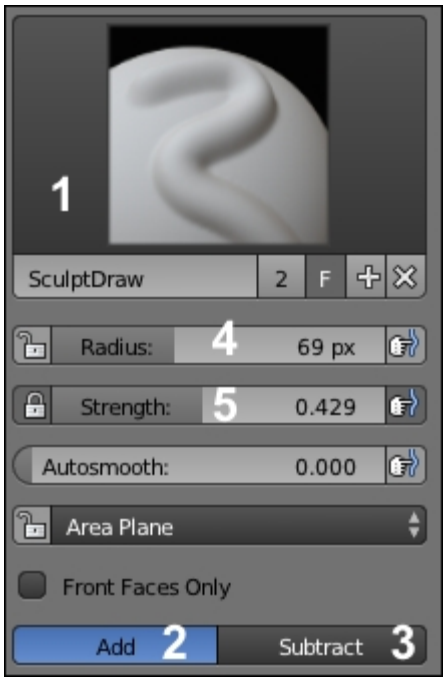
Sculpting usually takes more resources than poly modeling because the number of polygons will quickly increase each time you want to add details. This is why we need to activate some settings that will boost our viewport. This is done as follows:

1. The first setting that we will check is located under the **System** tab in the **User Preferences** window (*Ctrl* + *Alt* + *U*) and it is called **VBOs**. It is used by **OpenGL** (the rendering API used by Blender) to better organize the data displayed on the screen.
2. In the **Options** tab, under the **Options** subpanel in the left panel of the viewport (in the **Sculpt Mode**), we will activate the **Fast Navigate** option.
3. We will also ensure that the **Double Sided** option is turned off. To do this, we can use a nice little add-on called **Sculpt Tool**. After the add-on is installed, on the **Sculpt** tab of the left panel of the viewport we now have the **Double Sided Off** option. Note that you can always access any option by pressing the Space key in the viewport and by typing the name of the tool that you want.
4. Later, when we sculpt our objects, we don't want to have something else other than our objects in the viewport. So we will deactivate the grid, the gizmos, and any other viewport information that we don't need.
5. In order to do this, we will go to the right panel of the viewport. We can open this by pressing the *N* key, and under the **Display** subpanel we will check the **Only Render** option.
6. By checking this option, we will simply deactivate all the options that are below the **Only Render** option, such as **Outline Selected** that consumes a lot of resources of the viewport. Remember this option as we will toggle it on or off depending on our needs.

Anatomy of a brush

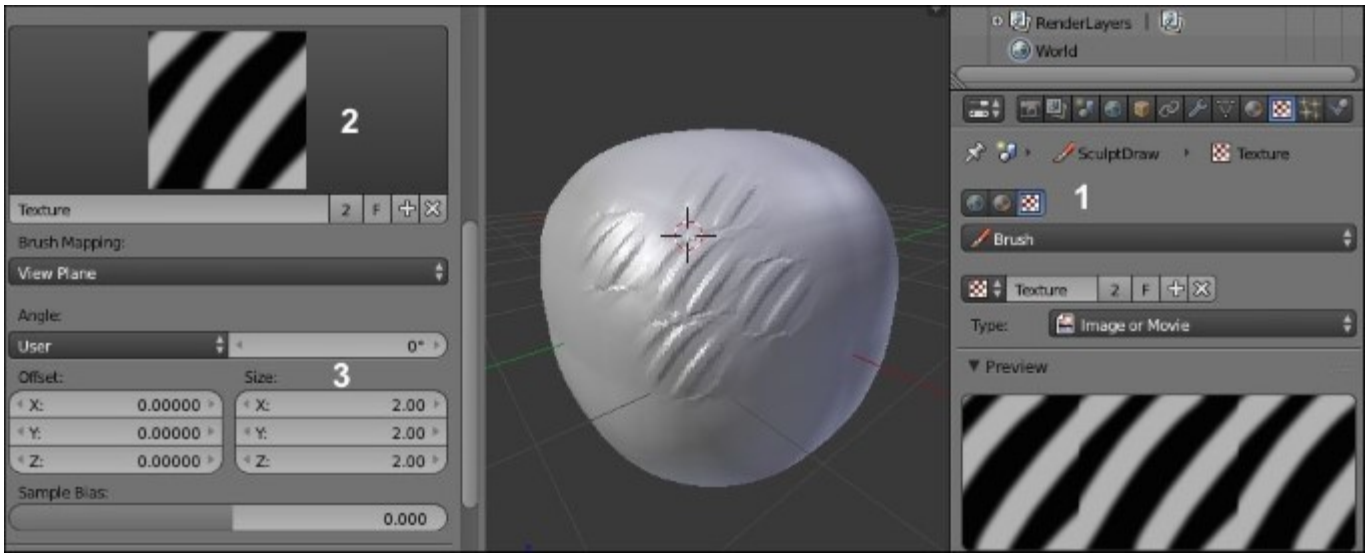
As previously mentioned, we will use a lot of brushes that behave differently in order to sculpt our alien. In this section, we will take our hands over the settings that are shared between all the brushes with the **Sculpt/Draw** brush as an example. Let's perform the following set of steps:

1. In order to do our experimentation, we will use a Cube primitive. In the **Object Mode**, we place our cursor at the center of the scene (*Shift* + *C*) and we add a Cube (*Shift* + *A*). Note that you can use the one placed by default in any new scene if you want.
2. The Cube has a low polygonal resolution so we will have to subdivide it by going in to the **Edit Mode**, select all its components, and use the **Subdivide Smooth** option under the **Specials** menu (the *W* key). We will repeat this action six times in order to have a good density of polygons.
3. In the **Sculpt Mode**, we will then select the **Standard** brush (if not already selected) in the left panel of the 3D viewport by clicking on the brush icon button under the **Tools** tab (refer to **1** in the following screenshot).
4. We can now draw on the subdivided cube. As you can see, it pushes the geometry. This is because our brush has the **Add** option activated by default. If we want to go deeper, we will need to switch to the **Subtract** mode. The **Subtract** option simply reverses the behavior of the brush. Both the options are placed under the **Brush** subpanel in the **Tools** tab (refer to **2** and **3** in the following screenshot). It's not very convenient to click on buttons in order to do such a simple manipulation, so we encourage you to use the *Ctrl* key while sculpting on your mesh to switch between these modes.
5. As you may have seen, we are not really precise because of the size of our brush.
6. In order to change the size, we will use the corresponding slider under the brush icon called **Radius** (refer to **4** in the following screenshot). We can (and recommend this to you) use the *F* key as a shortcut.
7. Now that we have more control over the size of our brush, we will change its **Strength** under the **Radius** slider (refer to **5** in the following screenshot). We can use the *Shift* + *F* key as a shortcut. As you can see, on the right-hand side of both sliders (Strength and Radius), there is a little icon that allow us to use the pen tablet sensitivity in order to dynamically change these options. We will only use this for the Strength option, so when we lightly press on the pen tablet, we will have less strength than if we were pressing it harder.



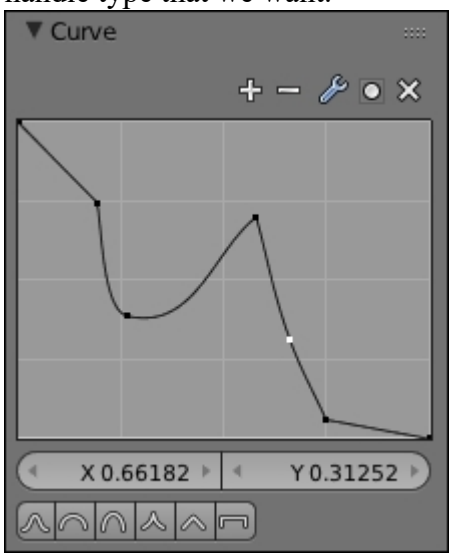
Brush options

8. Another interesting thing that we can set for our brushes is **Texture** (also called an **alpha**). An alpha is usually a black and white image that is useful while adding details such as skin pores or patterns to an object. When an alpha is added to a brush, the black pixels will remove the brush behavior during sculpting. To import an alpha, we will first need to go in the **Textures** subpanel of the Properties panel (on the far left of the interface, by default) and click on the third icon (a checker pattern) (refer to **1** in the following screenshot).
9. We can now add a new texture, and under the **Image** subpanel, we can open an image. We can now go under the **Textures** subpanel of the **Tools** tab in order to select our newly imported texture (refer to **2** in the following screenshot).
10. If we want, we can also change the repetition of our alpha by changing the **X**, **Y**, and **Z** size sliders (refer to **3** in the following screenshot).



Adding a Texture (alpha) to our brush

11. The last setting that we will test is the **Curve** profile of our brush. The curve of the brush is located under the **Curve** subpanel in the **Tools** tab. Changing the curve profile allows us to change the behavior of the brush. For instance, with our current brush, **Sculpt/Draw**, if we click on the last icon (the flat curve) under the curve, we can see that the brush is harder while sculpting over our cube.
12. To understand this setting better, imagine that this is half of the profile of a real brush (see the following screenshot). We can select each point that composes the curve and move it to change the curve profile. We can also add a new point on the curve by clicking anywhere on it. When a point is selected, we can remove it by clicking on the **X** button. This curve is called a Bezier curve, so we can also change the smoothness of a point using the Tool icon and choosing the handle type that we want.



A modified curve of a brush.

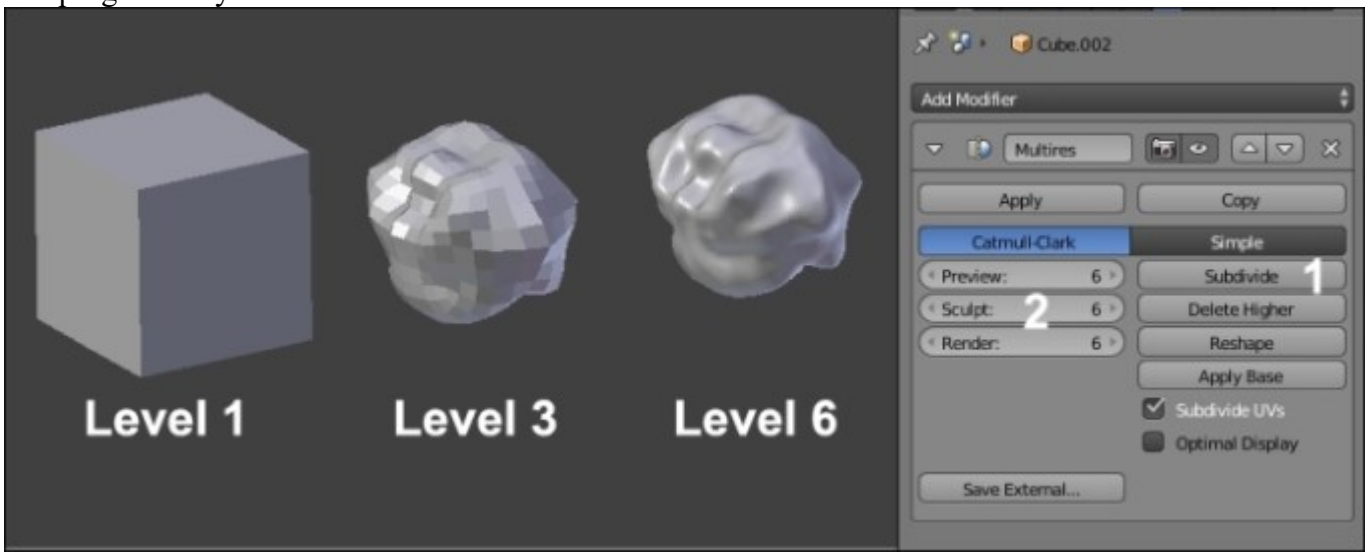
Dyntopo versus the Multires modifier

In order to test our brush settings, we subdivided our cube by hand but it's not practical while sculpting an object because we didn't have enough control over the subdivision. In order to have control over our mesh, Blender gives us two main methods, the **Multires** (a.k.a. Multiresolution) modifier and **Dyntopo**.

First touch with the Multires modifier

The Multires modifier is added to the modifier stack of an object and allows us to maintain subdivision levels of sculpture. For instance, we can sculpt at a low level (with a low resolution), and the details will be transferred to the higher levels and vice versa. We will test it right now! This is done as follows:

1. We will first create a new Blender file (by navigating to **File** | **New**) and then with the default cube selected, we will go to the Properties panel in order to add a Multires modifier.
2. We will subdivide our cube six times with the **Subdivide** button (refer to **1** in the following screenshot). If we were using a **Subdivision surface** modifier, our cube will be rounder. To test this out, we can go into the sculpt mode and start sculpting our cube with the Draw brush.
3. We can now move between the different subdivision levels with the **Sculpt** slider of the **Multires** modifier (refer to **2** in the following screenshot). As you can see, we don't lose our sculpted information while changing levels, we are just changing the amount of details of the object. Of course, when you are at a lower level, you won't have as much detail as at the higher levels. The goal of all of this is to give you the possibility of changing the main shape of your sculpture at a low resolution without overwhelming yourself with all the details that you have sculpted at the higher levels. So don't try to add details too early in order to get the shape right and progressively increase the subdivisions.

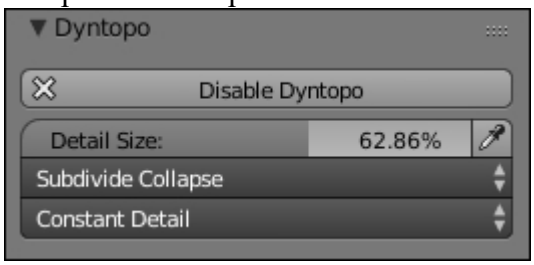


The Multires modifier with an example of three different levels of subdivisions

First touch with Dyntopo

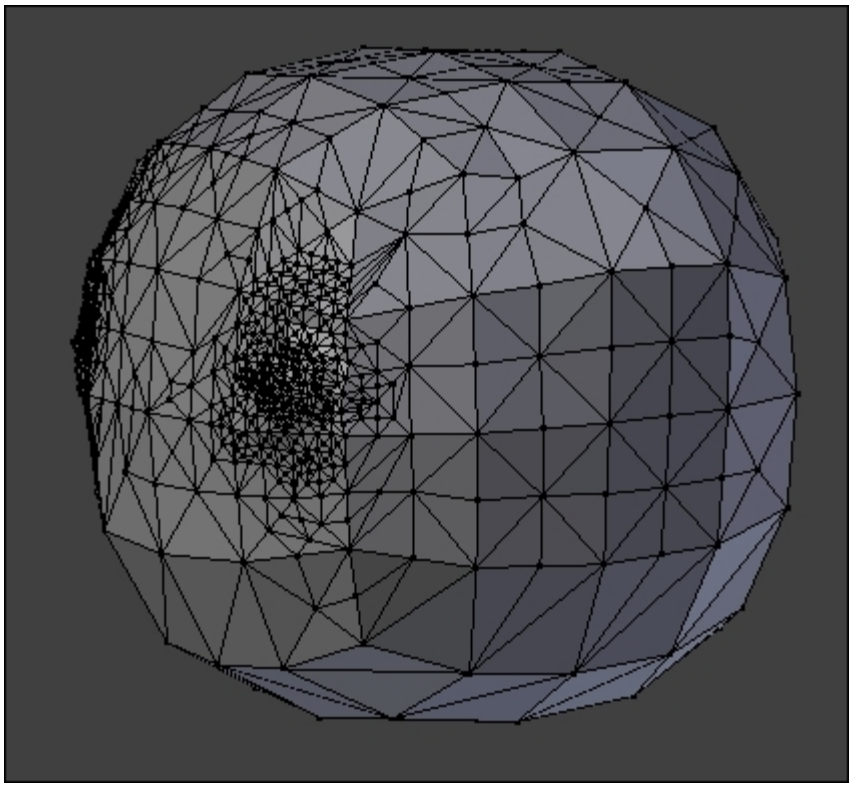
The Dyntopo method will generate details according to the amount that we choose. The geometry will be subdivided when we sculpt an object and will be located where we have placed our mouse pointer. We will be using this method for our alien soon, so let's test this to get used to it:

1. We will first create a new Blender file (by navigating to **File** | **New**).
2. The cube is a little bit low in resolution, so we will subdivide it twice with the **Subdivide Smooth** option (the *W* key).
3. In the **Tools** tab, under the **Dyntopo** subpanel, we can activate Dyntopo by clicking on the **Enable Dyntopo** button. Our cube will now be converted to triangles (this is not a problem because remember, while sculpting an object, we don't care about its topology, we care about its shape). If you want to look at the wireframe of the object, use the *Z* key or simply go into the **Edit Mode**.
4. By default, we are in **Relative Details** as you can see in the second drop-down menu. This option means that the amount of detail will be proportional to the distance of your working camera view. If we sculpt near the object, the amount of detail will be much more important than if we sculpt far from the object.
5. This method is nice, but there is another method that allows us to control the amount of detail without caring about our distance from the object. This is the constant details option (we will use this one for the alien). We can change from **Relative details** to **Constant Details** in the Sculpt details drop-down menu.



*The **Dyntopo** settings*

6. As you can see, we now have the detail size slider expressed by a percentage that allows us to change the amount of detail that our brush will generate on our mesh. With a small percentage, we will have finer details and vice versa.



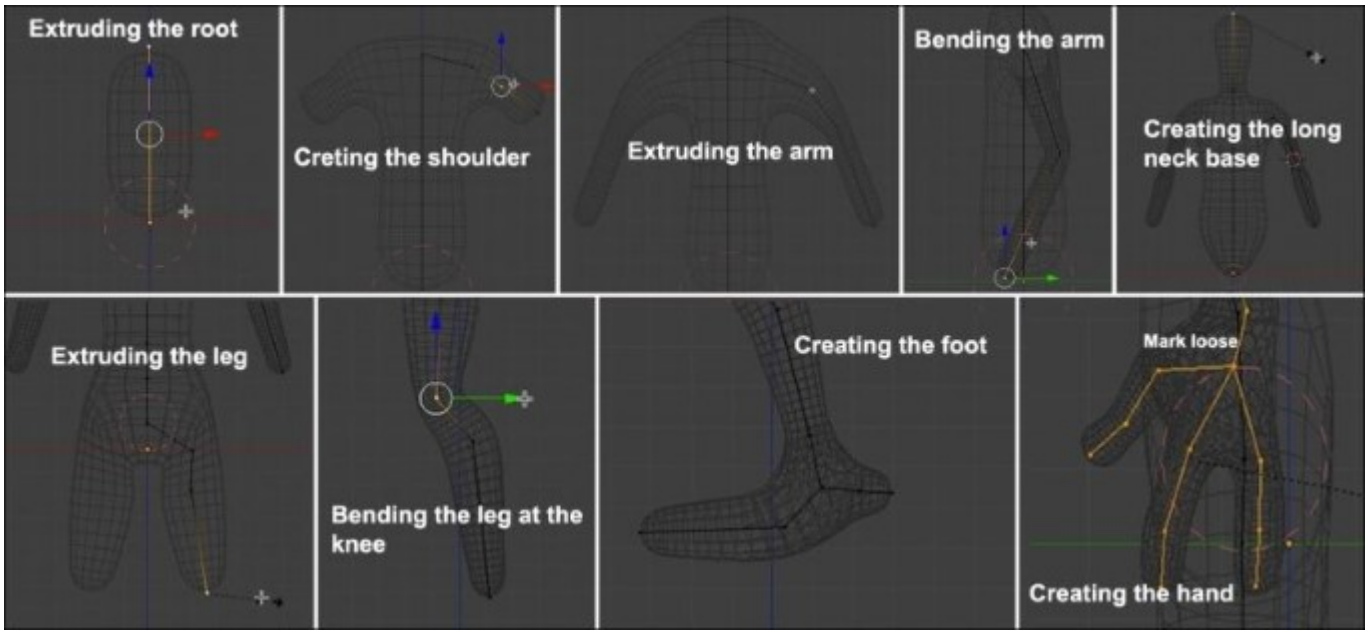
A Dyntopo mesh with different levels of sculpted details.

Creating a base mesh with the Skin modifier

Before we sculpt our alien, we need to have a base mesh that has roughly its proportions. If you want, you can use the methods that you've learned in the previous chapter in order to model it, but here we are going to use a cool modifier that Blender has to offer: the **Skin** modifier. Its goal is to create a geometry around each vertex. We can simply extrude some vertices as if we were doing a real wire armature, and the Skin modifier will add volume around it. For each vertex, we have control over the volume size. Let's start our base mesh:

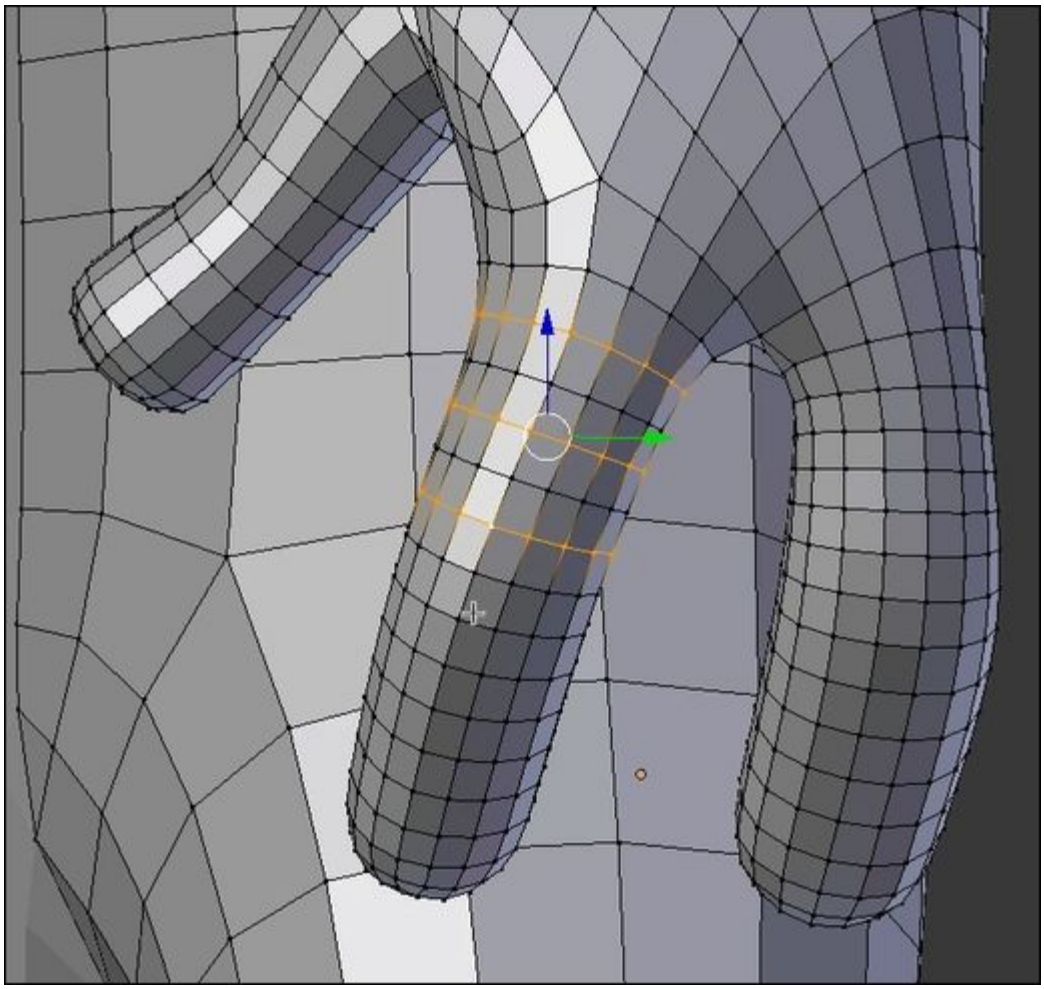
1. We will start by entering in to the **Edit Mode** of our default cube. Then we will select all the vertices (*A*) and merge them to a sole vertex at the center (press *Alt + M* and click on the center). We now have our root vertex that will be the pelvis of our alien.
2. It's now time to add a **Skin** modifier to our object in the modifier stack. As you can see, our vertex is controlling a new geometry around it. The geometry is low, so we will add a **Subdivision surface** modifier on top of the Skin modifier in order to have a smoother look. As you may have seen, the vertex has a red circle around it. This means that it is the root of our armature.
3. We can now extrude our vertex (**E**) on the Z axis in order to start the torso of our alien.
4. Now we will add a **Mirror** modifier with the **Clipping option** turned on. This modifier needs to be placed before the Skin and **Subdivision Surface** modifier (use the up and down arrows to move it to the first place). Ensure that the vertices that are on the symmetry axis are merged.
5. We can now select the top vertex (the base of the neck) and extrude it by pressing *Ctrl* and LMB to the right in order to create the shoulder. Be careful with the position of your vertex as the topology generated could be bad. Always try to move your vertices a little bit in order to see whether you can't have a better topology.
6. We will now change the size of the volume that has been generated around the shoulder vertex. To do this, we will use the *Ctrl + A* shortcut and we move our mouse to adjust it.
7. We will then extrude the arm. In order to give a more dynamic shape, we will bend it at the elbow. We then adjust its size. At this point, it is very important to match the proportions of the concept. Proportions means the length and size of the different members with respect to each other. If you need to constrain the size or change the volume on a certain axis, you can press *Ctrl + A + X*, *Y*, or *Z*.
8. Let's extrude the long neck of our alien.
9. It's now time to add the legs of our alien. We will do this by extruding the pelvis vertex at a 45 degree angle (press *Ctrl* and LMB). Then we extrude the leg and adjust its profile by changing the size of the different vertices (*Ctrl + A*). As we did for the arm, we will bend the leg a little bit at the knee location. Note that the pelvis vertex should be always marked as the root of the armature. If this is not the case, select it and in the **Skin modifier**, press the **Mark Root** button.
10. The foot will be then extruded and resized. We can create the heel by simply extruding the ankle vertex to the back. The ankle vertex needs to be marked as **Loose** in order to have a nice transition with the front and the heel of the foot. To do this, we select it and use the **Mark Loose** button in the Skin modifier.
11. It's now time to create the hand by extruding the wrist vertex. From the new vertex, we will extrude three fingers that will be rescaled appropriately. Note that the thumb is at a 45 degree angle from the other fingers. To add a more dynamic feeling to the hand, we will slightly bend the fingers inwards.

12. We will then extrude the base of the neck. From the newly created vertex, we can extrude the head vertex that will then be the base for the chin and the cranium.



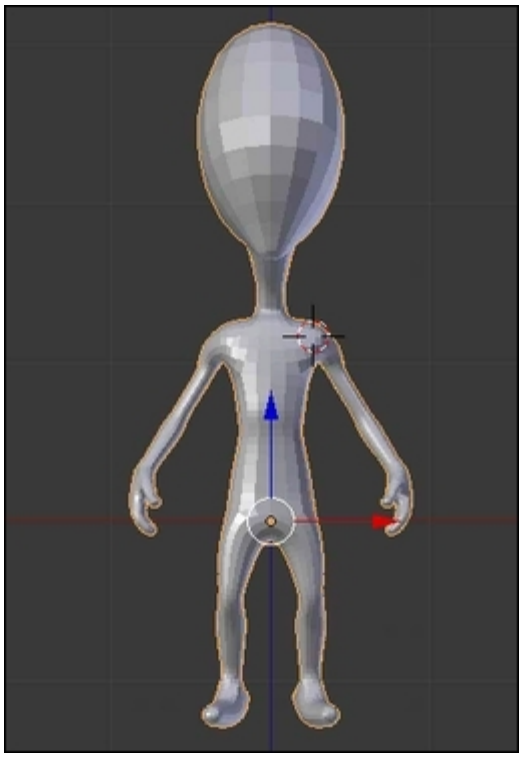
*The steps of the base mesh creation with the **Skin** modifier.*

13. We can now apply all our modifiers from the top to the bottom of the stack.
14. If we enter in the **Edit Mode**, we can see that some parts are very dense. This is why we are going to remove some edge loops from certain parts such as the fingers. However, rather than doing this for both sides of the model, we are going to split the mesh in two and add a mirror modifier. To do this, we first need to ensure that there is a symmetry axis in the middle of our mesh. If this not the case, you can use the knife tool (*K*) to create it. Then we can delete half of our model and add a mirror modifier as we did in the previous chapter. We can now select some edge loops where there is a lot of condensed geometry by pressing *Shift* + *Alt* and the RMB, and by pressing *X* we can delete them (delete the **edge loops**, not the vertices or faces).



Removing some edge loops of the dense parts.

15. The base mesh is now ready to be sculpted.

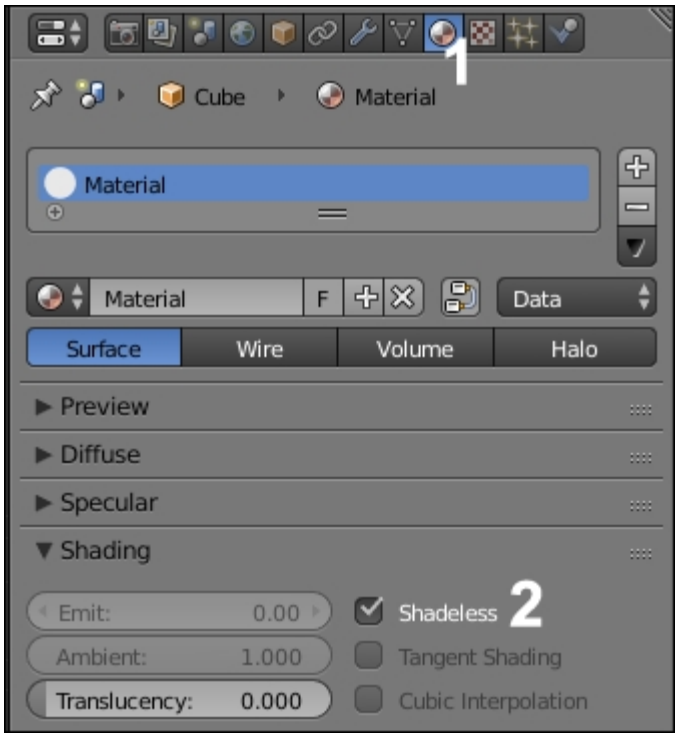


The final base mesh

Visual preparation

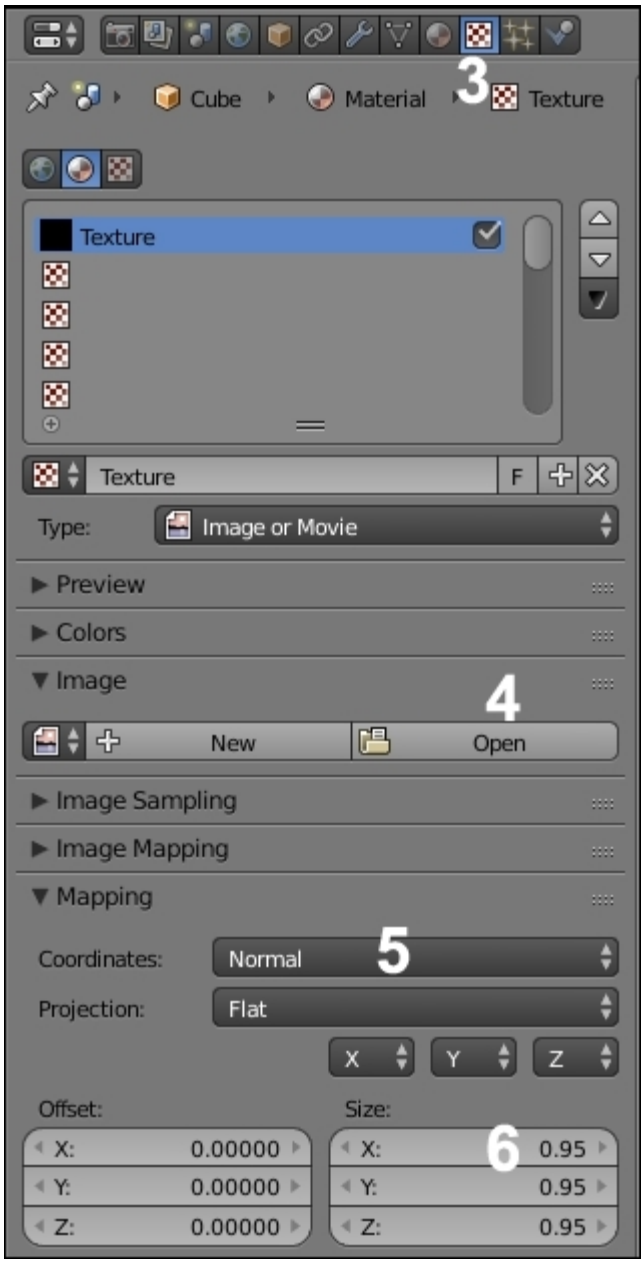
While sculpting, it's nice to use Matcap. It is simply an image that will be projected on your mesh in the viewport and that looks like a material. For instance, you can use a Matcap that reminds you of clay. Let's begin with our sculpting:

1. To set up a Matcap for our mesh, we will have to set up the default material of our mesh in the Properties panel under the **Material** tab (refer to **1** in the following screenshot). Note that if you can't see a material, you can press the **New** button.
2. Now we will check the **Shadeless** option under the **Shading** subpanel (refer to **1** in the following screenshot).



Creation of a new material with the Shadeless option.

3. As we have said before, a Matcap is an image, so we will import our image as the texture of our material. To do this, we go to the **Texture** tab (refer to **3** in the following screenshot) and we add a new texture by clicking on the **New** button. In the image subpanel, we will click on the **Open** button (refer to **4** in the following screenshot) and we choose our Matcap image.
4. A Matcap is mapped to a mesh according to its normals, so in the **Mapping** subpanel change the **Coordinates** from **UV** to **Normals** (refer to **5** in the following screenshot). We will also adjust the size of our projection by decreasing the **X**, **Y**, and **Z** size sliders to 0.95 (refer to **6** in the following screenshot).



Setting the Matcap image texture for our material

5. In order to see our Matcap in the viewport, we will replace the **Multitexture** display mode with **GLSL** in the right panel of the 3D view (*N*) under the **Shading** subpanel.
6. Last but not least, we will go into the **Shading** mode under **Texture Viewport** using the corresponding drop-down menu in the 3D view header. You can also use the *Alt* + *Z* shortcut to quickly switch to this mode. Our Matcap is now perfectly set up!

*Setting the **GLSL** display mode in the right panel of the viewport (N).*

An introduction to artistic anatomy

Before continuing with the alien creation, some basic knowledge of anatomy can be very useful. It is a basic discipline for a character artist. Don't worry, we will clarify the concerned parts of the body for each step with illustrations.

Of course it is an alien, so we can accept the fact that we don't always have to respect human anatomy for specific parts. He has a huge head, only two fingers and a thumb, and very different feet. His humanoid appearance imposes some anatomical likelihood. This is especially useful if you plan to animate it later on. Improving your knowledge of this topic will help you to understand the movements and postures better.

In the early 16th century, Leonardo Da Vinci was one of the first artists who tried to understand the human anatomy. While religious obscurantism prohibited the examination of corpses, he dared to defy this. By dissection and observation, he did many illustrations detailing the positioning of muscles, joints, nerves, and organs.

It won't be necessary in our case to know the scientific names by heart. It's rather important to know and understand their general forms. Overall, it's more about the comprehension of human body mechanisms.

Note

Many good books treat this subject. For more information, you can have a look at these websites:

- http://md3dinc.com/
- https://www.anatomy4sculptors.com/
- http://www.3d.sk/

Sculpting the body

We are continuing the modeling of our alien using Dyntopo as we had previously mentioned. This will allow the creation of the antennae very easily while they are not yet present in the base mesh.

The following is the preparation of our environment before sculpting:

1. We will set the optimizing options that were previously explained.
2. We can adjust the lens parameter in the right panel of the 3D viewport (*N*). A high value lessens the focal deformations.
3. We must check the mirror options in the left panel of the 3D viewport (*T*) to Symmetry/Lock by choosing the axis of symmetry. It is very important in order to save time.
4. We will then activate the **Dyntopo** option in the left panel of the viewport. A detail size of around 25 percent is enough to start. It depends on the size of your model.

For a better understanding, we are going to start sculpting by adding details by iteration.

The head

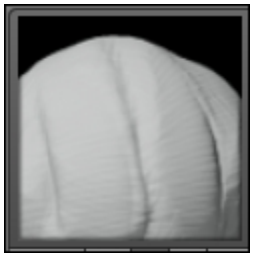
We start defining the jaw and chin with the **Clay Strips** brush (refer to **1** in the following screenshot). It is unnecessary to have too many details for the moment. While sculpting, always remember to define the main shapes (the volumes) and then gradually move towards the details.

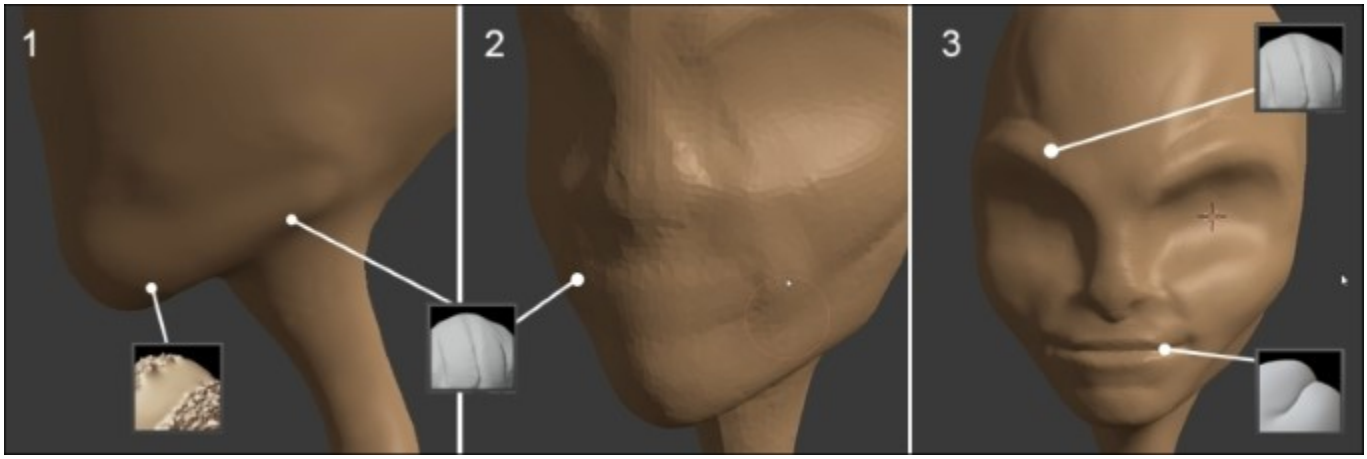
We then accentuate the delimitation between the jaw and neck without exaggeration. A strength of **0.5** is enough.

Note

The Clay Strips brush

The **Clay Strips** brush is a very useful brush to define muscles, dig or add polygons in a straight direction with pretty sharp outlines. It is the equivalent of the Clay Buildup in Zbrush.





1. In order to dig into the polygons, we press *Ctrl* while sculpting. It allows us to switch to the **Subtract** mode on the fly.
2. Then we slightly smooth the added geometry with the **Smooth** brush for a better blend of the created shape. You need to remember to smooth the shape very often in order to avoid having something too grainy.

Note

The Smooth brush

This is often a very useful brush. It allows you to soften and smooth your shapes. This brush is so useful that there is a special shortcut that you absolutely need to know. Hold the Shift key while sculpting in order to use it. It works while using any brush. When you use it over a big density of polygons, it will be harder to smooth your shape, so it will be mainly useful before working on the details.

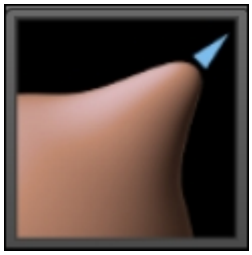
3. Now that the jaw is sketched, we will go to the side view. Then we have to adjust the silhouette of the neck, as well as the skull with the **Grab** brush. Our alien has a very curved neck.
4. We will adjust the shape of our model little by little by turning it around. It's very important to observe your model from different points of view.

   Do not hesitate a moment to look at the reference sketch and position the alien in a very similar pose. You can also create a new 3D view editor to keep an eye on the view of your choice (in our case, we've used the orthographic side view).

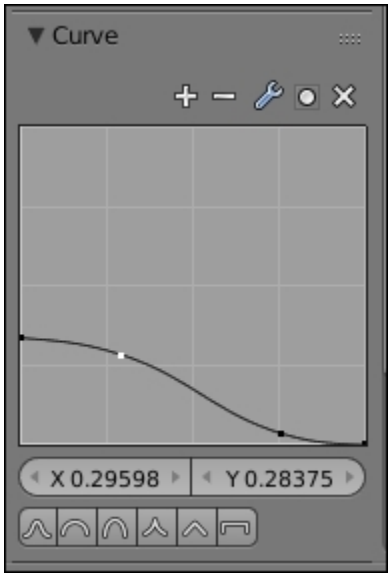
Note

The Grab brush

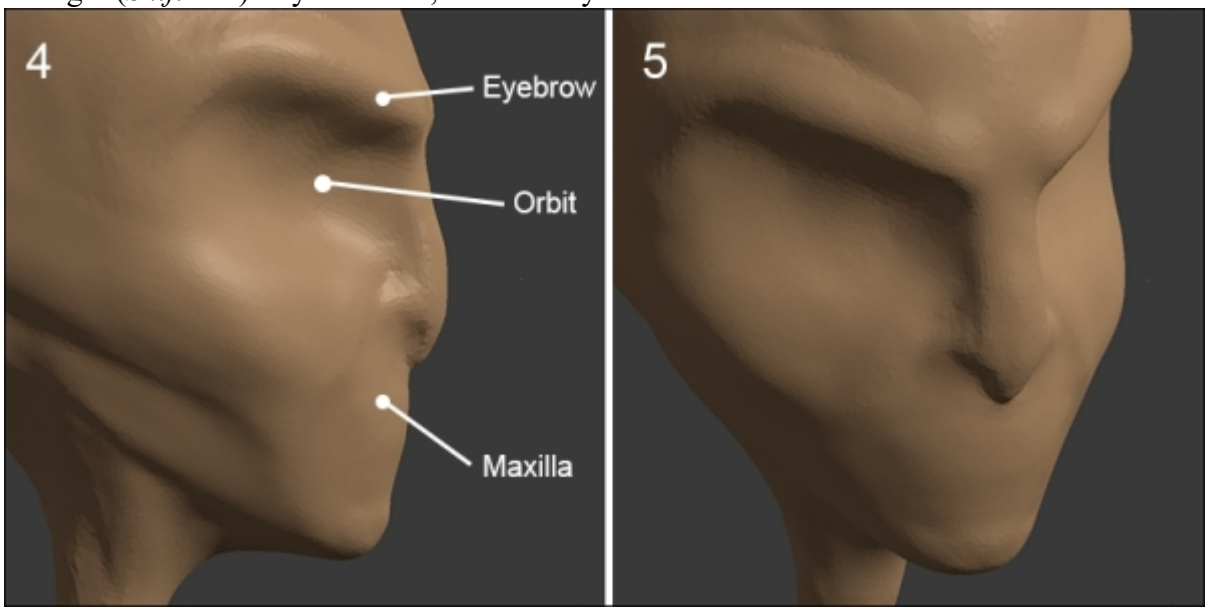
This is often a very useful brush when you start a new model. It works a little bit in the same way that the **Proportional Editing** tool does. If you have any difficulties using it, you can go to the **Curve** options in the left panel of the viewport and modify the curve to decrease its profile.



In our case, we are going to use a very soft curve for the **Smooth** brush.



1. Again, we will take the **Clay Strips** brush to keep going on the face of the alien in order to dig the orbits and accentuate the eyebrows by adding matter (refer to **2** and **3** in the previous screenshot). Be careful to not add too much volume at this place. Do not hesitate to decrease the strength (*Shift* + *F*) of your brush, if necessary.



2. We will also start to sketch the nose and the mouth (refer to **3** in the previous screenshot). Before detailing the lips, we need to start adding some new volume. We form a rounded edge created by the maxilla and the jaw. Remember that the front of a set of teeth has almost a semi-cylindrical shape. It's easier to create the mouth rounded volume from the bottom view.

3. Once this is done, we can start sketching the opening of the mouth with the **Crease** brush, which allows us to draw a mined line. Consider the fact that the geometry is dynamic, so don't hesitate to add some resolution and pinch your shapes in order to make them more accentuated.

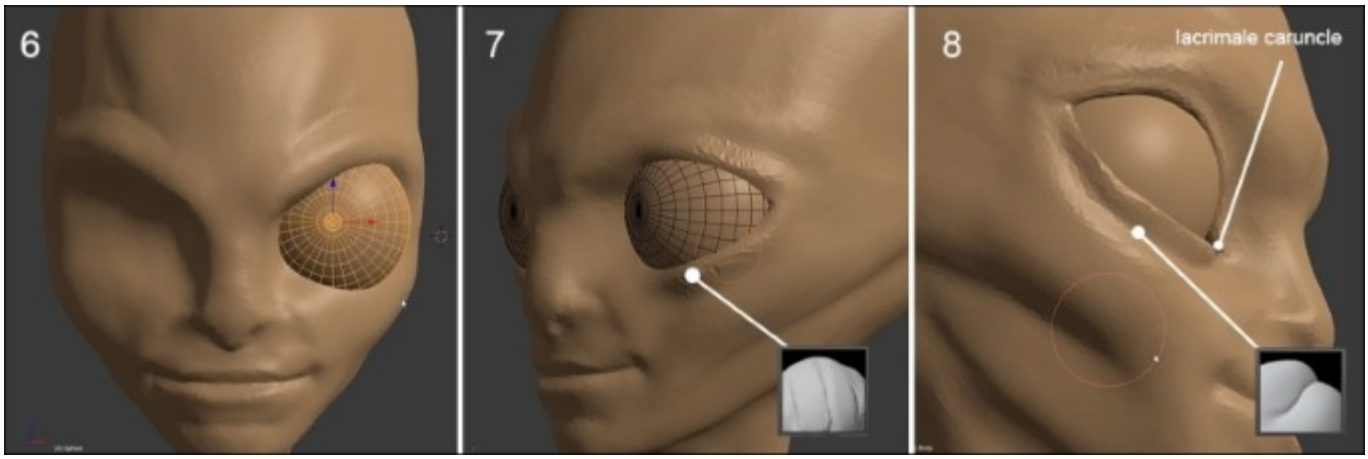
Note

The Crease brush

This brush will allow you to draw lines by digging or adding some volume to your shapes while being pinched. It is perfect to accentuate muscles and make them well visible. It is the equivalent of the Dam Standard in Zbrush.

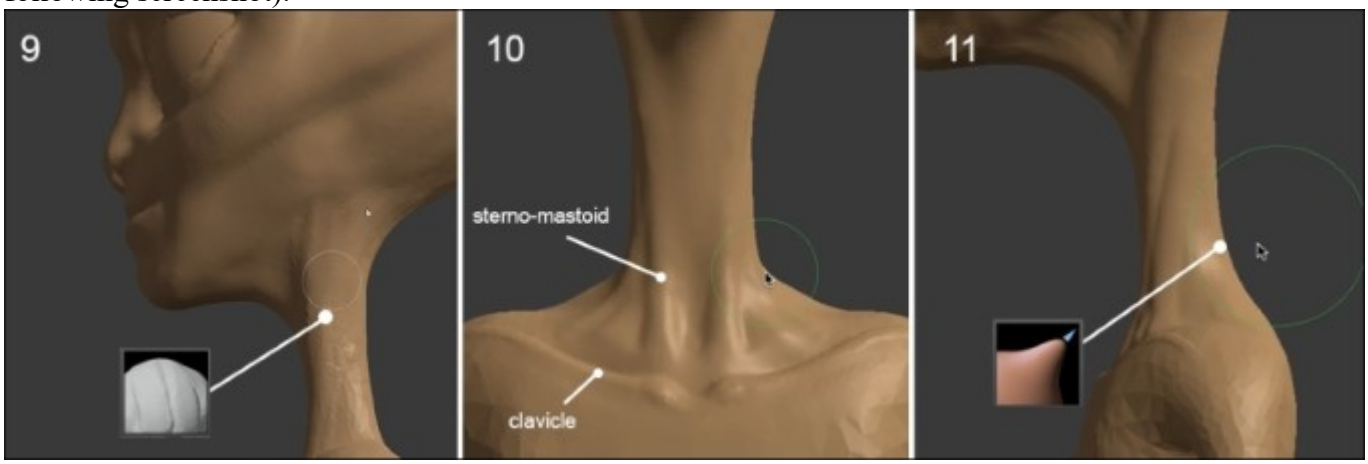
4. We will again use the **Clay Strips** brush in order to give some volume to the lips.

   Remember to leave a little gap between the lower lip and the top lip. The top lip is a little bit more forward than the lower lip from a profile view.
5. In the face view (refer to **5** in the previous screenshot), we will go back to the **Object Mode** (*Tab*) and place the 3D cursor where we want the eye at the middle of the orbit. It doesn't matter if the orbit is not completely dug.
6. We will add **UVSphere** (*Shift* + *A*). We will resize this with the **Scale Tool** (*S*), and we will position it (*G*) in the front view (*1*) just as in the side view (*3*). For a good placement of the eye, looking at the wireframe can be helpful. You only need to go to the Object Data tab and then to the **Group** in the Properties editor, and check the **Wire and Draw all Edges** option.
7. Then, we can sculpt around the eyelids with the **Clay Strips** brush by being careful to accentuate the outside and inside corner (the lacrimale caruncle) (refer to **7** in the following screenshot).
8. In order to accentuate the eyes, let's pinch the upper and lower eyelids with the Crease brush (refer to **8** in the following screenshot).



Finding a good position of the eye is not an easy task. You can move the facial structure with the **Grab** brush if you have problems. The top eyelid must be slightly forward. Conceptually, the eyes are inordinately big, so be careful that they don't touch each other.

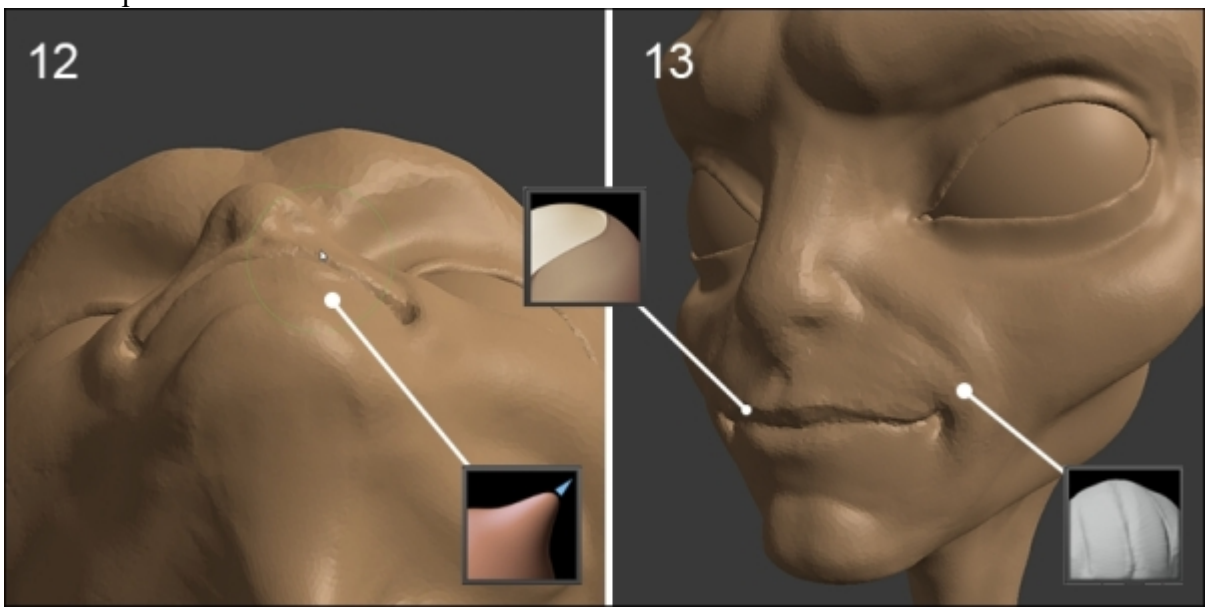
9. Now that the face begins to take shape, we will continue with the neck. We adjust the shape a little bit more with the **Grab** brush, then we start sculpting the muscles and bones of the neck. We carve the clavicles with the **Clay Strips** brush.
10. Then we will move on the sterno-mastoid muscles. It is a muscle group that starts from the mastoid near the ear and attaches to the sternum and the clavicle. We keep working with the **Clay Strips** brush (refer to **9** and **10** in the following screenshot).
11. In the side view, we can polish the silhouette of the neck with the **Grab** brush (refer to **11** in the following screenshot).



Now let's refine the face:

1. We will come back to the mouth by adding some volume to accentuate the circular muscles around the mouth. Be careful to adjust the level details of Dyntopo to around 10 percent (refer

to **13** in the following screenshot). As you may have seen, when you are sculpting an object, you don't directly get the shape that you want, so you always need to go back and forth over the different parts.



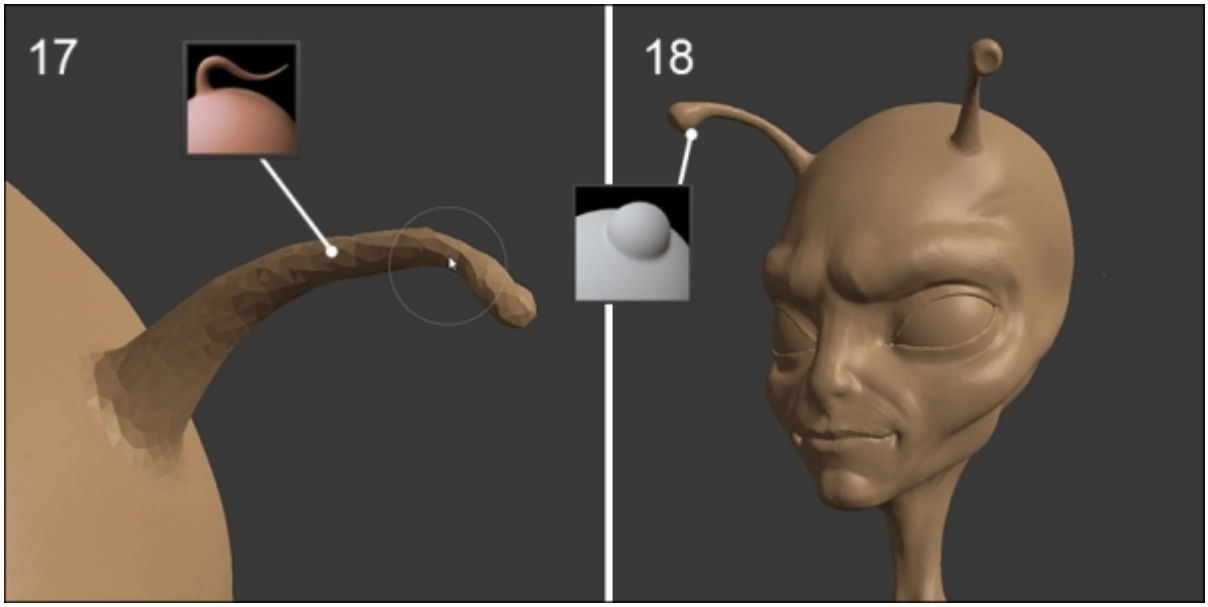
2. This brings us to accentuating the wrinkles that make the junction with the cheeks.
3. Then we will go and pinch the upper lips with the **Pinch/Magnify** brush. The lower lip doesn't need to be as pinched like the upper one. Again, you can increase the level detail of Dyntopo to around 7%.

Note

The Pinch/Magnify brush

This brush allow us to pinch the polygons outwards or inwards (in the **Subtract** mode). It is perfect in our case to detail the lips, the wrinkles, or accentuate the contour of muscles. It is often used to get a cartoon style or for a hard surface modeling where you need to sculpt angular surfaces.

4. We will take a moment to turn the head, including the top view (**16**); then we adjust the round shape of the skull with the **Grab** brush.
5. Now, it's time to add the antennae (refer to **17** in the following image). For this, we are going to use the **Snake Hook** brush with a Dyntopo level detail of around 14%. The difficulty will be to find a good point of view of the head because we can't move the view while extracting the geometry with the **Snake Hook** brush. We must be positioned on the side in order to be able to extract the matter from a little area on the top of the forehead and stretch it outwards in a good direction. Do not hesitate to make several tries if this does not suit you. You can always adjust the size of the brush and the level of detail.



Note

Undo while being in the Sculpt Mode

Unfortunately, the Undo function of Blender is not very optimized for the moment in the **Sculpt Mode**. It can be very slow, so do not use it too often. In many cases, you can probably quickly fix your mistakes without Undo.

The Snake Hook brush

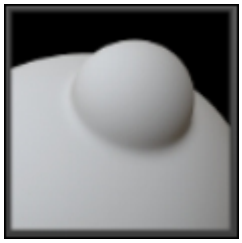
This is very useful to sculpt horns or tentacles. This brush is more interesting with Dyntopo. The problem with a mesh that uses a Multires modifier is that topology problems quickly appear with a lack of geometry. As long as the topology is dynamic, we can easily create an arm, a leg, or anything else. We can extend this as long as we wish the shape of our model to be.

6. Once the antennae are sculpted, we will add some polygons with the **Clay Strips** brush, then we will smooth them with the **Smooth** brush. We will magnify the extremity with the **Inflate/ Deflate** brush (refer to **18** in the preceding screenshot).
7. We will end this by digging forward a little bit with the **Clay Strips** brush in order to break the rounded shape.

Note

The Inflate/Deflate brush

This brush will allow you to inflate volumes by pushing the polygons in the normal's direction, or the inverse in the opposite direction in the **Subtract** mode. It can be very useful for meshes with closely spaced surfaces that are difficult to sculpt. This gives a very fast volume that makes it much more comfortable for sculpting.

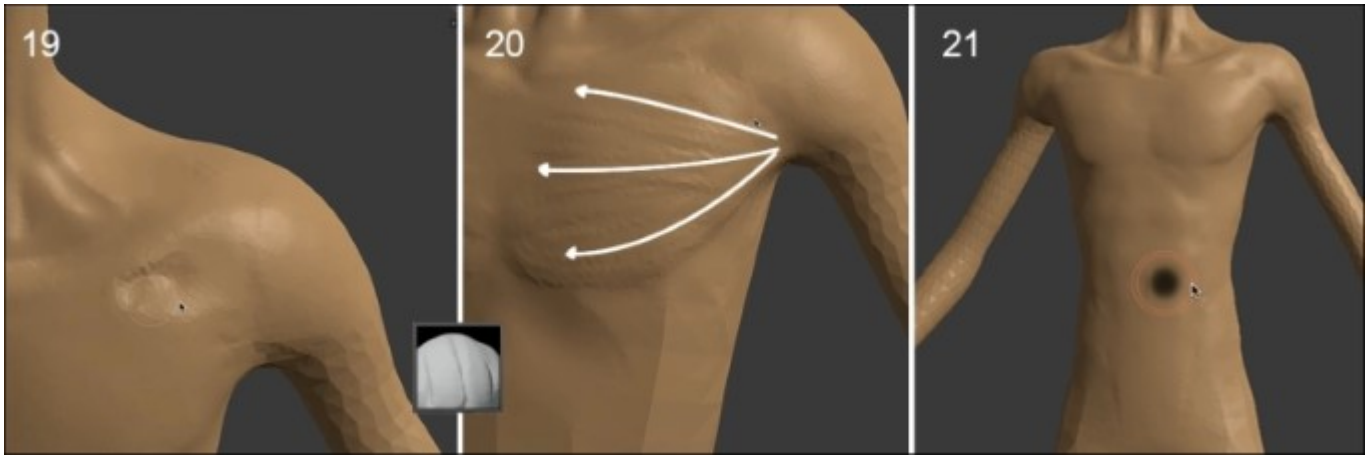


So our little alien has now its telepathy organs. We are going to sculpt the torso.

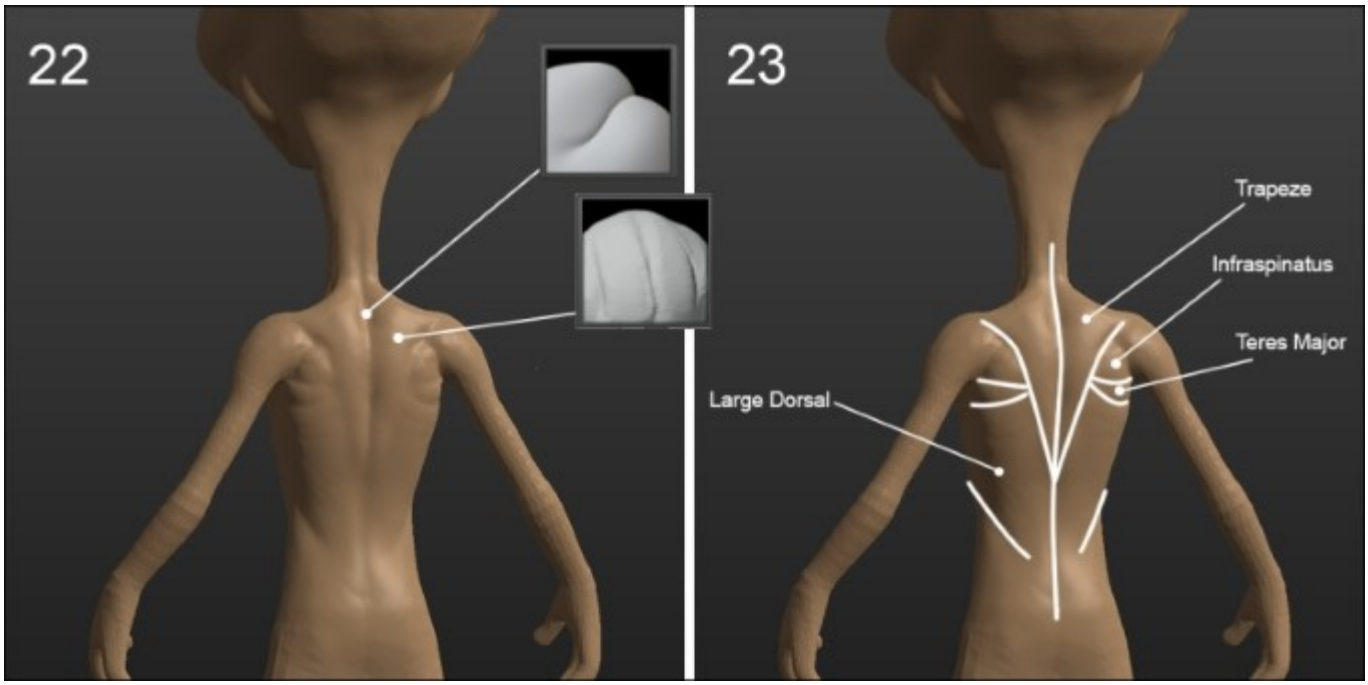
The torso

We will start the torso by sketching the pectoralis major muscle:

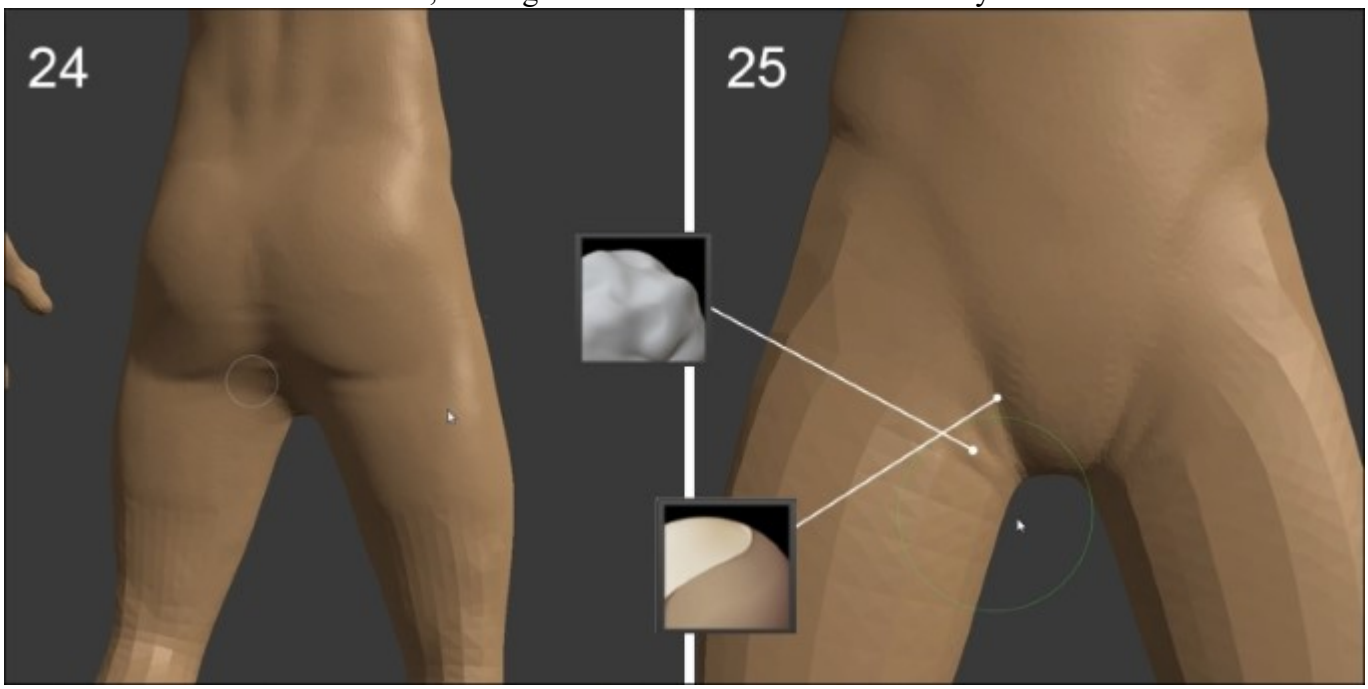
1. We will start by smoothing the surface and adding enough details.
2. With the **Clay Strips** brush, we will dig the dividing lines with the clavicle and the shoulder. (refer to **19** in the following screenshot).
3. Then gradually, we will add some volume accentuating the muscle fibers. The brush strokes start from the bottom of the shoulder at the clip with the biceps and go to the center of the chest (refer to **20** in the following screenshot). Get used to guiding your brush movements in the direction of the muscle.



4. Once the pectoral is sculpted, we will slightly accentuate the bottom of the chest and the abs with very light touches of the **Clay Strip** brush. This is to suggest forms rather than showing them (refer to **21** in the preceding screenshot).
5. We will then work on the back part of the alien. It is a complex part of the body. So we will start to draw the muscles (refer to **22** and **23** in the following screenshots) to gain visibility with the **Crease** brush. We can soften and smooth the muscle shapes. Then we will accentuate the spine with the **Pinch/Magnify** brush.



6. Now we will sculpt the buttocks. Avoid putting too much volume here. Turn the model around and observe the side view a moment, if necessary adjust the silhouette. Remember to draw a pinch line to accentuate the bottom of the buttocks with the **Pinch/Magnify** brush. This forms a fold between the buttock and the thigh (refer to **24** in the following screenshot).
7. We will add a few folds to show that it is a combination.
8. We will use the **Pinch** brush to accentuate the lower abdomen and pelvic bones (refer to **25** in the following screenshot). Unless you desire a different sexual orientation for our alien, feel free to add some volume to his crotch, it brings a little more realism. Don't be shy.



Note

The Clay brush

This brush allows you to add planar relief with a few soft edges. It is quite close to the **Clay Strip** brush that you already know but with a less sharp effect. It adds volume by raising with a low intensity. It is very good to refine organic shapes with precision.

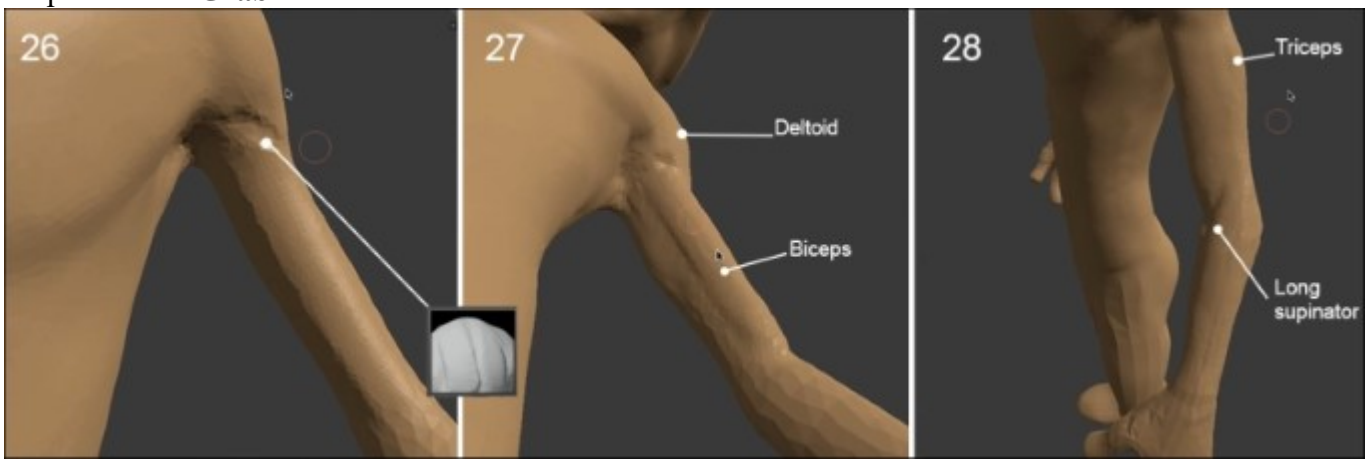
The arms

Now, let's start the arms. We can see in the drawing that they are pretty fine and not very muscular. His hands have two fingers and a thumb.

We won't need a lot of muscle details. We will be just interested in the major forms.

Let's begin the process:

1. We will start by digging the part between the shoulder and the biceps with the **Clay** brush (refer to **26** in the following screenshot). This accentuates the shoulders.
2. We will smooth a little, and then add some volume to the biceps (refer to **27** in the following screenshot). Slightly, we mark the outline of the muscles with the **Crease** brush and adjust the shape with the **Grab** brush.



3. Then we will work on the triceps that is located at the back of the arm. It is a muscle connected to the deltoid and covers the entire rear portion of the upper arm. We give it some volume by drawing the muscle fibers with some touches of the **Clay Strips** brush.
4. The forearm is a complex area of human anatomy. It is usually quite difficult to sculpt. It consists of several muscles that twist to ensure the mobility of the hand and the fingers. For our alien, we simplified it by lessening the muscle visibility. With the **Clay Strips** brush, we will draw the long supinator that emerges because it forms the junction with the end of the biceps (refer to **28** in the preceding screenshot) near the elbow. We will then add some volume to the elbow.
5. We will start from the elbow, and we mark a slight stroke of the **Clay Strips** brush in the direction of the wrist.
6. The wrist is reinforced by slightly accentuating the bones on the sides of the upper part.
7. We go and dig and slightly flatten the lower part of the wrist. (Refer to **30** in the following screenshot.)

The hand is also a fairly complex part of the body. We will not detail the anatomy here. We will try to focus on the main forms that compose it. Observe your own hands for better understanding of the forms.

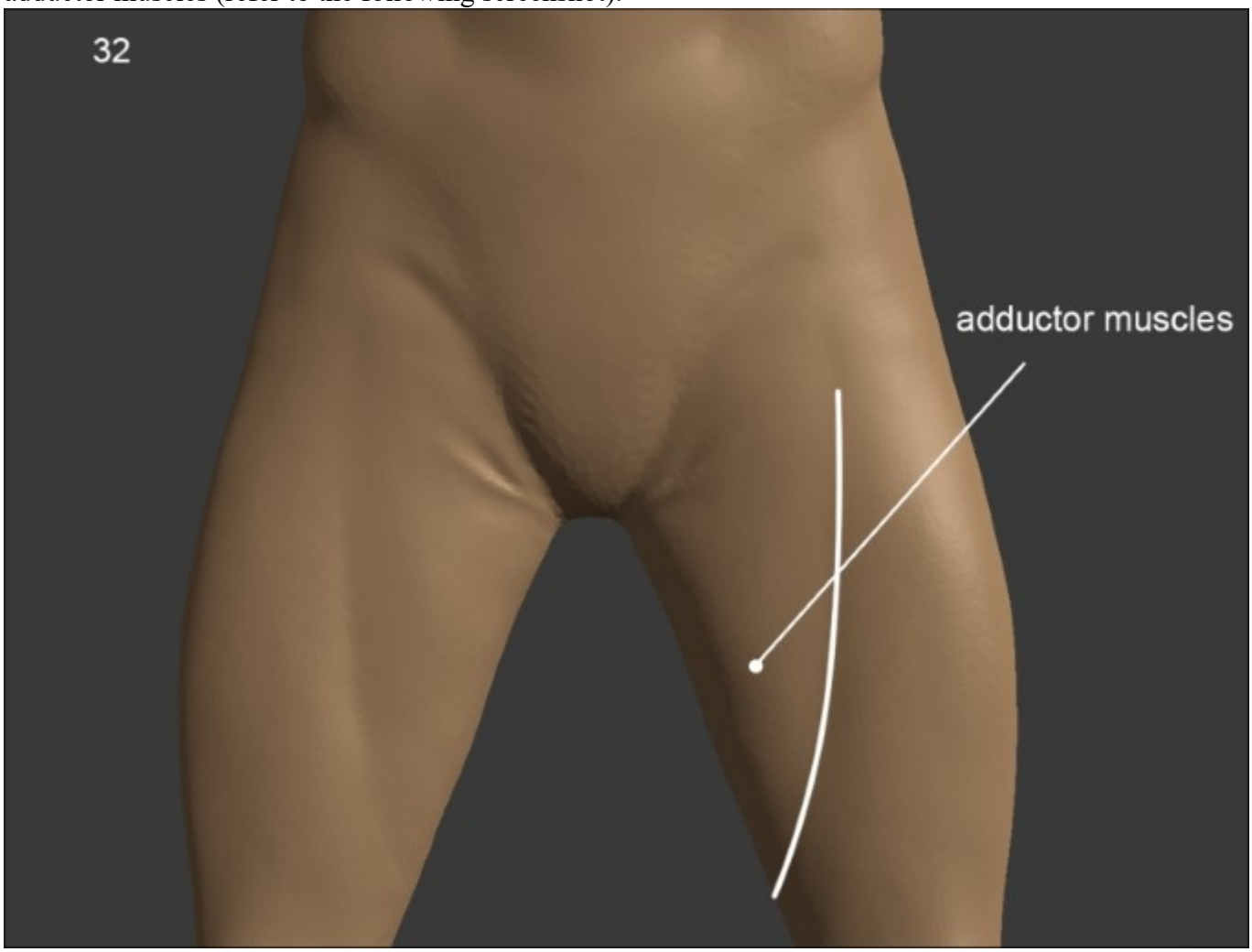
Now, we will begin forming gristle on the upper part of the hand with the **Clay Strips** brush.

1. We have placed a bit of volume to the different phalanges in order to accentuate them. (Refer to **29** in the preceding screenshot.)
2. There is some skin between the two fingers and between the index finger and thumb that we will dig. This skin allows the flexibility and elasticity of finger movements.
3. We will take the **Crease** brush and mark the lower part of the phalanges where the folds of fingers will be.
4. We will keep working with the **Crease** brush and draw the lines of the hands. The three main lines are enough to give the appearance of a palm (refer to **30** in the preceding screenshot).

The legs

We continue our sculpture with the legs. We can see that he has quite muscular thighs. The feet have a dynamic style that reflects the legs of a rabbit.

1. As with other parts of the body previously created, we will adjust the silhouette of the legs with the **Grab** brush before detailing the shapes.
2. We will slightly dig a line from the hip to the inside of the thigh that allows us to accentuate the adductor muscles (refer to the following screenshot).

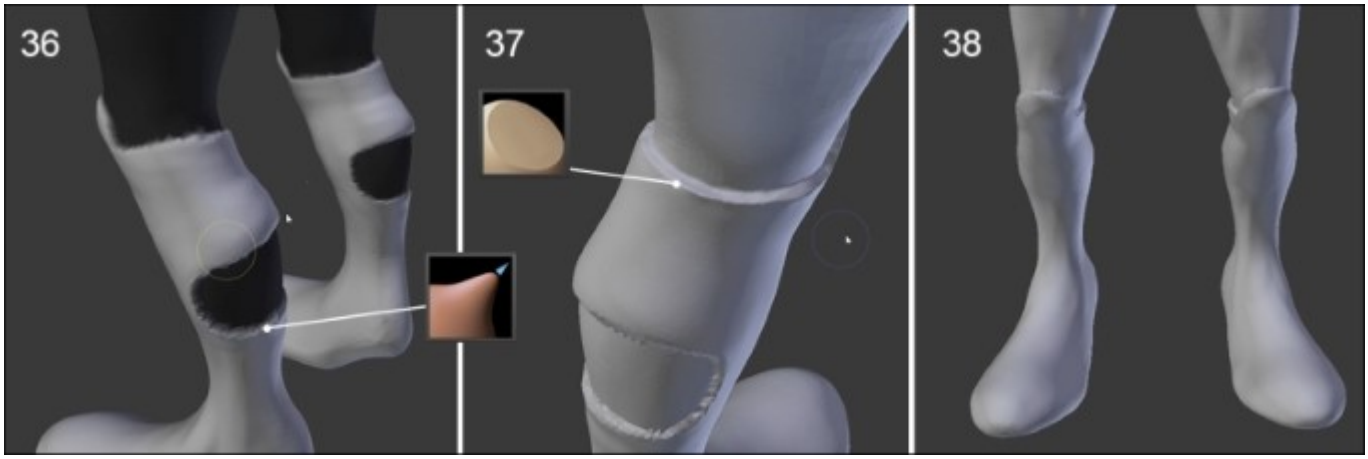


3. Now it is time to use the **Mask** brush that will be only shown without our Matcap activated, that's why you can't see the Matcap in the following screenshots. The boots are up to the knees and have a window over the calves (refer screenshot **34**). It is necessary to have enough polygons in order to get the mask contour sharp.
4. In order to highlight the edges of boots, we will reverse the mask with the shortcut *Ctrl + I* (refer to **35**).

Note

The Mask brush

This is a quite special brush. It allows us to mask an area of the mesh. It means that this area stays unchanged when any other brush is used as long as it is masked. Thus, we can create shapes that would be impossible to do otherwise. The uses are many. It is very useful for extruding surfaces.



5. With the **Grab** brush, we will pull the polygons at the edge of the masking. Then we slightly raise them (refer to screenshot **36**).
6. We increase the level of detail to enhance the edges of the boots. We mark the separation with the **Flatten/Contrast** brush. We need to zoom enough and adjust the brush size (*F*) accordingly.

Note

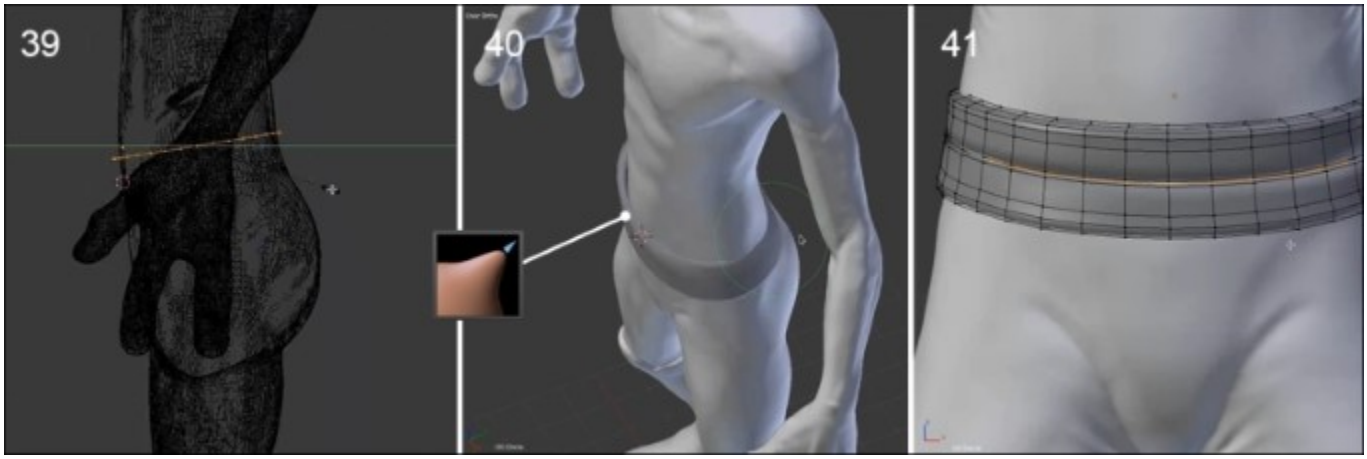
The Flatten/Contrast brush

In the **Flatten** mode, this brush allows us to smooth over a surface while digging slightly, or otherwise in the **Contrast** mode, it greatly increases the height of the relief. These two very different functions make it an even more interesting brush.

By using the same technique, we can sculpt the collar in the neck area.

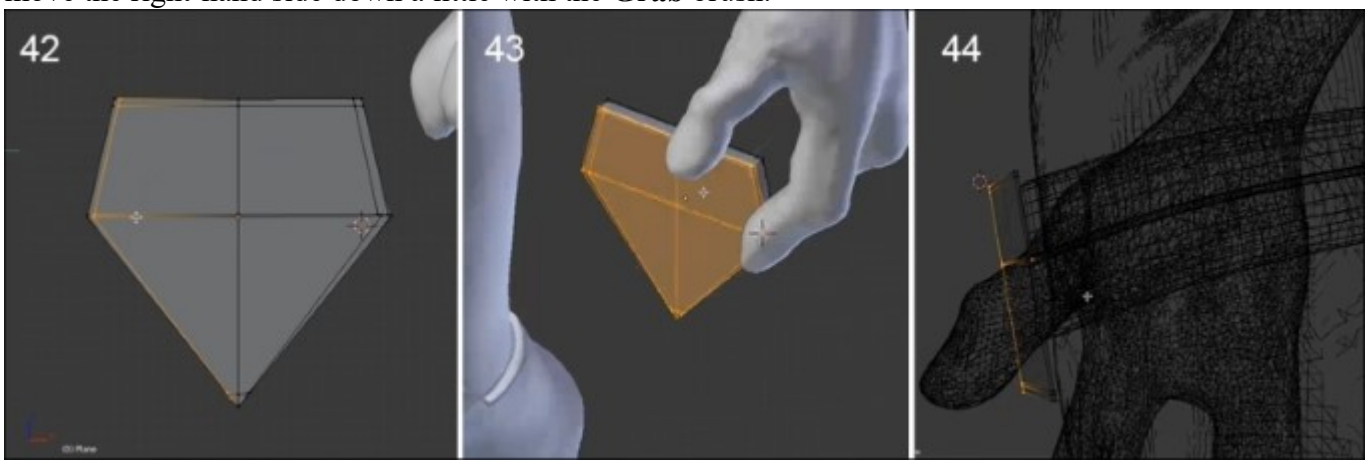
The belt

The belt needs to be treated separately because it's not about sculpting. We are going to use more the traditional tools that we saw in the previous chapter. But as you'll see, it is very interesting to mix the different techniques that you've learned in this chapter with polygonal modeling tools. That's why we are going to use the **Grab** brush in order to wrap the belt around his waist.



We will start the modeling of the belt with a primitive circle. After this, we will then place our cursor at the center of the character in the **Object Mode** from the top view in order to add a new circle with 32 vertices.

1. In the **Edit Mode** (*Tab*), and with the **Wireframe** option of **Viewport Shading** on, we will adjust the size of the circle by scaling on the Y axis (*S* + *Y*). We then rotate it a little bit in order to match with the shape of the alien (refer to **39** in the preceding screenshot).
2. We will then extrude the circle to form the height and the thickness of the belt. As we said before, we can now use the **Grab** brush (in the **Sculpt Mode**) in order to stick the belt to the waist. In our case, it's as if we were using the **Proportional Editing** tool (refer to **40** of the preceding screenshot).
3. Then we will go back to the **Edit Mode** in order to add more resolution with the Loop Cut tool (*Ctrl* + *R*). We will also place a loop cut in the middle of the belt on which we will add a little **Bevel** (*Ctrl* + *B*).
4. In the middle of the bevel, we will add a new edge loop that we will scale along the normals (*Alt* + *S*) (refer to **41** in the preceding screenshot).
5. We can now switch back to the **Sculpt Mode**, and with the X symmetry option off, we can move the right-hand side down a little with the **Grab** brush.



Now that we've finished the belt, it's now time to add the belt buckle as follows:

1. In the **Object Mode**, we will add a new plane.
2. Then we will go in the **Edit Mode** (**Tab**) and add a horizontal and a vertical edge loop (*Ctrl* + *R*).
3. We will then resize these edge loops so that they form a diamond shape (refer to **42** in the preceding screenshot).
4. It's now time to add a **Subdivision Surface** modifier.
5. We will then add some edge loops on both sides in order to maintain the diamond shape.
6. In order to add thickness to the buckle, we will do some extrusions of the whole geometry (*A* and *E*). As always, we will maintain the shape with the Loop Cut tool (*Ctrl* + *R*).
7. We will also scale the front polygons of the buckle.
8. Finally, we can place our belt buckle at the right place in the **Object Mode**.

There you go! Our little alien is ready for crazy galactic adventures!

A render of the final alien sculpt with Blender Internal Renderer

Summary

In this chapter, you've learned a new modeling technique that is best suited for your organic models. By mixing this method with polygonal modeling techniques you will be able to create awesome characters in a very short time! If you have a powerful computer, you can go into further details such as skin pores and wrinkles using alphas, for instance. For now, the alien character doesn't have a good topology, so we will learn how to create a new topology over the model and extract the details of our sculpting in the next chapter.

Chapter 4. Alien Character – Creating a Proper Topology and Transferring the Sculpt Details

This chapter will be more technical than the previous one. We will see how to create a production-ready character with a nice topology, starting with the sculpture of our little alien. Of course, we can't go through all every possible techniques to reach our goal but you will have a solid understanding of what a good organic topology is. You will also learn how to retrieve the details of a sculpture with a normal map. Furthermore, you will learn how to enhance the look of the alien with an ambient occlusion. These maps could be a good starting point to create a more complex and rich texture later. As you learn more and more tools, you will be able to have more possibilities to express your imagination. What you really need to grasp in this chapter is the logical way of doing a good topology, because each object needs one topology according to your needs. So, let's dive in!

This chapter will cover the following topics:

- Understanding the retopology process
- Using the UV unwrapping tools
- Baking normal maps and Ambient Occlusion
- Displaying the baked maps in the 3D viewport
- Making a good topology

We will now create a retopology of our sculpture by using some of the tools that you've already encountered during the robot toy modeling, and some new tools. But, wait a minute, why are we remodeling our alien if we have already sculpted it?

Why make a retopology?

The main goal of doing a retopology is to have a clean version of the sculpture with a good topology. It means that the mesh geometry needs to follow the shapes of the sculpture by defining proper edge loops. A good topology is also a must-have when you want to animate an object, and it's even more important when you are dealing with organic shapes. The muscles need to correctly bend, so this is why we are following them with edge loops. But, of course, we can't delimit each of the muscles, so we are thinking more about their overall form, treating them as groups, such as the pectorals muscles.

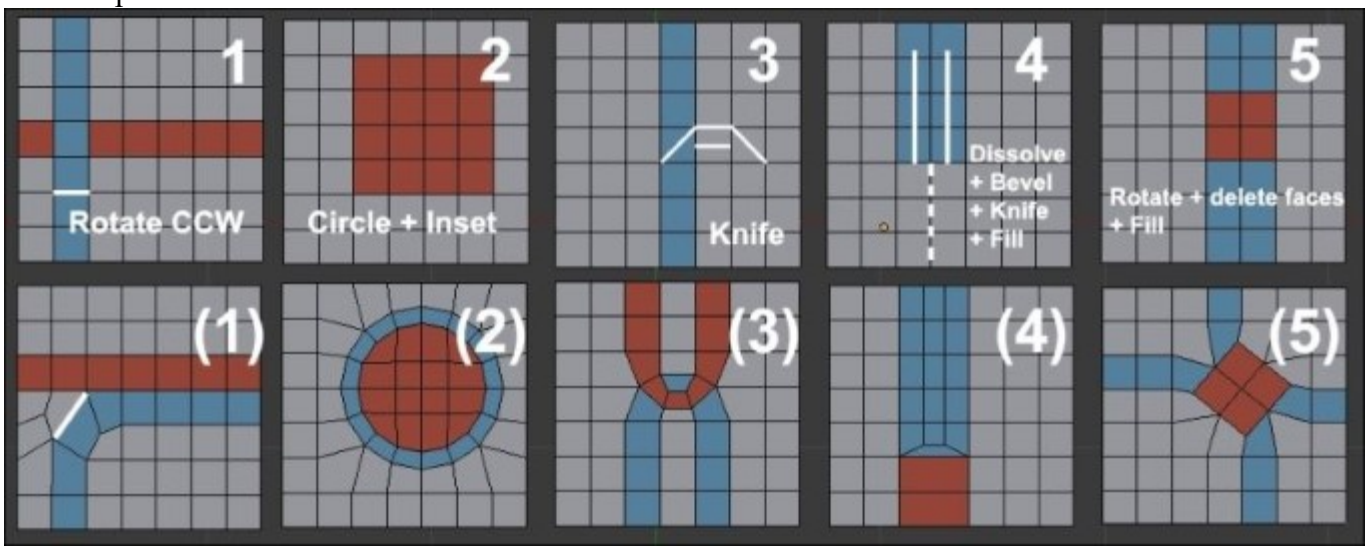
Another goal of a retopology is to have a less dense object. I don't know whether you have already made the mistake of entering into the **Edit Mode** of the alien sculpture, but if this is the case, you have seen a tremendous amount of polygons organized in a fancy way. Technically, the process will be as easy as adding new geometry that snaps to the sculpture. However, it could quickly turn out to be a puzzle if you don't know what you are doing.

Possibilities of arranging polygons

As we have mentioned before, we will be using pretty much all of the poly modeling tools that we've learned previously, such as the grab, rotate, or face creation tool. But what really matters while doing a retopology is the arrangement of the polygons through the loops that defines the shape. In this section,

we will give you some useful techniques in order to help you rework the flow of your topology. Before reading further, we advise you to train yourself on a subdivided plane:

1. Create a new plane (*Shift* + *A*).
2. Select it in the **Edit Mode** and under the **Specials** menu (*W*), select the **Subdivision** option.
3. Redo step 2 thrice.



Five topology cases you may encounter

We will now go through the five cases presented in the preceding screenshot.

In many cases, you will need to change the direction of a loop. As you can see in the first case of the preceding screenshot, we have two colored face loops. We will rearrange them so that the blue one doesn't cross the red one. Remember that in order to select a face loop, we will use the *Alt* + RMB shortcut.

1. To resolve the first case, we will select the bottom edge of the blue face situated below the cross intersection.

   We will then select the **Rotate Edge CCW** (Counter Clock Wise) in the **Edge** menu (*Ctrl* + *E*).

   Now, we can rearrange the polygons to get a nice round corner.

   Sometimes, you may want to have a circular shape in your topology. This occurs mainly in hard surface modeling.
2. In order to resolve the second case, we will select the piece of geometry that will be of circular shape and use the Mesh Loop add-on (by pressing *W* and selecting **LoopTools** | **Circle**) to form a circle. You can also move your vertices one by one if you don't want a perfect circle.

   Now, we will maintain the geometry by doing an inset of the selected faces. As you can see, our circle perfectly incorporates the flow of our geometry.

Another situation that you may encounter is when two face loops forming an arc are stuck together. This technique can be useful when defining muscles or articulations. We will later use this for the knee of the alien.

3. For the third case, we will start with the Knife tool by cutting the three edges that form an arc.

   As you may have seen, we now have two triangles that we need to resolve in quads. So, we will add a cut in the middle of both of them with the Knife tool or the Loop Cut tool.

   You will often need to reduce the number of polygons at some location. For instance, there could be lot of condensed polygons behind the head, so in order to have less of them for the back, we will make a U shape.
4. So, our goal for the fourth case is to make a U shape with blues faces that leaves us with only one face under it:
   1. To do this, we will first dissolve all the vertical vertices that are in the middle of, and below, the blue faces (the dashed line in the screenshot).
   2. Now we can do a bevel of the edges that separates the two vertical lines of the face.
   3. We can now use the *J* key to join the two vertices that form the base of the U shape (the one marked in the screenshot).
5. We will now show you one last technique, but you need to keep in mind that there are many other methods that we can't show you here because it would require a whole book! Sometimes you need to rotate some polygons in the geometry, but you don't want stretches after rotating them.

   If you look at the last case of the preceding screenshot, the red square of the polygons has been simply rotated with the *R* key.

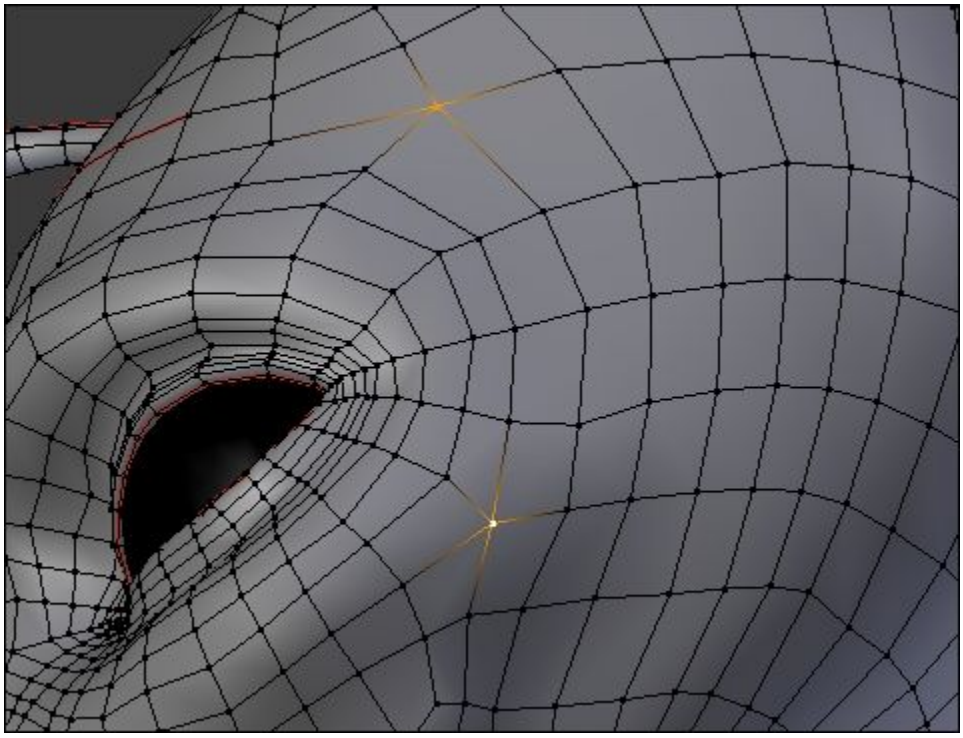
   At this point, the geometry around it should be stretched. The best way to solve this is to simply remove the parts that are causing you problems and recreate the geometry in a better way. In our case, we've done this by bridging the square outline back to the rest of the geometry with the *F* key.

Errors to avoid during the creation of retopology

We will now talk a little bit about the main problems that you should avoid while creating a proper topology. The first thing that we have already expressed before is that we need to try to have as few triangles possible. Triangles are bad because they break the face loops, and they can cause some rendering artifacts with the lighting, and the topology could be harder to maintain.

Some people may think that this is not a problem because triangles are used a lot in the video game industry, but usually, the triangles that we see in a game model result from a tessellation made by the game engine after the model was created. They are also needed in cases where there is a lack of performance, especially in mobile games where the amount of geometry needs to be lowered. But the performance of smart phones are doing better with time, so this is not going to be a problem in the future. Sometimes, triangles can be placed at a position that doesn't bend a lot. For instance, the ear of a human rarely deforms, so it's not a problem to place a triangle here. When you add triangles, do it in such a way that it doesn't bother the silhouette of your object.

Another thing that you want to avoid is poles. A pole is a vertex that connects a minimum of four edges. Usually, poles are useful to redirect face loops, so they are usable but they need to be carefully managed. You can create poles if, and only if, you need to change the flow of your face loops and if you are in a place where the geometry won't bend a lot during animation. For instance, on the human face, we can place poles on the cheekbone because we need to redirect the topology. You will encounter this with the alien's head.



An example of two selected poles

Don't be discouraged if you don't get the topology right the first time, come back later and you'll have a clear mind to try to figure out the problem again. Another thing to keep in mind while doing a retopology is that if you want to have a certain loop, create it without waiting because you'll be quickly overwhelmed by polygons, and you could face problems when you need to remove a lot of geometry in order to connect the loop back. Creating a good topology is like solving a puzzle—it always has a solution!

Density of polygons

While doing a retopology, you need to think about the general flow of your topology. If you've done this right, you'll be able to quickly add or remove edge loops in order to increase or decrease the density of the mesh. The more geometry you have, the more you will be able to be precise in the approximation of the sculpted shape. Of course, you need to take into account the eventual constraints that you will have. For instance, if you are creating a mobile game character, it's best to try to reach the minimum number of polygons that gives you the global silhouette. Choosing the right number of polygons is more of a decision that you'll take on the fly.

Making the retopology of the alien character

Now that we have seen a few basic techniques, we are going to see a practical case by working on our little alien character sculpture.

Preparing the environment

Before starting the retopology of our little character that came from distant worlds, we must prepare the working environment for the method that we will use. There are several possibilities in Blender to do a retopology, including with very good add-ons (Retopology MT and Retopology Tools, for example). We will not use them here; instead we will focus mainly on internal Blender tools.

1. We will start by placing the cursor at the center of our sculpted mesh. For this, we will select the mesh and press *Shift* + *S* to open the snap menu. Then, we will choose the **Cursor to Selected** option.
2. We will then create a plane in the **Object Mode** (pressing *Shift* + *A* and selecting **Mesh** | **Plane**) that we will place in front of the eyes. This new object that has been created is going to be the new mesh for our character. We rename it as `Alien_Retopo`.
3. In the **Edit Mode**, we will bisect the plane, delete the vertices on the left-hand side, and then we will add a **Mirror Modifier**. Don't forget to check the **Clipping** option.
4. We need to make a transparent shader that will allow us to work comfortably in order to visualize the mesh. In the **Material** menu, we will check the **New** button. It's also good to name this; in our case, it is `M_Retopo`. This way, we can look through our polygons and check the **Transparency** option with an **Alpha** value of `0.208`.
5. We will then go in the **GLSL** mode in the **Shading** menu of the right panel of the 3D viewport (*N*). This allows us to visualize the new material that has been created.
6. We will also check the **Backface Culling** option just below the **Shading** menu to avoid being too bothered by the rear faces.
7. We switch to the **Texture shading** mode in the corresponding drop-down menu located in the 3D view header.
8. After this, we need to add a **Hemi** light (by pressing *Shift* + *A* and selecting **Lamp** | **Hemi**) that is to be placed just above the 3D model in the direction of the ground. We will set its energy value to **1**.
9. We will activate **Xray** in the Properties editor in the **Object** tab under the **Display** subpanel.
10. For our comfort, we need to change the size of the vertices. For this, we will go to **File** | **User Preferences** | **Theme** | **3DView** | **Vertex Size**. We will enter a value of **6**.
11. In the **User Preferences** menu, we will check that the **F2 add-on** is activated.
12. In the 3D viewport header bar, we will activate **Snap** (it looks like a little magnet) in **Face Mode** (just on the right-hand side of the snap icon).
13. We still have to activate the buttons located a little more on the right-hand side called as **Project Individual Elements** on the surface of other objects and **Snap onto itself**. Let the mouse hover over a button for a moment so that you can see its tooltip appear.

Now we can start to build the new topology.

The head

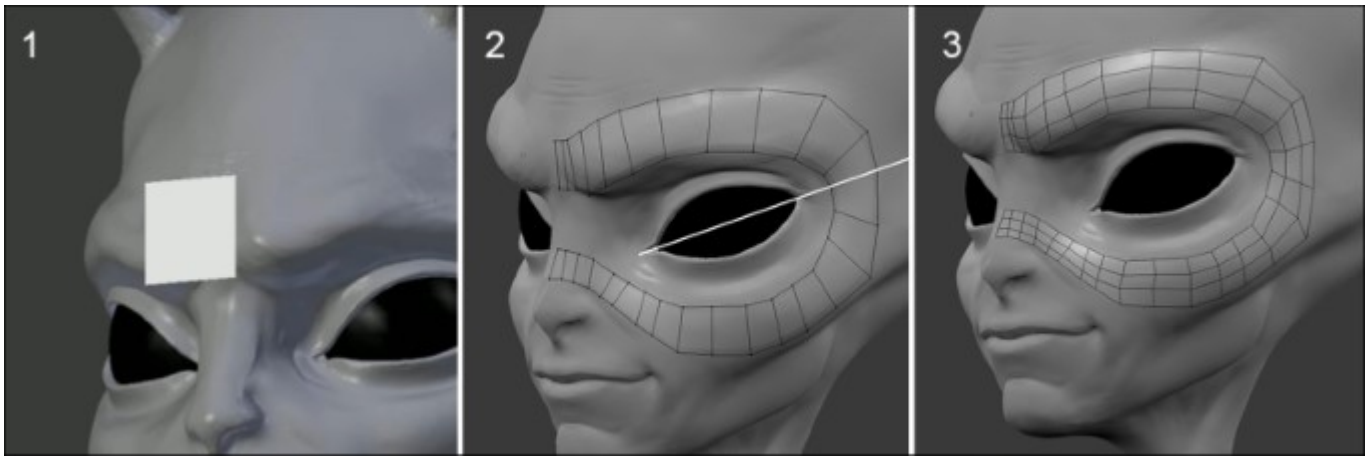
Let's start our retopology with the head of our alien character:

1. In the **Edit Mode**, we will snap our first polygon with the Grab tool (*G*).
2. We will then begin to form the first face loop around the eyes. We will select the edge on the right-hand side of the plane and pressing *Ctrl* and a left-click, we will create new polygons in sequence on the surface of the sculpted mesh. There is no need to be very accurate while tracing the first face loop. We will be able to reposition the vertices and adjust the number of polygons thereafter. So, we will make the first face loop, which passes through the eyebrow, the cheek, and joins the symmetry axis right at the nose. At the corner of the eye, we need to be careful while placing an edge that extends the diagonal direction of the eye (refer step **2** in the following screenshot). This face loop is called the **mask**.

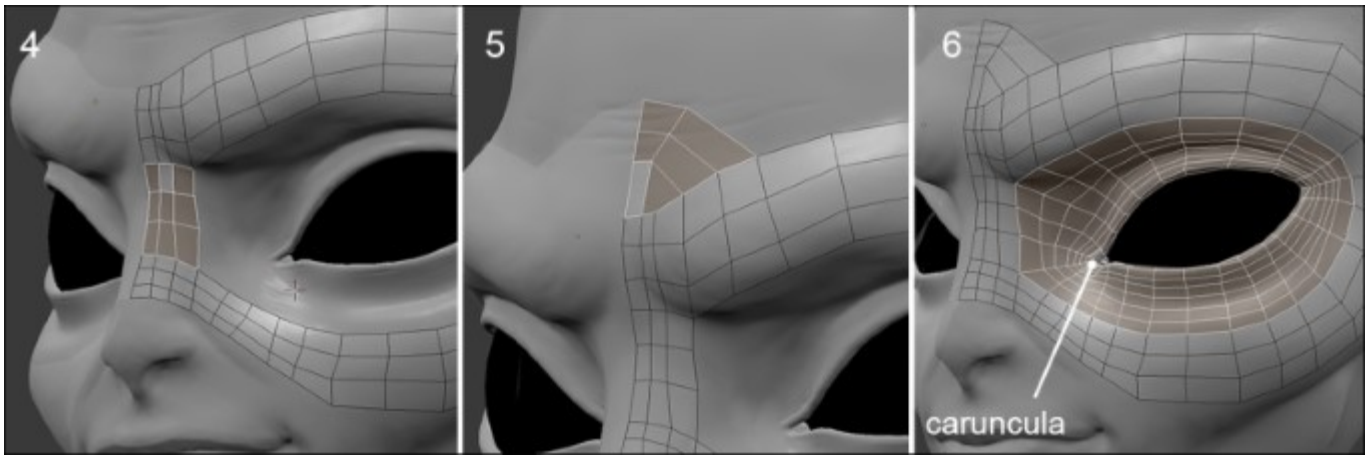
Note

Pair number of polygons

Be careful to always have a pair number of polygons whenever you form a face loop. This will allow you to connect the polygons more easily. If in some cases you only have an odd number, you can try to delete an edge loop and replace the polygons around (with **G**) it.



3. We will then add two edge loops (*Ctrl + R*). Try to equalize as much as possible the size and the distribution of the polygons. We will reposition the vertices to have a rounded shape (refer to step **3** in the preceding screenshot). For our selected components snap on to the sculpture, we have to move them with the Grab tool (*G*). Remember to do this often, but only with the components that directly face your point of view.
4. We will add an edge loop to the nose in order to have enough polygons to better define the outline shape. We will have a 28-face face loop.
5. We will join the nose with the new polygons (refer to step **4** in the following screenshot).
6. To save time, you can start making a shape with a few big polygons to cover a large area that you want to work on. You can then define the number of cuts that you need and snap the topology, rather than make many little polygons one after the other.



Try to follow the shape of the sculpted mesh as much as possible.

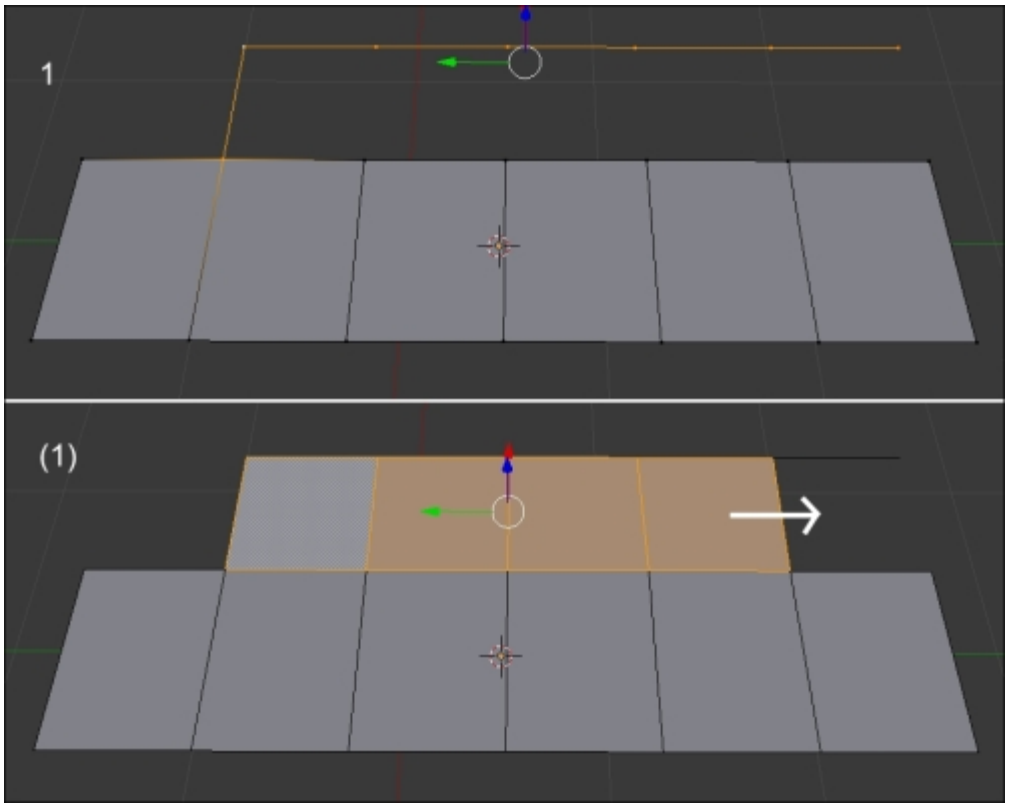
Note

F2 add-on

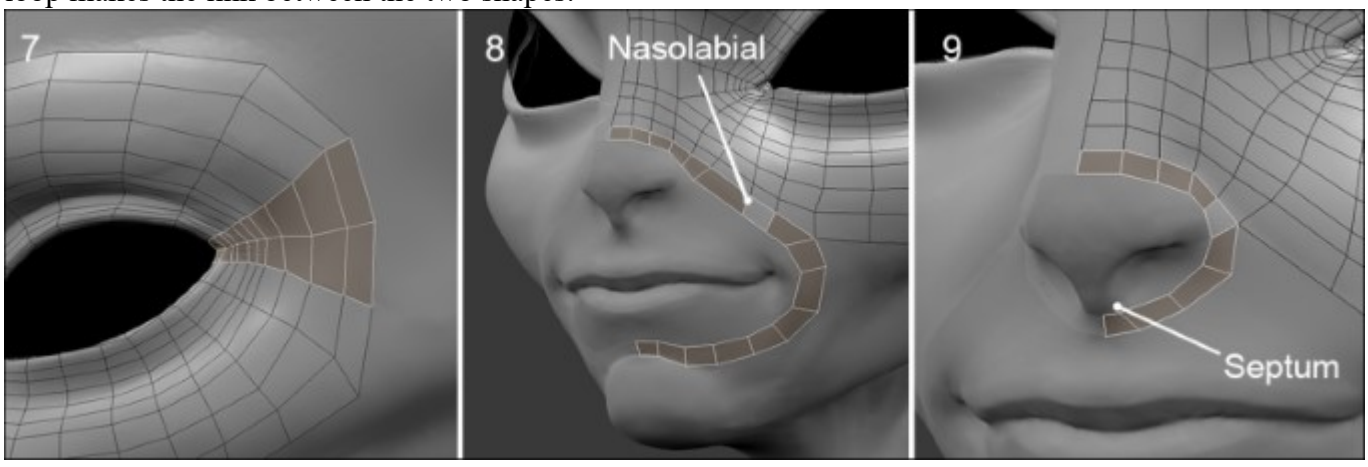
We mentioned this while preparing the environment. *F2* is an essential add-on in Blender that allows us to quickly create faces.

Between the two rows of the edges, the *F2* add-on can generate the missing faces. You can also generate a face by selecting one vertex on the intersection of two faces by pressing *F*. Repeat the process to quickly make a face loop, and remember to weld them (select the polygons, and then press *W* and select **Remove doubles**).

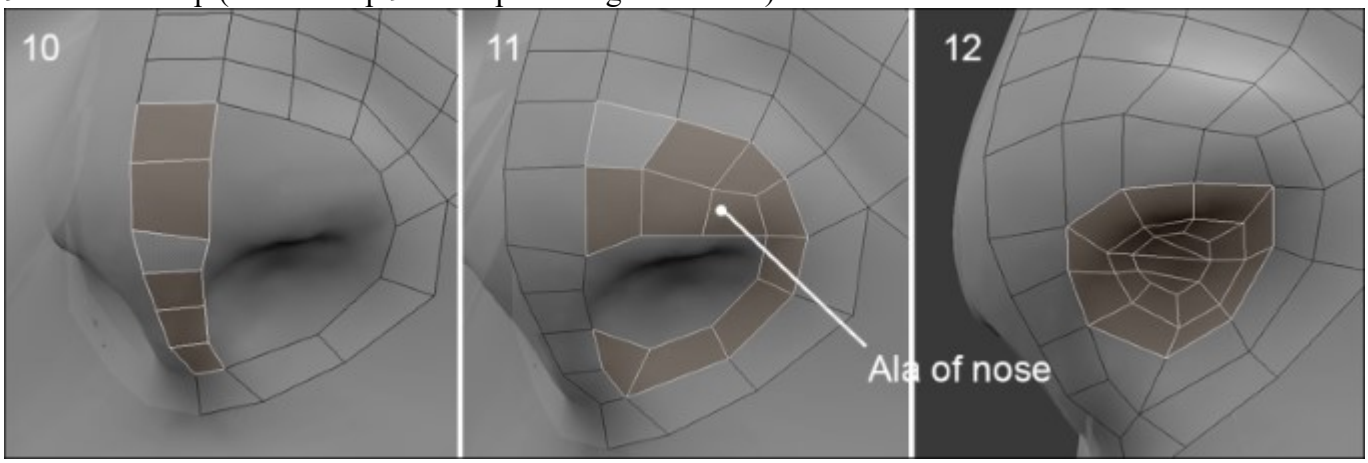
You can also generate the face loops in the edge mode. You need to select a starting edge (the one to the left and perpendicular to the *x* axis according to the following screenshot) by pressing *F*. The sense of creation of a face can be determined by the placement of the mouse if the starting edge is at the center of the face loop that is to be created.



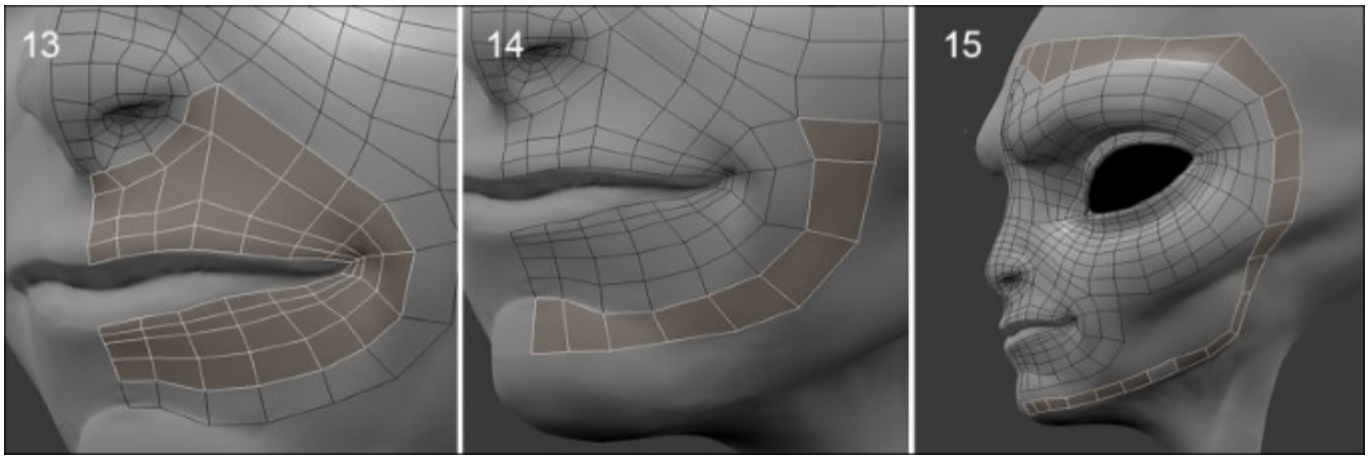
7. We can see that we have a pole. We will avoid making it too visible, so we will place it under the eyebrow.
8. We complete the topology of the upper nose with a polygon reduction on the forehead (refer to step **5** in the preceding screenshot).
9. Now, we will select the edge loop that bypasses the eye, and we do an extrusion by changing the scale (*E* and *S*). We will then place the vertices on the outline of the eye (*G*). We will have a 24-face face loop (refer to step **6** of the preceding screenshot).
10. We will add two edge loops (*Ctrl* + *R*) around the eye that we will reposition correctly (*G*). A first edge loop takes the shape of the eye, marking the fold with the eyebrow. A second edge loop makes the link between the two shapes.



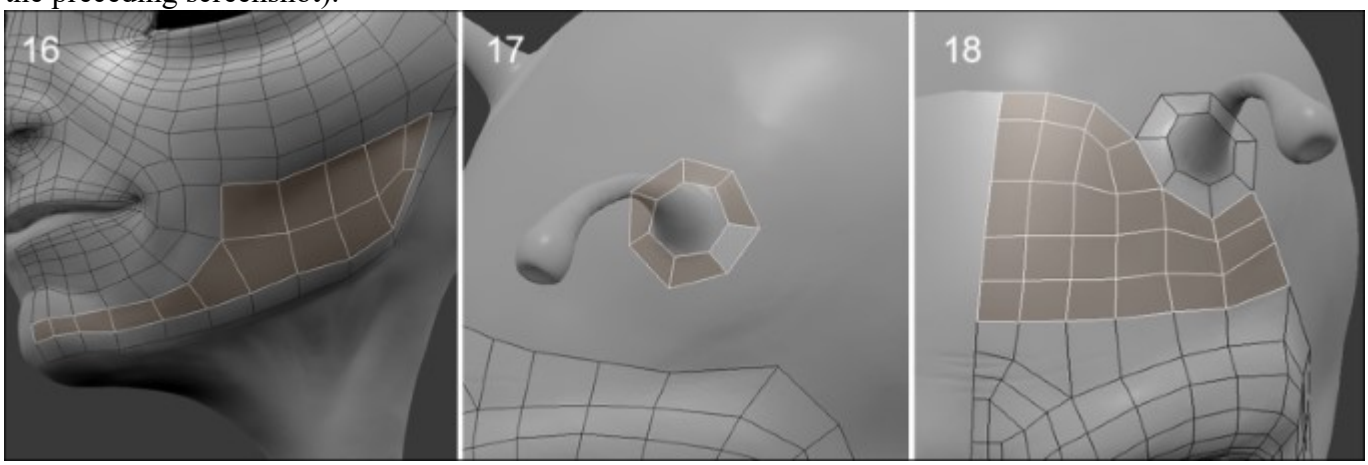
11. At the left outer corner of the eye, we will tighten the vertices. We will take care to have a conical shape that spreads outwards (refer to step **7** in the preceding screenshot).
12. Now, we create a face loop from the nose to the chin. It is called the nasolabial loop. We will add the polygons using *F2* (by selecting a corner vertex and pressing *F*). Just like the eye, we extrude an edge and form a face loop from it that follows the circular shape of the mouth. We will have a 16-face face-loop (refer to step **8** of the preceding screenshot).
13. We will add a face loop that comes around the nose. This allows us to define the nose. Remember to slightly bring the edges at the bottom to the septum of the nose. We will have a 9-face face loop (refer to step **9** of the preceding screenshot).



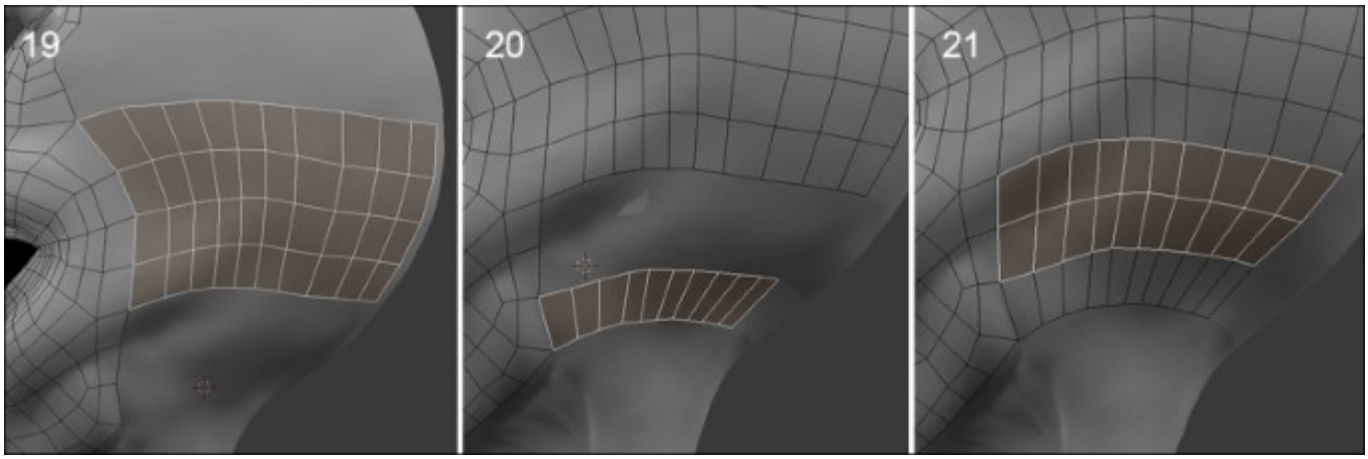
14. We will create the topology that forms the top and bottom of the nose. We will have a 6-face face loop (refer to step **10** of the preceding screenshot).
15. Once the nose shape is outlined, we will begin the ala of the nose. We will extrude a face loop that starts from the top of the nose down to the bottom of the nose, avoiding the nostril. This arrangement allows us to better create the shape of the nostril loop (refer to step **11** of the preceding screenshot).
16. We will extrude the edge loop of the nostril with a scale transformation (*E* and *S*). We will then close the hole by connecting the edges (refer to step **12** of the preceding screenshot). If you don't have enough edges to make four-sided polygons, you can use a triangle. It must be as hidden as possible.



17. We will start from the corner of the nose to make a face loop that goes under the mouth.
18. We will then close this part by following the shape of the lips. Take care to slightly pinch the corner of the mouth. The topology must be as symmetrical as possible between the top and bottom of the mouth in order to connect them later. We will have a 14-face mirrored face loop (refer to step **13** of the preceding screenshot).
19. We will continue around the mouth by adding a polygon strip above the chin (see **14** and **15** in the preceding screenshot).



20. Now, we will make the face loop of the jaw, which ascends to make the contour of the face. We have a 22-face mirrored face loop (refer to step 15 of the preceding screenshot). We need to align the faces correctly for an easy connection and close the area of the jaw (refer to step **16** of the preceding screenshot).
21. We will make the contour of the antennas with an extruded vertex on the top of the forehead in a circular manner (refer to step **17** of the preceding screenshot).
22. We will then continue to form the polygons of the forehead, in line with what was done, by making a reduction in the number of faces, avoiding the antennas (refer to step **18** of the preceding screenshot).

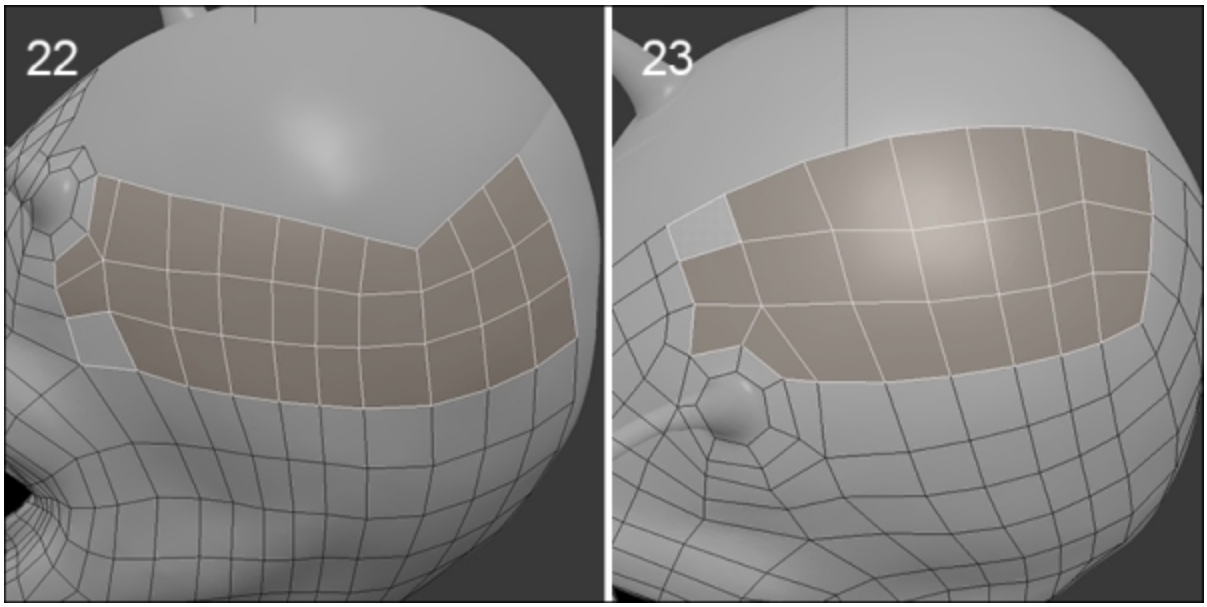


23. We will select three edges on the temples and create a polygon strip that follows the back of the head. We need enough faces to have a nice rounded.
24. We will then select the edge located at the jaw, and we will continue to make the topology of the back of the head (refer to step **20** of the preceding screenshot).
25. Now that the face strips are facing each other, we can fill the missing topology with the Fill and Loop Cut tools. (Refer to step **21** of the preceding screenshot.)
26. We will continue to make the faces on the back of the head in the same way. We will form an angle to close the top of the head later more easily. (Refer to step **22** of the following screenshot.)

Note

The Smooth option

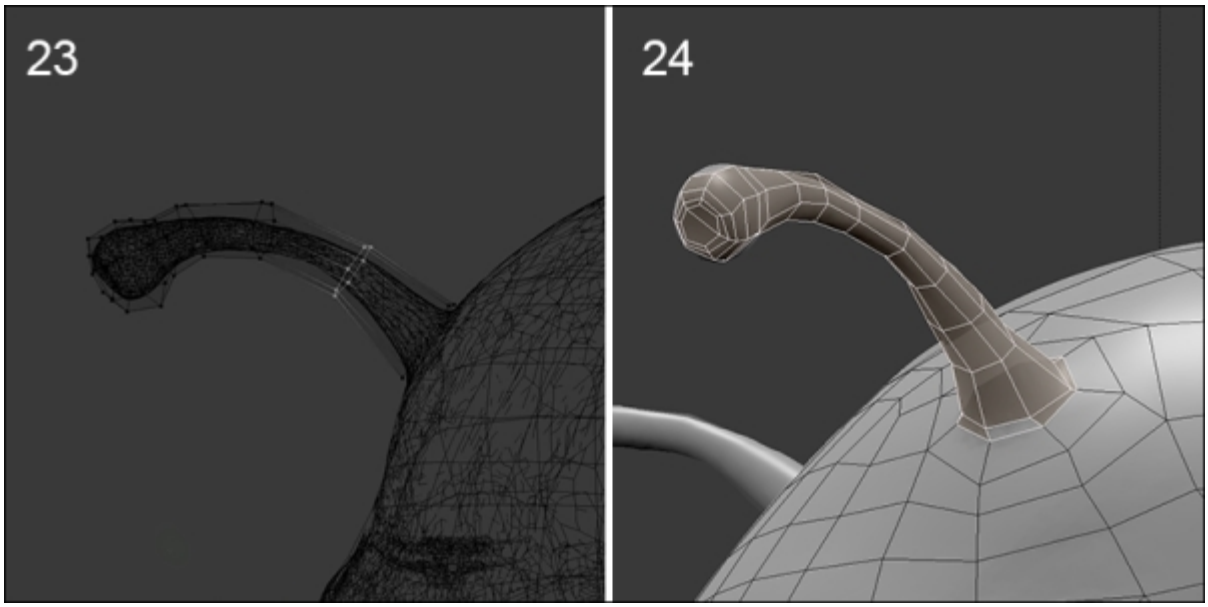
In order to have a topology as homogeneous as possible, we can use the **Smooth** option of the **Specials** menu (**W**). This allows us to relax the polygons. We first need to select the faces we want to modify (**C**). Be careful as you have to snap them after the process (**G**), so select only the visible faces in the view.



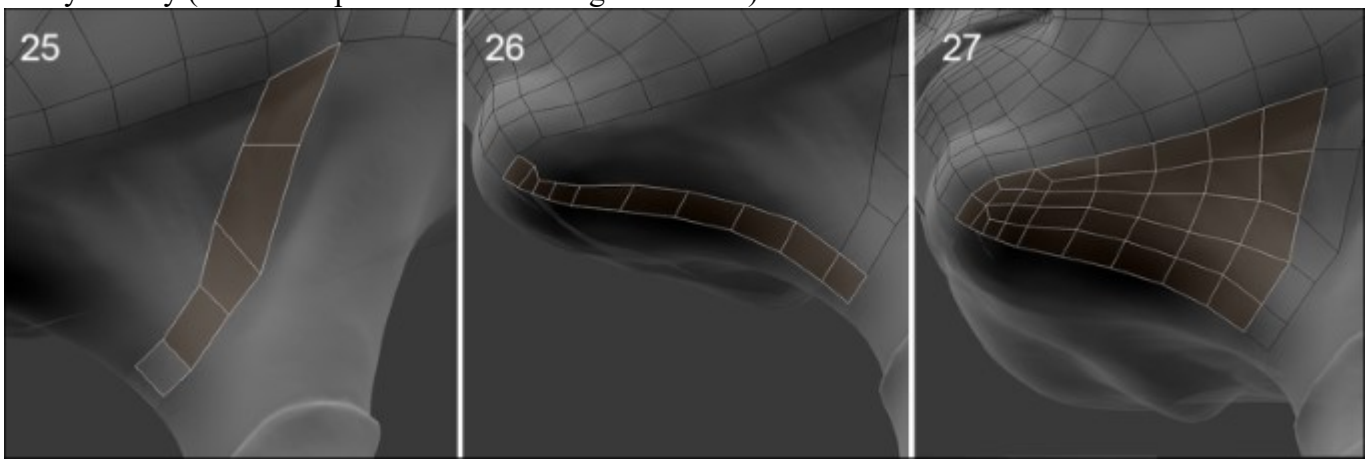
We now need to close the topology of the top of the head. We must pay attention to respect the number of edge loops created previously. You can apply the smooth option to quickly relax some misaligned faces.

Now we are going to make the antennas that require another technique:

1. We will duplicate the edge loop situated at the base of the antenna already created to make a new object (by pressing *P* and select **Selection**).
2. We need to deactivate the **Snap** option and the **Project Individual Elements** option.
3. We can extrude this new object following the shape of the antenna. The polygons must completely cover the surface of the sculpted mesh. You can set **Wireframe Shading Mode** in order to view your geometry better.
4. Once we get the desired topology, we add **Shrinkwrap** modifier and select the sculpted mesh in the **Target** parameter, called **Body**. This modifier allows us to snap a mesh on another.
5. We must apply the modifier by clicking on **Apply** button, and then we will join it back to our other piece of geometry. For this, we select the mesh of the antenna, then the mesh of the head, and we press *Ctrl* + *J*.
6. As we want to snap our components on the sculpture again, we will restore the **Snap** and **Project Individual Elements** options.
7. We can add a few edge loops to the antenna in order to get a good enough shape.
8. We will connect the faces in the back of the head in continuity with the topology already done.



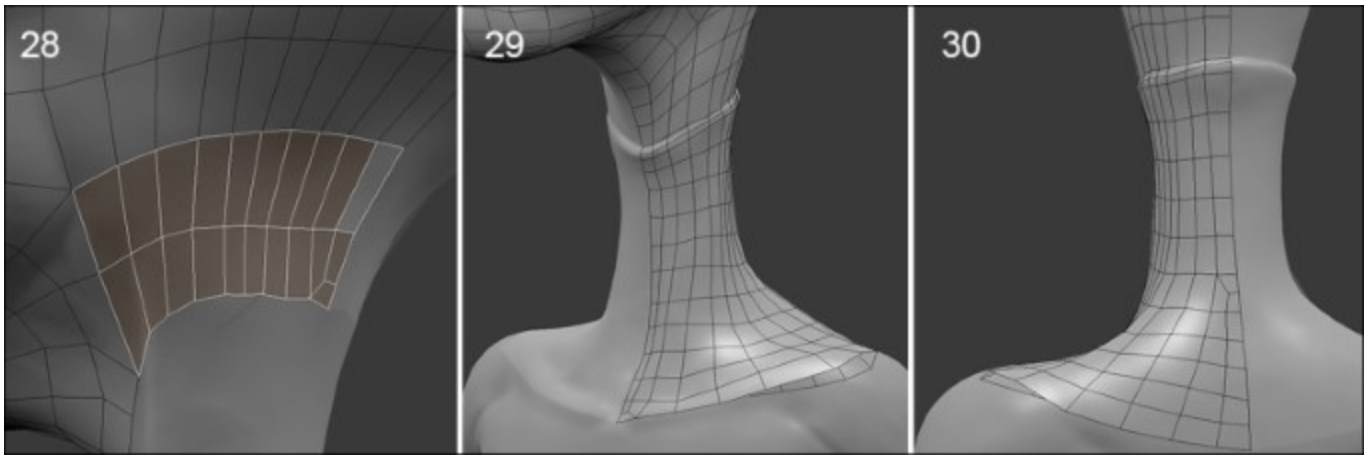
9. From the angle of the jaw, we will create a face loop that ends at the symmetry axis on same level of the glottis (refer to step **25** of the following image).
10. We will then connect the face loop to the chin by adding enough edges. We need to connect the entire underside of the jaw. We have a 10-face face strip (refer to step **26** of the following screenshot).
11. This area forms a triangular shape. We need to make loops to reduce the number of polygons around the chin. We can easily reduce the polygon flow by redirecting the loops on the axis of the symmetry (refer to step **27** of the following screenshot).



12. To better visualize your polygons, you can hide the ones that you do not want to see. You can use the *H* shortcut. Press *Alt* + *H* to make them reappear.

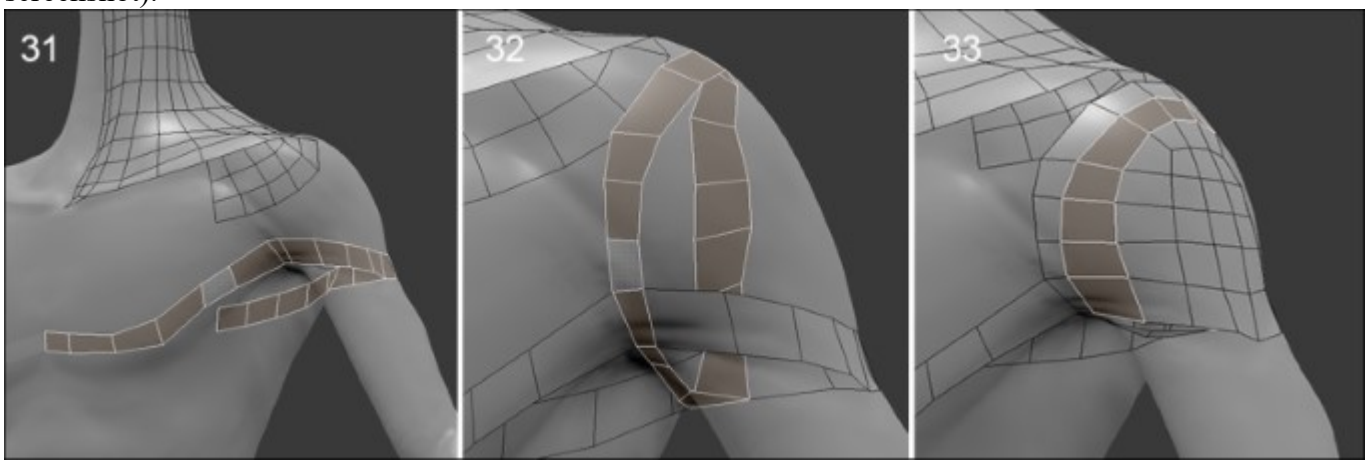
The neck and the torso

We will continue this retopology with the neck.

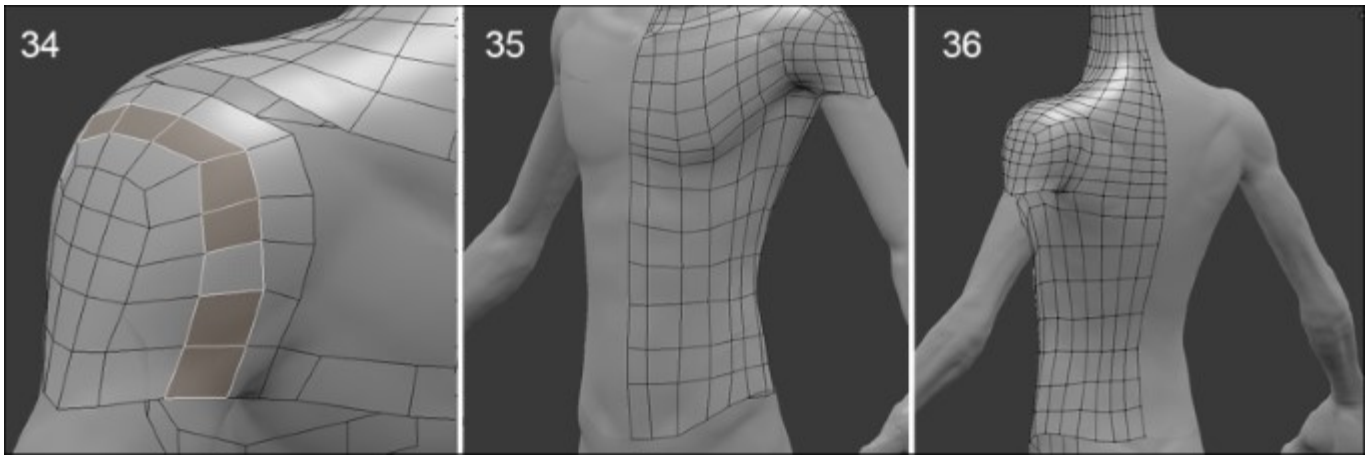


1. We will make a small polygon reduction at the back of the neck in order to lighten the density a little bit. We only need to progressively extrude the polygons along the neck. We will then align the polygons on the collar. We slightly pinch the edges to restore the volume of the collar outline.

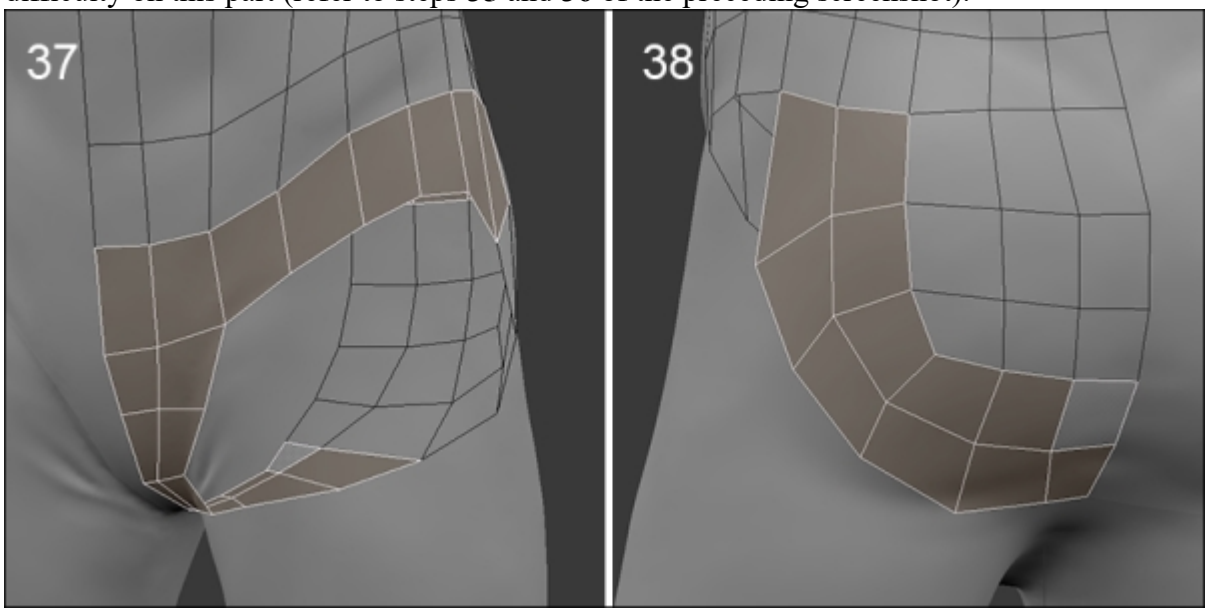
   The challenge is to properly harmonize the polygon flow along the front and the rear.
2. We will continue to extrude to the clavicles. Using loops, we will add a polygon row in the middle of the nape, and we will reduce one on the sternum (refer step **28** of the preceding screenshot).



3. There is an important face loop to place in case we want to animate the arms later. It follows the lower pectorals and shoulders muscles. We will have a 20-face mirrored face loop (refer to step 31 of the preceding screenshot).
4. We will make another important face loop that goes vertically around the shoulder and passes under the arm. We will have a 16-face mirrored face loop (refer to step 32 of the preceding screenshot).



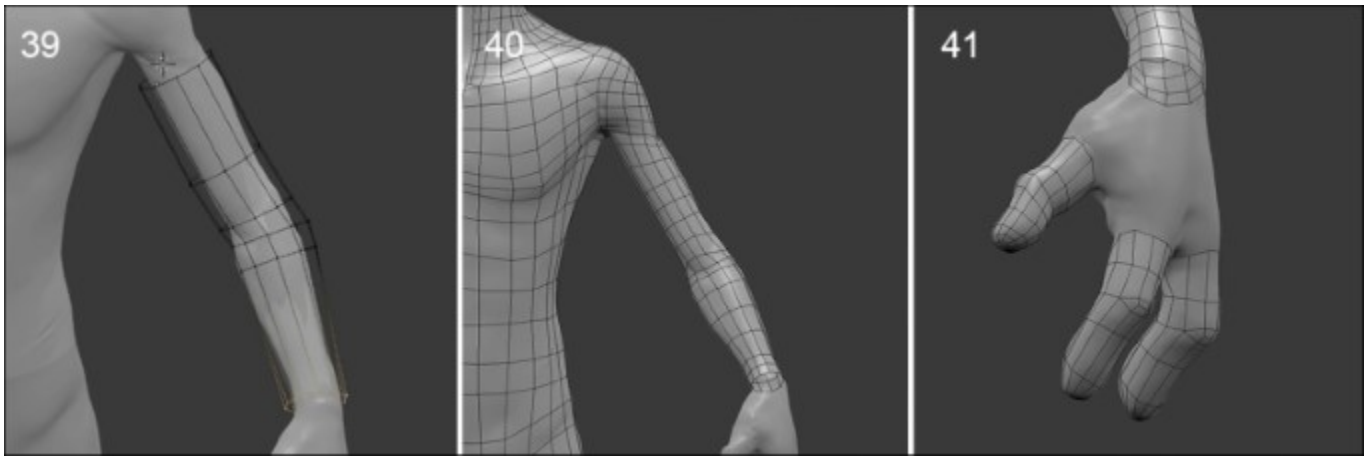
5. Therefore, we will connect the polygons to form the chest and the upper back. On the shoulder, a new face loop links the polygons that intersect vertically and horizontally (refer to step **33** of the preceding screenshot).
6. We will extend the topology straight to the pelvis in a cylindrical manner. There is no particular difficulty on this part (refer to steps **35** and **36** of the preceding screenshot).



7. We end the hip with a strip of polygon that joins the axis of symmetry and we will create the crotch by connecting the buttocks polygons. Don't add too many edge loops, just what is needed. We will have a 6-face strip to make the connection (refer to step **37** of the preceding screenshot).
8. Now, we will make a face loop that follows the shape of the buttocks. This face loop allows us to have a good deformation when animating the legs. To connect the vertical polygons that come from the back, we will make a second face loop right in this area (refer to step **38** of the preceding screenshot).

The arms and the hands

It is time to make the topology of the arms. Considering they are quite thin and not very muscular, we are going to use the same technique that we used for the antennas.

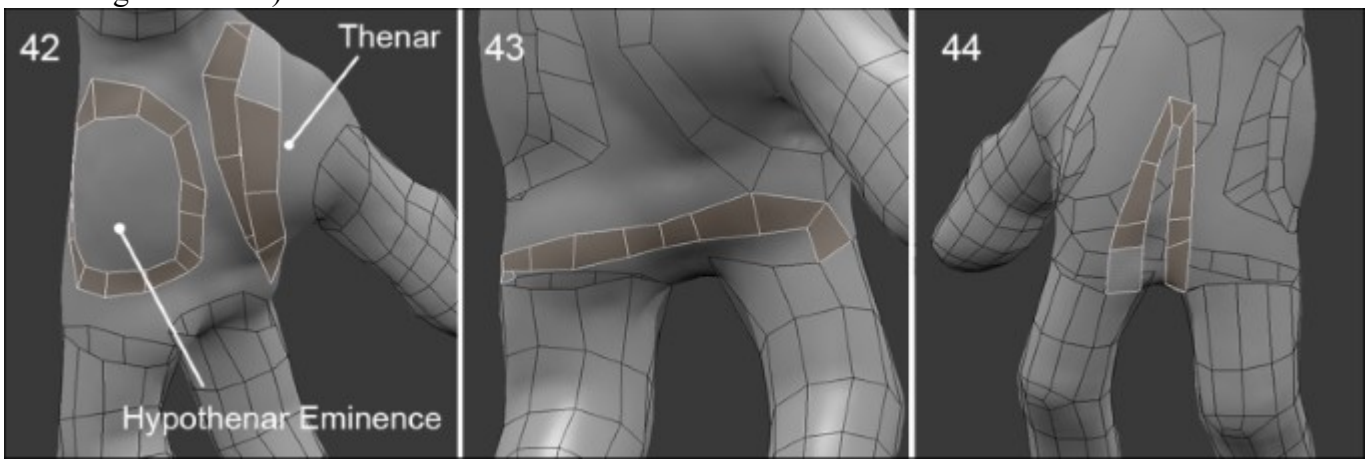


1. This time we will add a 10-face cylinder (*Shift* + *A*) positioned around the arm. Be careful to match the number of faces on the shoulder (refer to step **39** of the preceding screenshot).
2. To facilitate the visualization of two meshes, we will parent our semitransparent material by first selecting our retopology, and then selecting our new cylinder by pressing *Ctrl* + *L* and selecting **Materials**.
3. We will add a **Shrinkwrap** modifier with the sculpted mesh as a target.
4. We need enough polygons to get a sufficiently smooth shape. We will bring, little more loops on the elbow in case we need to animate the arm later so that it will bend appropriately.
5. As for the antennas, we will apply the **Shrinkwrap** modifier in order to freeze the new shape. We will then join the arm to the rest of the body (*Ctrl* + *J*).
6. Now that we have only one mesh, let's connect the arm to the shoulder by selecting the two opening loops and by bridging them together (by pressing *W* and selecting **Bridge Edge Loop**). Since we have the same number of vertices on the two sides, Bridge must work fine. We will then need to adjust the topology by adding some edge loops here and there. Don't forget to snap back the vertices on the sculpture (refer to step **40** in the preceding screenshot).

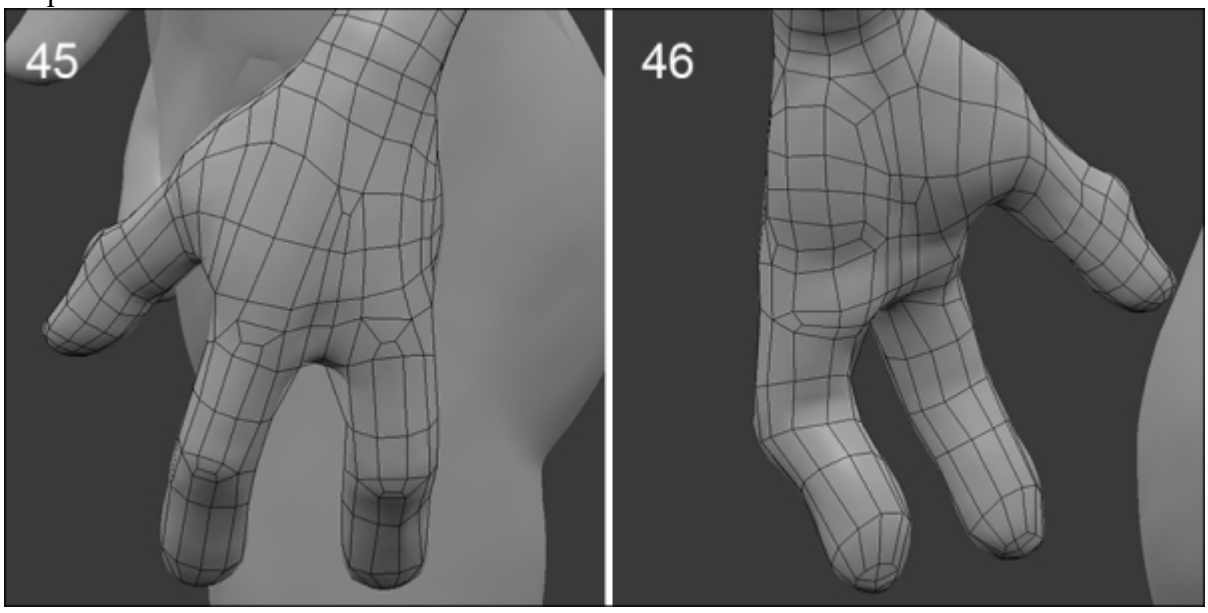
Let's start the hand. It is a particularly delicate part of the anatomy to do:

1. We will start by doing the fingers using an extruded circle again and the **Shrinkwrap** modifier for each finger. This is the same technique used for the antennas and the arm. This time, we need to extrude a 10-edge circle for each finger. This number is important because we need to have enough polygons for the hand. Pay attention to the orientation of the thumb (refer to step 41 in the preceding screenshot).
2. Once this is done, each finger is closed by joining six four-sided polygons. Our character has only two phalanges by the finger, so be sure to place three cuts around each of them if you want to properly animate the fingers later.

3. Then, we will make some of the important face loops of the palm. There is a 10-face face loop that passes around the thumb (the thenar muscles). There is a 14-face face-loop on the opposite side of the hand near the little finger (the hypothenar eminence) (refer to step 42 in the following screenshot).



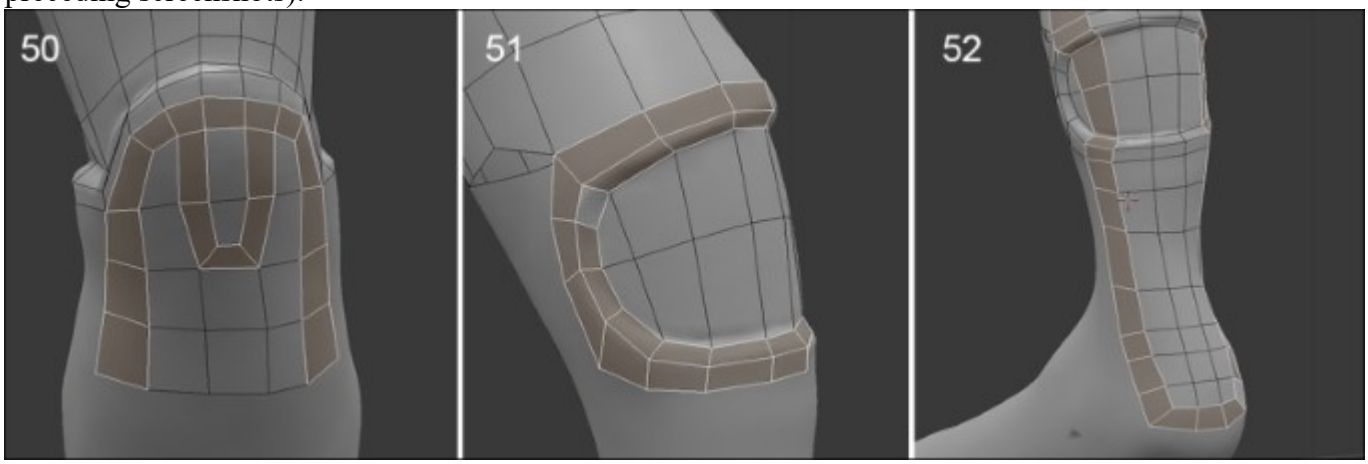
4. We will create a face strip in order to accentuate the basis of the fingers. It is also important to leave a space between each finger (refer to step **43** in the preceding screenshot).
5. Now, all the face loops have to be connected. We have the needed density to only use four-sided polygons without too much difficulty. Train yourself to fill holes with the number of polygons that you have. In this case, we can't change the number of polygons that wraps around each finger (ten in our case) (refer to step **44** in the preceding screenshot).
6. We will add a few more horizontal edge loops around the phalanges in order to have a smooth shape.



The legs

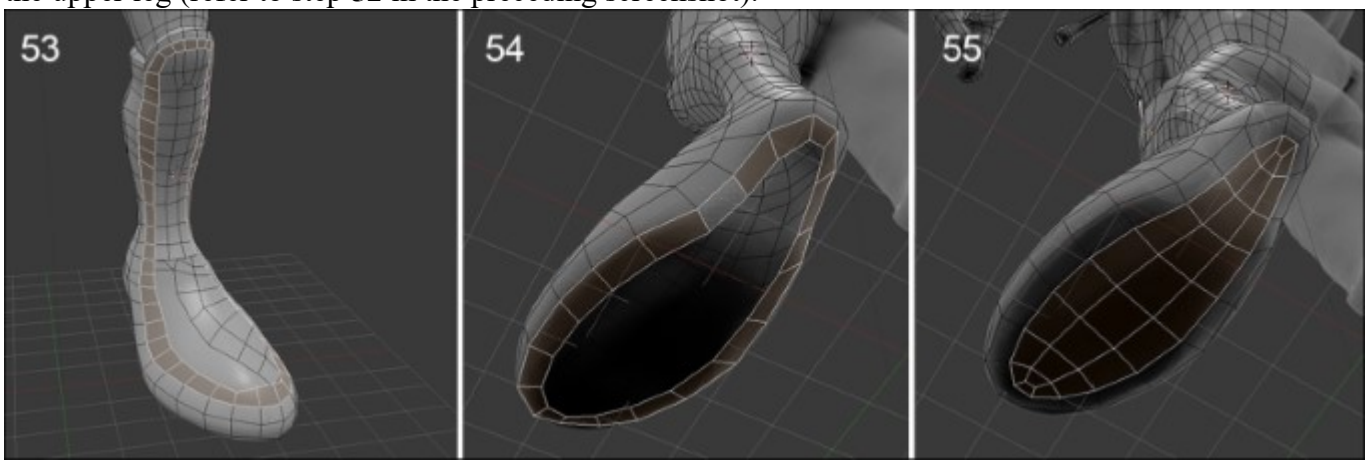
We are going to finish this retopology with the legs. Let's begin with the thigh:

1. We will select the edge that makes the outline of the leg and extrude it to the knee (*E*). As before, we will add a few edge loops (*Ctrl* + *R*), and then adjust and snap the topology on the surface of the sculpted mesh. We could have used the technique used for the antennas and arms with a cylinder but the thigh is wide enough and less complex (refer to steps **47** and **48** in the preceding screenshots).



2. We will add an edge loop a little tighter near the knee.
3. We can see that the boots are just above the knees, so we will make two edge loops that stick to the relief (refer to step **49** in the preceding screenshot).
4. On the front of the knee, there is a loop that will reduce the number of polygons on the shin. It goes up and down along the thigh (refer to step **50** in the preceding screenshot).
5. There is another face loop to be created that will follow the rounded shape of the boot at the knee. It goes up toward the thigh (refer to step **51** in the preceding screenshot).

6. We must create the missing polygons on the top of the boot in line with what was done until the opening of the calf.
7. This opening of the boot requires two face loops that allow us to maintain the hole. We must pay attention that it is easily connectable between the top and bottom. This looks like the circular topology case that we had analyzed before. We have two 18-face mirrored face loops.
8. We will continue the retopology by creating a loop that goes under the heel and connects with the upper leg (refer to step **52** in the preceding screenshot).



9. Let's go back to the front of the thigh and continue to form a strip of polygon that goes around the foot and forms a long face loop. It goes through the knee. This allows us to control the polygon flow by adapting it to the shape. We just need to keep the same number of polygons on both sides of that face loop in order to connect them properly (refer to step **53** in the preceding screenshot).
10. A final important loop remains to be made. This is located under the foot. It follows the outline of the plantar arch with a 20-face face loop aligned so that we can easily connect it to the rest of the foot (refer to step **54** of the preceding screenshot).
11. We still have to connect the inside of this underfoot face loop. To move from one row to three rows of polygons along the length of the foot, we must create a face loop by cutting the faces at the ends of the feet (refer to step **55** of the preceding screenshot).

The retopology of this little character is now over. We complete a mesh that offers many possibilities, such as creating textures or doing animations. All of this couldn't be envisaged with a very high polygon density and sculpted 3D model. Having a few polygons will facilitate the process. The important thing now is to get the small details that we had sculpted back on our mesh that we can describe as a **low poly mesh**. For this, we must discover a process called UV's unwrapping.

A presentation of each important face-loops of the alien

Unwrapping UVs

Now that we have a clean topology, we can learn more about the UV unwrapping process that we introduced in the first chapter. We have to do this in order to project the sculpted details on our clean mesh. Before starting, let's see what UVs are.

Understanding UVs

The goal of the UV unwrapping process is to flatten the 3D mesh in a 2D space in order to project textures (2D images) on it. To understand the process better, imagine that we are going to remove the skin of our alien in order to flatten it (I know that the metaphor is a little gory, but stay with us, there won't be any blood). If we have to do this, we would need to detach the skin by cutting along the imaginary seams and then flattening it down.

Another way to better understand UV unwrapping is to think about how clothes are made and take the steps in the reverse order. For instance, at the beginning, a shirt is a flat piece of tissue where all the seams are marked down. Then, the different cut pieces are attached together in order to form the volume of the shirt, like the sleeves. So, if we were unfolding a cloth along its seams, we will tell Blender where the seams are placed on our 3D mesh. After this, the ones that have been marked and the model that has been flattened down will correspond to each vertex, edge, or face between the 3D model and the flattened version of it; this is actually a 2D representation of the geometry.

In general, we call the coordinate axes of a 2D space the *x* and *y* axes, but in this case they are named as U and V. Hence, the name **UV unwrapping process**. Another really important thing to note is that we often want to avoid any overlap of geometry in the UV space. This is because when we use UVs of an object in order to project 2D textures on it, we seldom want to have the same texture information twice on the 3D model. We will also need to optimize the UVs so that they don't generate strange distortions.

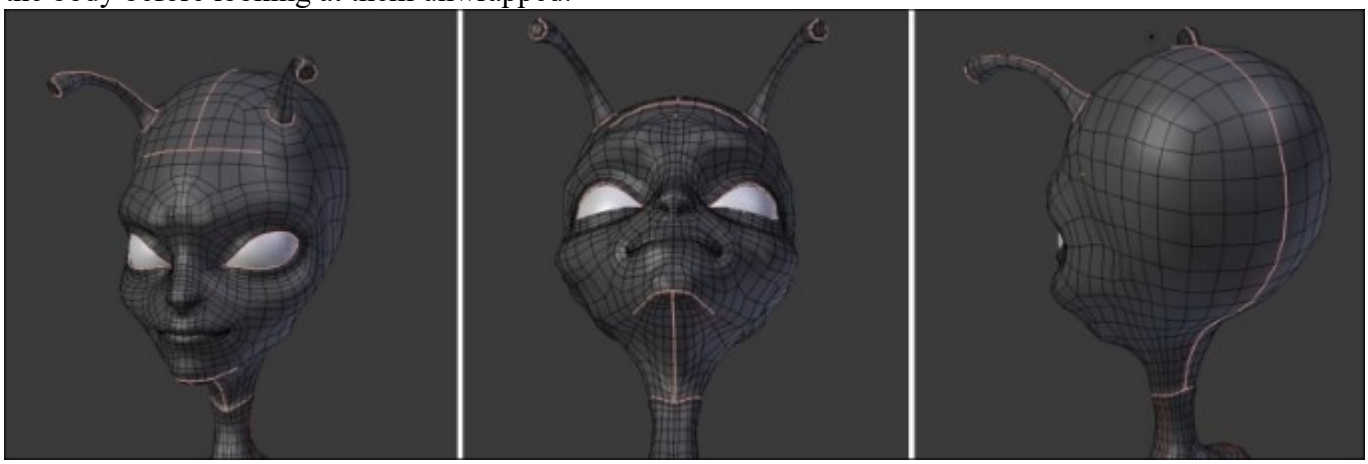
Lastly, similar to the shirt example, our UVs will be usually separated in different parts called islands. For instance, we will disconnect the head of our alien from its hands or other limbs. These islands need to have a proportional scale to the geometry data they represent (that is, the head is bigger than the hands, so it needs to be bigger in the UV space).

The placement of the seams

It's now time to unwrap the UVs of our alien character. So, as we stated before, we first need to tell Blender where the seams are going to be:

1. We will first select all the edges that start from the lower back of the neck to the middle of the forehead (the ones that are on the symmetry axis). We can also select four perpendicular connected edges together at the end of the previous group of edges selected. It will look like an inverted T shape from the front view.
2. Now, in order to mark our selection as seams, we will go to the **Edges** menu (*Ctrl* + *E*) and select the **Mark seam** option.
3. Always try to think as if you were using a cutter, so the mesh will be disconnected in the UV space along the marked edges. Also, note that the seams are in red.
4. Now, we will select the top collar edge loop (press *Alt* and the RMB) where it meets the previously marked edge and mark it as a new seam again. This will completely separate the head and neck portion into a new island.
5. In order to have less deformation under the chin, we will select edges starting from the top of the collar (where we placed the seam) to the chin along the axis of symmetry. We will add five perpendicular edges to our selections and mark our selection as new seams. We've done this in the same way as the forehead.

6. In order to separate the antenna in its own island, we will select the circle loop of its base and mark it as a seam. We will also have to mark a vertical line of edges that start from the previously marked seam and end at the top of the antenna.
7. Then, we will end this seam by following the circular cap of the top of the antenna without closing it completely, leaving an edge unmarked. When you are marking the seams, try to think of an approximation of the shape that you know in the real life. For instance, in this case, the antenna looks like a candy wrapper.
8. To end with the head, we will select an edge loop around the eye and mark it as a seam. If you want to remove a seam, you can simply select a seam, and in the **Edges** menu (*Ctrl* + *E*), you can select the **Clear Seam** option. We've now completed the UVs of the head, but we will finish the body before looking at them unwrapped.



The head seams

9. Now, we will select a vertical edge loop around the shoulder and mark it as a seam. This is going to represent the separation between the arm UVs and the torso UVs.
10. In order to clearly see that the arm UVs are disconnected from the rest of the body, you can go in the **Face** mode, hover your mouse on the arm and press the *L* key (selection of the linked parts).
11. We will then separate the torso UVs from the legs by marking an edge loop situated under the belt.
12. It's best to consider the torso UVs as two separate islands, the front and the back. So, we will split this by marking the edges that connect the shoulder's vertical seam to the seam of the collar. These edges follow the angle created by the neck and the shoulder.
13. After this, we will mark a line of edges that start under the arm and go right to the belt seam. As you can see, using the linked parts method presented before, the torso is now in two parts that are delimited by the collar, belt, and shoulder seams (of course, we always have the mirror modifier turned on, so we can only see half of the torso's linked parts)
14. We will then mark an edge loop that follows the top boot outline.
15. As we did with the torso, we will split the legs UVs into two islands. To do this, we simply mark the edges that start from the belt seam (following the same direction of the seam on the side of the torso) and end at the boot seam.

16. For now, the legs are still in one part, so we will add a new seam starting from the boot seam and ending below the pelvis bone. Now, our legs UVs are split into two islands.
17. It's now time to mark the seams of the boots. To do this, we will to the bottom view (*Ctrl + 7* numpad key) and mark a loop that follows the footprint of the boot. We will also mark the loop that follows the hole of the boot on the calf muscles.
18. We will then mark the vertical edges of the back that connect the footprint seam to the hole seam and the ones that connect the hole seam to the top boot outline.
19. Lastly, we will mark the seams of the hand and the arm. We will mark the wrist edge loop in order to disconnect its UVs from the arm. Now, we will mark a continuous line of edges that start from the tip of the thumb and follow the silhouette of the hand by connecting to the wrist seam. As you can see, we didn't split the top of the thumb so that the interior palm and the back of the hand share the same UV Island. Then, we will mark a continuous line of edges from the wrist to the shoulder in order to do the UVs of the arm.

Congratulations! You've finished marking all the seams, and we are now ready to unwrap the alien. In order to do this, we must first apply our **Mirror** modifier. As you can see, all the seams are now mirrored to the other side; what a time saver!

1. Now it's time to unwrap the alien. Let's see how this is done. We will first select all its geometry by pressing *A*.
2. Now, we will press the *U* key to get the **UV Mapping** menu and select the **Unwrap** option. Unwrap is now done!
3. To see the UVs, we will open a new **UV/Image Editor** by splitting our 3D view into two and changing the new window to the appropriate editor.

The upper body seams

The seams follow the volume of the each part of the character.

The lower body seams

As you can see, in the **Edit Mode**, all our 3D geometry is being flattened down in the UV space. We are now ready to reorganize each island. Note that if you still have problems while understanding the relation between the 3D space and the UV space (don't be disappointed if this is the case, UVs are hard to understand in the beginning), you can use the linked method presented before to see how each part is represented in the UV space.

The placement and adjustment of the islands

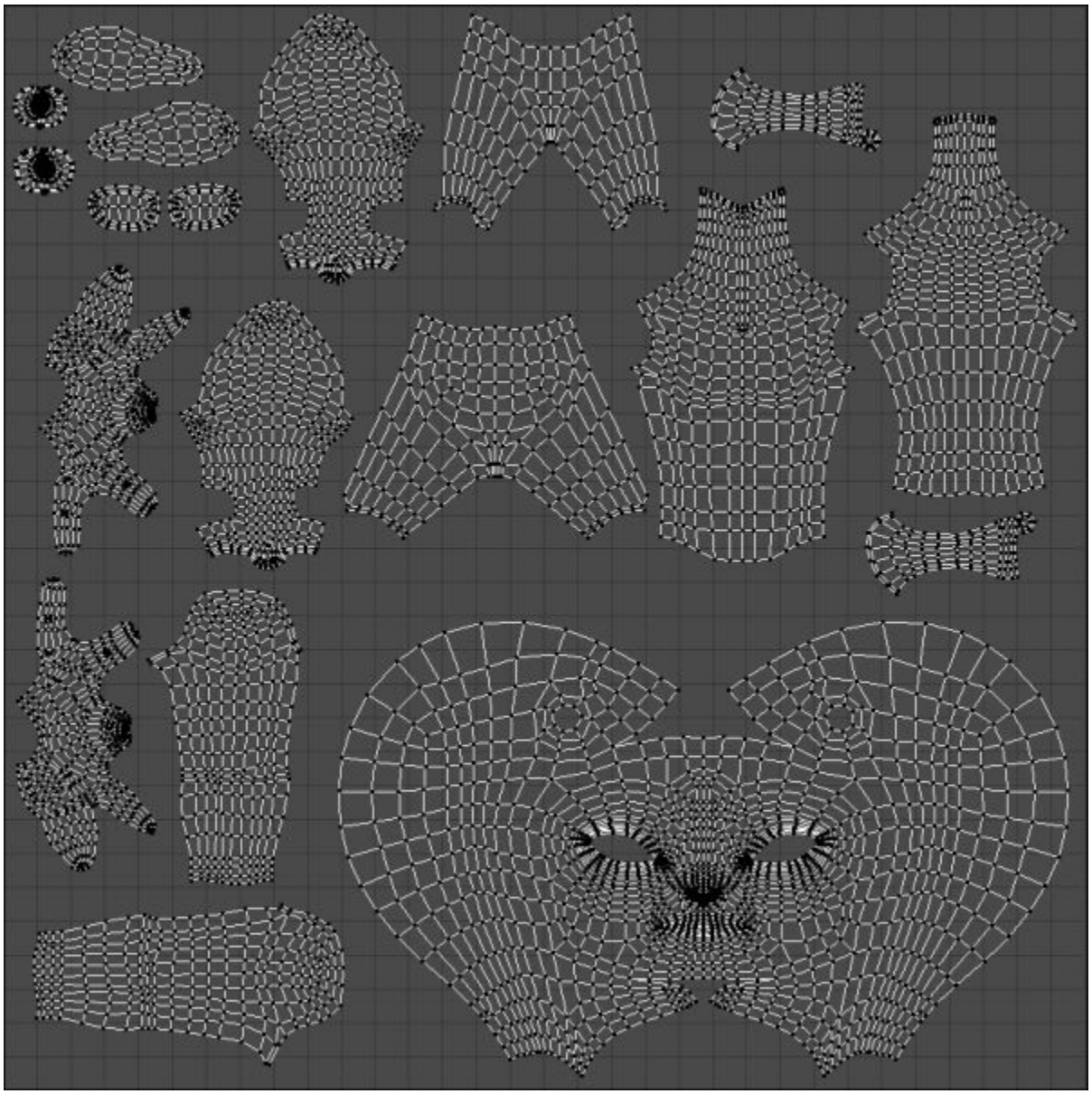
We are now going to adjust each island in the UV space. As you can see, all our islands are bounded to a square. Everything that is on the outside won't be used, so all the islands need to be tightly packed in it. Moreover, we'll have to check whether there are any important deformations on our 3D mesh. If this is the case, it means that each texture will be stretched whether we want it or not.

1. In order to adjust the different island placements, in the **UV/Image Editor**, we will first need to enter the **Island UV selecting** mode.

   This allows us to select every island and place it with the grab (*G*) tool, scale (*S*) or rotate it (*R*) in the **UV/Image Editor**.
2. When we try to place the islands correctly on the UV space, we need to be sure that they don't go out of the square bounds (represented by a grid).
3. You can also use the **Pack Islands** tool located under the **UVs** menu, in order to have this automatically done. But it's always best to do it by hand.
4. Another thing to remember is to have the island proportional in scale to what they represent on the mesh. This was automatically done when you first unwrapped your mesh, but if you scale the islands by hand, be aware of this. However, you can sometimes counter this rule when you need more texture details on a specific location. For instance, the head is one of the most important part of the alien, so we can scale its island a little bit more.
5. Once you've packed all your islands, you can add a checker texture to check whether there are any important stretches with your UVs. We will add a predefined texture by first entering into the **Edit Mode** of the alien.

6. We will click on the **+ New** button in the header of the **UV/Image Editor**, and select a **UV Grid** image under the **Generated Type** drop-down menu. We can then validate it with **OK**.
7. In order to see the test grid on our alien in the viewport, we will need to enter the **Texture shading** mode located under the **Viewport shading** drop-down menu in the 3D viewport header. We can also toggle it on or off by pressing *Alt + Z*.

Now that we see our test grid on our alien, how can we interpret it? If all the squares are still approximate squares, then it means we don't have any important deformation. Their orientations don't matter too much here.



The final UV Island placement

The baking of textures

Now that we have UVs on our little alien, we can take a look at how we can transfer all the details from our sculpture to the new **retopologized** mesh. The details in the sculpture are simply there because of the amount of geometry it is composed of, but in the retopology, we have kept the amount of details to about 5000 polygons, in order to have a manageable mesh.

The baking of a normal map

The best solution that we have in order to transfer the details is to bake a normal map that will act with lights to give the impression of detail.

What is a normal map?

At its lower state, a normal map is an image or a texture that will be projected on the mesh through the UVs. This' is why it's important to UV unwrap the object. Note that if the UVs are stretching at some place, the details that are in the normal map will then be stretched too. As mentioned before, a normal map will do its magic with the lighting. It is composed of red, green, and blue pixels that respectively represent the X, Y, and Z orientations of the normal of a face. So, we will need to create a normal map that will contain all the normal information of the high poly sculpture. The higher the definition of the normal map texture will be, the most precise the details will be.

Making of the bake

In order to create a normal map from our sculpture, we will need to use the bake tools of Blender.

1. We only need to see the sculpture and the retopology in the 3D view. To hide the rest, we will select them both and press *Shift* + *H*.
2. We can now deselect all our objects by pressing *A*.
3. In order to do the bake, we will have to first select the alien sculpture and then the retopology (this becomes the active object).
4. Now, in the Properties editor, under the **Render** section, we will expand the **Bake** subpanel.
5. The first option to choose is what type of map (or texture) we want to bake. So, under the **Bake Mode** drop-down menu, we will select **Normal Map**.
6. The next thing we'll have to check is the **Selected to Active** option that tells Blender to bake from the sculpture to the active object (our retopology).
7. We will then need to add a blank texture to bake our normal map on. So, we enter the **Edit Mode** of the retopology and click on the **+ New** button (or the + icon on the right-hand side of the texture list), and instead of selecting a UV grid, we choose a **Blank** texture with a width and height of 4096 pixels.
8. Before baking our map, we need to click on the **Smooth Shading** button; otherwise, we will see the polygons on our bake.
9. Last but not least, we will come back to the **Bake** panel and click on the **Bake** button. Don't forget to save your map (**Image** | **Save As Image** or simply press *F3*) or it will be lost!

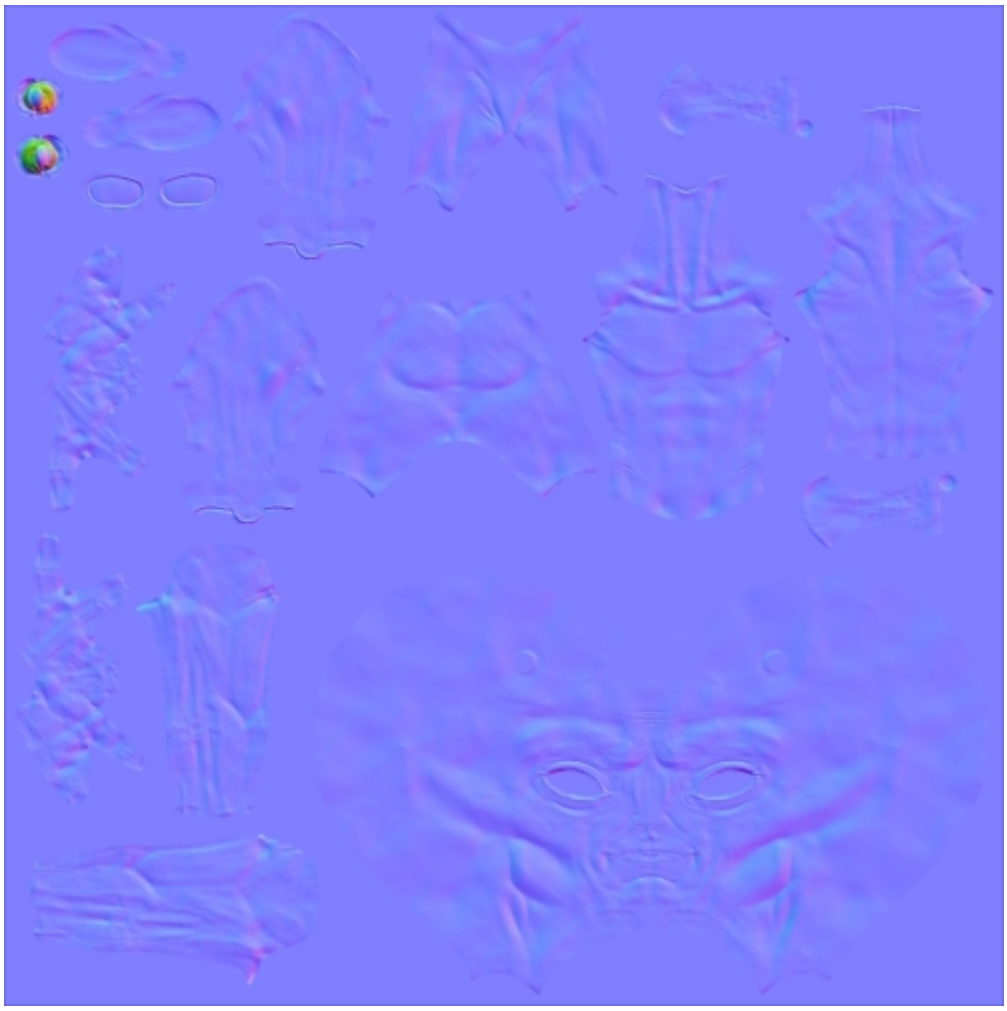
At this point, you should see that the normal map has started to appear. If you get an error message, it may be because you didn't add the texture on the low poly while you were in **Edit Mode**, or that you've selected the retopology before the sculpture. If you want to have your normal map packed into the

`.blend` file, you can go to the **File** menu and select the **Pack All into blend file** in the **External Data** subpanel.

Note

About the size of the textures

Usually, textures aren't rectangle. They are set with the power of two of the width and the height. The common sizes are 256 x 256, 1024 x 1024, 2048 x 2048, and 4096 x 4096.



The baked normal map of our alien

Displaying the normal map in the viewport

Now that we have a nice normal map baked, we will show to you how to display it in the viewport:

1. We first need to be in the **GLSL** mode (press *N* and select **Shading Subpanel** | **Material mode** dropdown).

2. We also need to add a new material on our low poly. To do this, we can click on the **New** material button under the **Material** tab of the Properties editor.
3. We will then go to the **Texture** tab of the Properties editor and click on the **New texture** button.
4. Under the **Image** subpanel, we can choose our normal map with the left-hand side drop-down menu.
5. Then, we will check the **Normal Map** check box under the **Image Sampling** subpanel.
6. For now, Blender will only interpret the map as a diffuse map. We want to tell Blender to use the normal information of the map. So, under the **Influence** subpanel, we uncheck **Color slider** and check **Normal slider**.
7. Now, we will need to add a light in the 3D viewport (press *Shift* + *A* and navigate to **Lamp** | **Hemi**) and orient it correctly.
8. Lastly, we need to enter the **Texture shading** mode. We can now appreciate the comeback of our sculpture details. If you want, you can move the light around to feel the relief of the normal map.

The baking of an ambient occlusion

Now that we have the normal map set, we will improve our alien with another map called an ambient occlusion.

Understanding the ambient occlusion map

An ambient occlusion is a black and white texture that represents the contact shadows of a mesh. The contact shadows are the shadows produced by the small proximity of objects. In order to have a nice ambient occlusion, we need to increase a sampling parameter that corresponds to the "noisiness" of the shadows. The more samples you have, the smoother the shadows will be. This map will then be multiplied on top of our diffuse material color.

Note

Multiplying colors

In computer graphics, black is represented by a value of `0` and white by a value of `1`. So, when we multiply a color with black, its result is `0`, and when we multiply a color with white, its result is the color. For instance, we will take two colors, J and K, a pure blue that is represented by `R(J): 0`, `G(J): 0`, and `B(J):1` (**RGB** means **Red Green Blue**), and white, `R(K): 1`, `G(K):1`, and `B(K):1`. When we multiply both, we will have *R(J)*R(K) = 0 * 1 = 0*, *G(J)*G(K) = 0 * 1 = 0*, and *B(J)*B(K) = 1 * 1 = 1*, so the resultant color is `R:0`, `G: 0`, and `B:1`. It is the original blue.

Creation of the bake

We will now follow the same principle as the normal map, but we will change the sampling value:

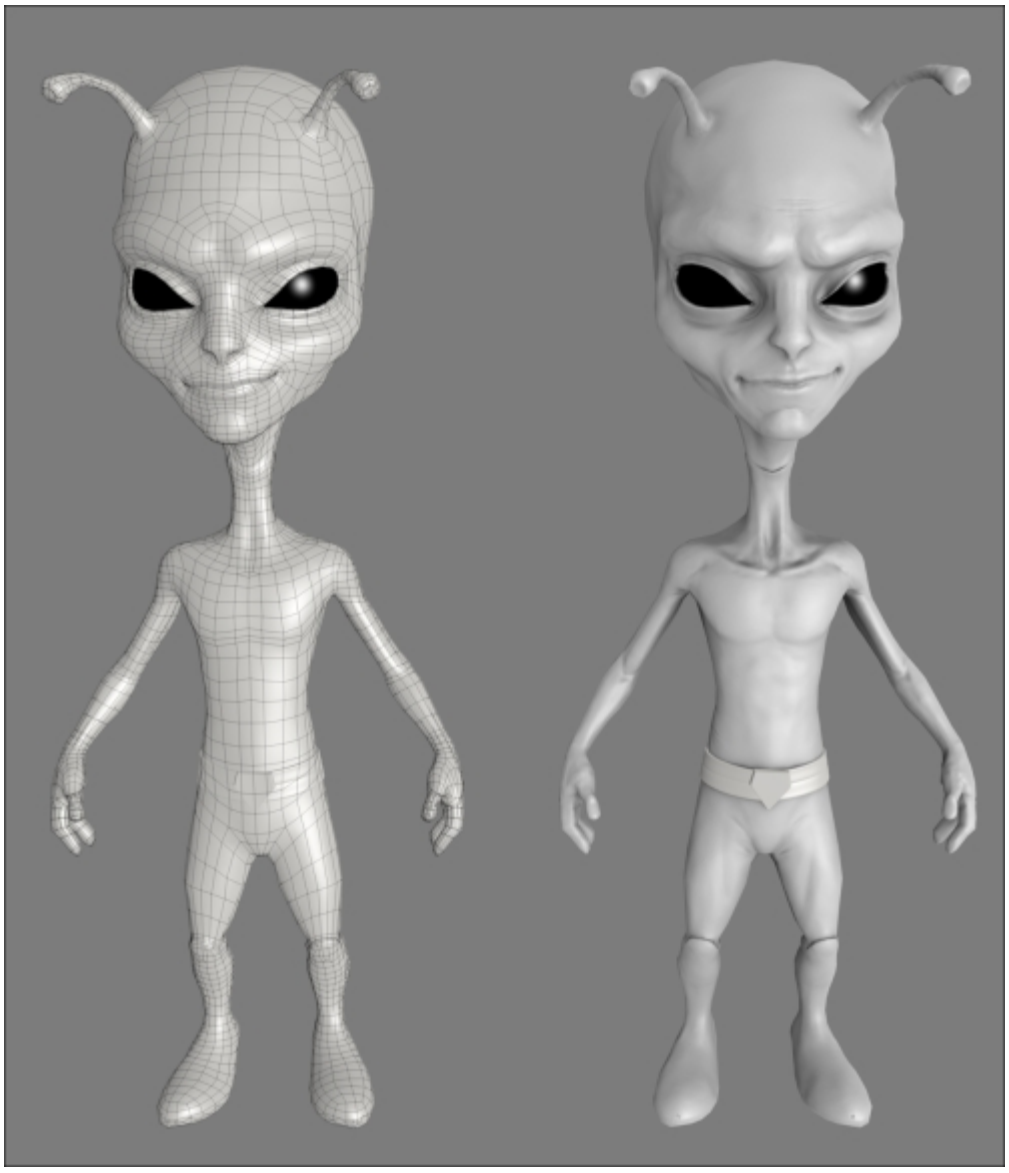
1. The sampling slider is situated under the world tab of the Properties editor in the **Gather** subpanel. Even if it's grayed out, it will work for the bake. We will set it to **10** in our case. Don't go too high with this as it will increase your baking time.
2. Now, refer to the normal map baking process, but instead of choosing a normal map in step 5, choose an ambient occlusion. Again, don't forget to add the blank texture in **Edit Mode**, and save it after the bake. You can also name your textures in the **UV/Image Editor** in the header with the corresponding text field.

The baked ambient occlusion map of our alien

Displaying the ambient occlusion in the viewport

In order to see the ambient occlusion applied to our mesh, we will have to add a new texture to our material. This is done as follows:

1. First, we will select our alien, and then we will select the material that has the normal map on it in the **Material** tab of the Properties editor.
2. We will, then, go to the **Texture** tab of the Properties editor and add a new texture below the normal map. In order to do this, we select the second slot and click on the big **New** button.
3. We can now choose our ambient occlusion under the **Image** subpanel.
4. Under the **Influence** subpanel, we turn the color slider on but we change the **Blend Mode** from **mix** to **multiply**, as we had previously explained. As you can see, this perfectly works in the viewport when the **Texture** shading mode is turned on. We can clearly see the contact shadows of our mesh around his eyes, for instance.



The alien with a proper topology (shown on the left-hand side) and with its normal map and ambient occlusion (on the right-hand side)

Summary

In this chapter, we saw how to create a proper retopology based on the sculpture made in the previous chapter and retrieve its details with a normal and ambient occlusion map. There are other maps that you may want to create, such as a diffuse, displace, or lighting map. Now let's go to another project, the Haunted House!

Chapter 5. Haunted House – Modeling of the Scene

Welcome to the scary project!

In this chapter, we will model a haunted house that we will texture and render in the future chapters. You will use the modeling techniques that we have already seen in the previous chapters and learn some new techniques using some useful modifiers and time-saving tools. Moreover, you will learn how to correctly organize your scenes by grouping objects and placing elements in layers. Now that you have more experience with Blender, we aren't going to show you all the steps in detail but rather describe the key points of the process. If you have any difficulties, you can always go back to Chapter 2, *Robot Toy – Modeling of an Object*, and Chapter 4, *Alien Character – Creating a Proper Topology and Transferring the Sculpt Details*, in order to review some of the modeling techniques. Let's start our scene! In this chapter, the following topics will be covered:

- Modeling on scale
- Blocking the house
- Advance modeling tools
- Modeling with curves
- Organizing the scene

The final haunted house should look like the following screenshot:

Blocking the house

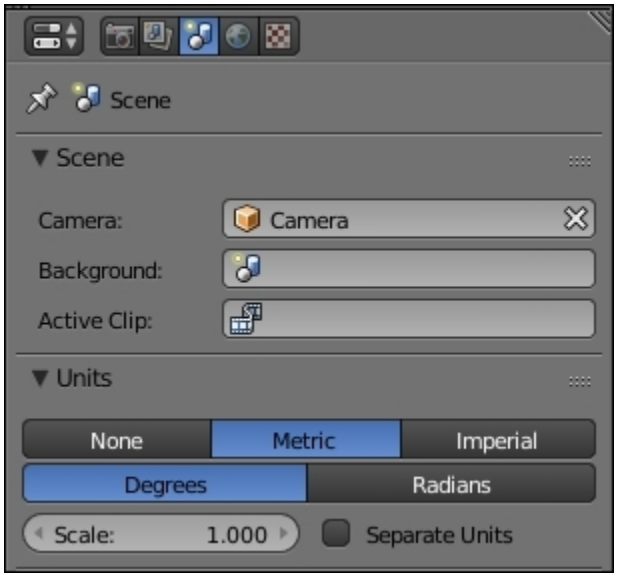
Before going into detail, we will start by testing different shapes in order to create the concept of our house. It is like a 3D sketch.

Working with a scale

In order to create our haunted house and its environment, we need to work with a real world scale. Indeed, when you are working on objects, such as buildings, where the scale matters, it is important to remember to adjust the units of measurement of Blender.

Blender uses, basically, its own unit of measure: the **Blender units** that correspond to a fictitious unit of measure. You aren't going to encounter Blender units in the real world.

There are two other unit systems of measurement in Blender that you can use: the metric system and the imperial system. We prefer the **metric system**. For this, go to the Properties panel on the right-hand side of the user interface under **Outliner** (in the default layout). In the **Scene** tab, you will find the **Units** tab. Choose **Metric** and **Degrees**.



The metric system allows us to work in kilometers (km), meters (m), centimeters (cm), millimeters (mm), and micrometers (µm). Let's choose meters in our case. For this, we set the **Scale** value to **1.000**. A value of **0.1** would make us work in centimeters.

To know the size of your 3D models, in the **Object Mode**, you can look at their size in the **Transform** tab on the right-hand side panel of the 3D viewport (*N*). This information is also given in the **Dimensions** section for the *x*, *y*, and *z* axes.

You can then display the size of the selected edges. In the **Edit Mode**, under the **Transform** tab, go to the **Mesh Display** tab, and check the **Edge Info** | **Length option**. If you want to measure something, Blender gives you a ruler under the **Grease Pencil** tab in the left 3D view panel (*T*). To use this, simply click on the **Rule/Protractor** button and drag it in the 3D view.

Be careful to always apply your scale and move or rotate the transformation of your objects when you manipulate them in the **Object Mode**. To do this, we open the **Apply** menu (*Ctrl* + *A*) and select **Rotation and scale**. It is important to avoid involuntary deformations after this.

Blocking the bases of the house

To make this house, we don't start from concept art but from an idea and a few references found on the internet.

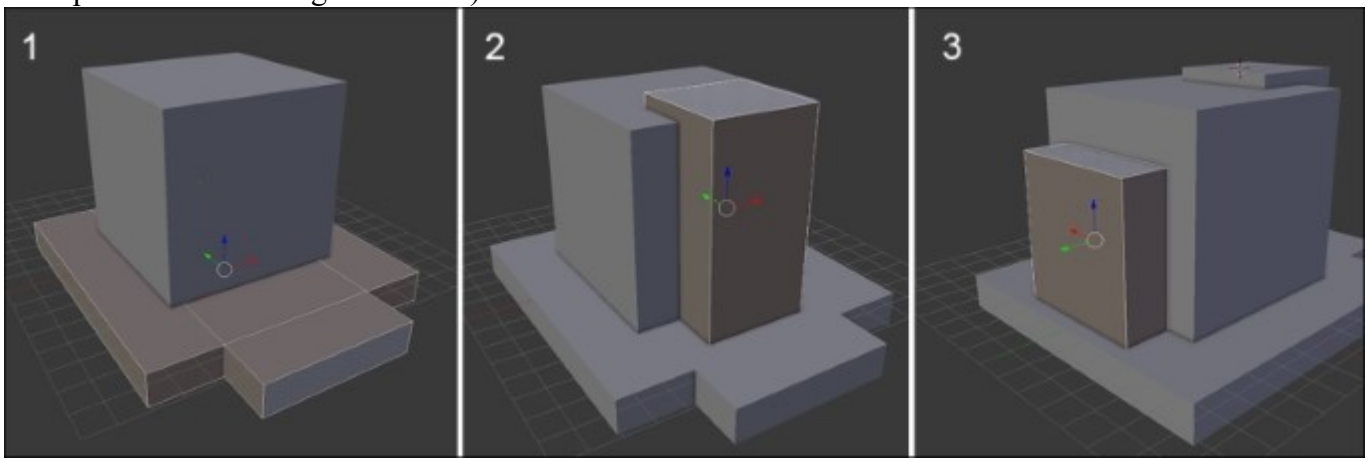
It is very important during any creation to spend a little time documenting to confront the different possibilities of shapes and styles. We need to see what has been done previously and be informed enough to be precise in our work.

As we are not completely sure of the form as a whole, we will adopt a method that involves testing and quickly developing ideas with simple forms. This method is called **Blocking**. This is done as follows:

1. We will begin the modeling by adding a cube at the center of a new scene (*Shift* + *A* and select **Mesh** | **Cube**), which will represent the central part of the house.
2. In the **Object Mode**, we will adjust the size in the **Transform** tab to have something realistic. It is an imposing house, so we will set 7 m on the *z* axis, 7 m on the x axis, and 8.5 m on the *y* axis.
3. In the **Object Mode**, let's duplicate our cube (*Shift* + *D*) in order to make the terrace. So we will scale it to a height of 1.26 m, then we will place it at the base of our haunted house under the main block previously created.
4. The terrace is not completely cubic. We will add two edge loops and an extrusion to the front, which is less wide (on the *x* axis) than the main block.

   It is necessary that this terrace is large enough to be credible, so we will create a passage of at least two meters wide. It is not necessary to be very accurate for the moment, but be aware of your measures, and remember to apply the transformations when you switch back to the **Object Mode** (refer to step **1** in the following screenshot).
5. To improve the general shape of our house a little, we will add a new cube that fits in the front of the central part of the house, centered on the *x* axis.

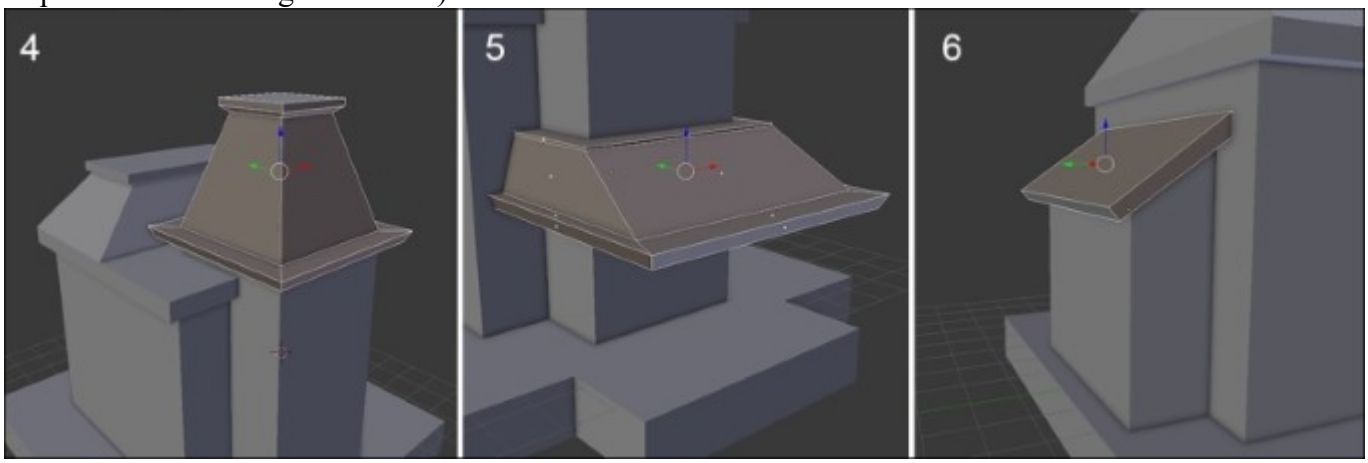
   The height of this cube exceeds the height of the other cube by one third. The rest is hidden in the central block. It has a square base, and it is higher than the main block by about 45 cm (refer to step **2** of the following screenshot).



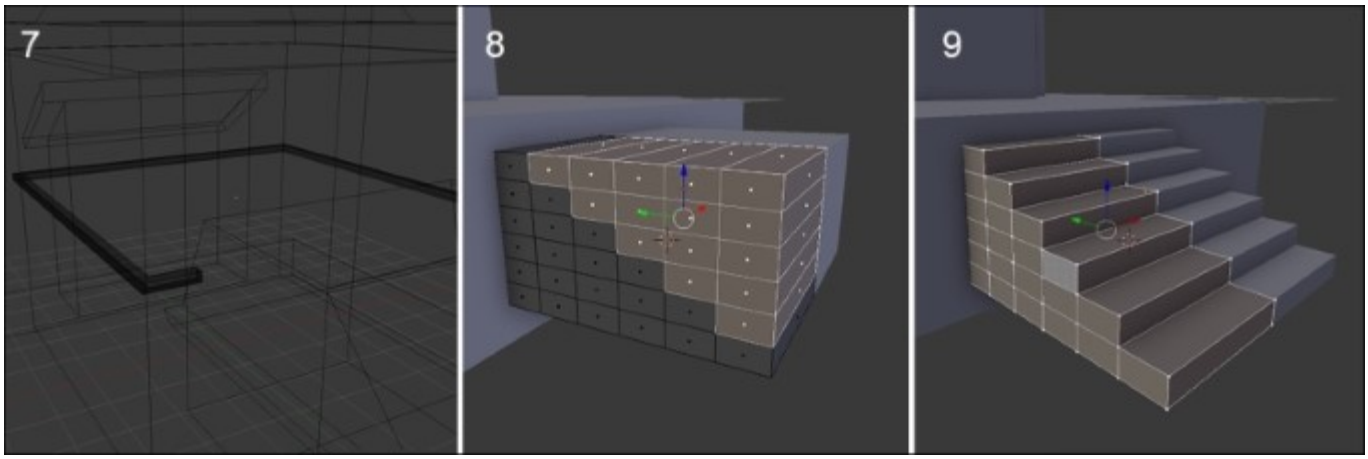
6. Likewise, on the rear part, we will duplicate our front block (*Shift* + *D*) and move it to the other side on the *y* axis. This block is lower than the main block. Its size is 4.7 m on the x axis, 1.67 m on the *y* axis, and 5.3 m on the *z* axis.

Now we have our basic volumes. We can now make the roof that is composed of several parts; one for each block. This is done as follows:

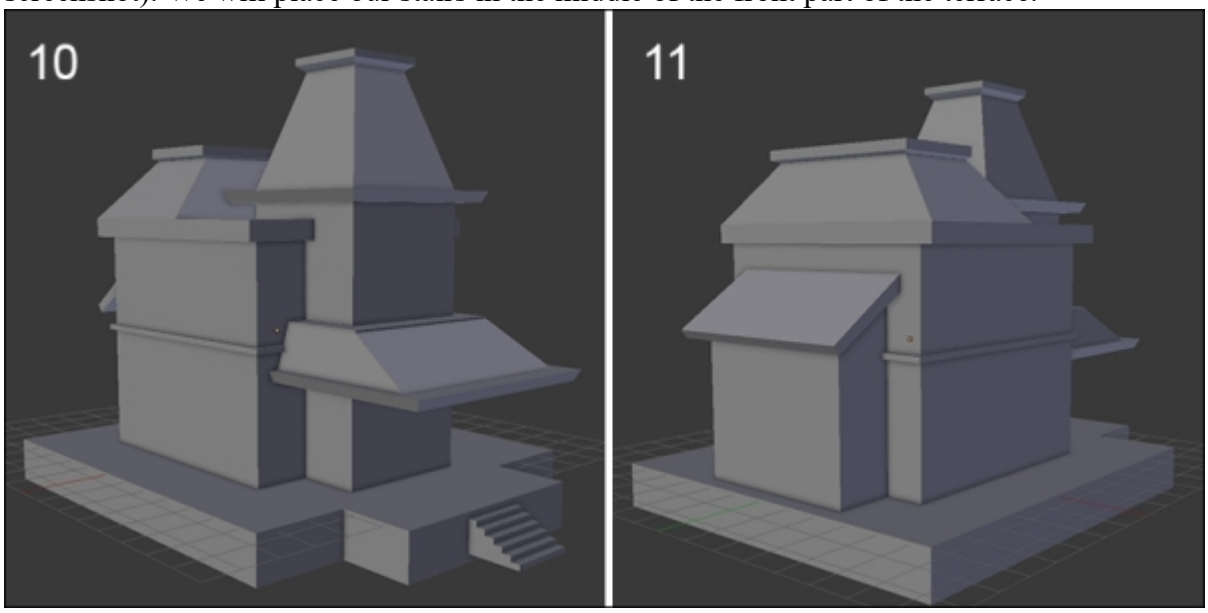
1. We will begin with the roof of the front block by adding a new cube (press *Shift* + *A*, and select **Mesh** | **Cube**). We need to adjust its size to be larger on the *x* and *y* axis.
2. We will move down the top face to flatten it. It is the bottom part of this section of the roof.
3. Then we will do an inset (*I*) and an extrude (*E*) to the top. We will adjust the scale of the top, and then we will add two extrusions (*E*) to make the top thicker and finish the shape (refer to step **4** of the following screenshot).



4. We will duplicate this part of the roof (*Shift* + *D*), and we will place it on the rear half of the central block. We will scale it on the *x* axis (press *S* + *X*) in order to have the same width as the central block. This is also lower than the roof of the front part, so we will also scale it on the *z* axis (*S* + *Z*) (refer to step **4** of the following screenshot).
5. In the same way, we will make another roof that covers the front portion of the terrace. It will be supported by pillars. We will again duplicate our roof that is cut in half, and we will adjust it according to the dimensions of the front of the terrace. We will remove the top part to form a small balcony (refer to step **5** of the preceding screenshot).
6. For the roof of the rear block, we will slightly change the style with a simple tilted platform. We will change the rear block to bevel it. We will duplicate the top face (*Shift* + *D*) and make a new object with it (press *P* and select **Selection**). We will need to make an extrusion on the *z* axis to add a thickness, then we will adjust the size and the position of the wireframe in the **Shading Mode** (refer to step **6** in the preceding screenshot).



7. We will now mark a boundary of two floors with a concrete ledge. For this, we will need to add a new cube (press *Shift* + *A* and select **Mesh** | **Cube**) scaled on the *y* and *x* axes (*S* + *Shift* + *Z*) to be around the main block. We will give it a height of 15 cm and make it exceed the block by about 20 cm (refer to step **7** in the preceding screenshot). We will do an inset (*I*) on the top and bottom faces, then we will delete the nonvisible faces.
8. Let's form the stairs. We will add another cube, then we will resize it to be 84 cm on the *z* axis and 1.5 m on the *y* axis. We will need to divide it into six equal parts horizontally and vertically. In order to gain time, we will add a **Mirror modifier** (refer to step **8** in the preceding screenshot). We will remove the unwanted faces, and then extrude the contour of the top towards the symmetry axis to create the missing faces (refer to step **9** in the preceding screenshot). We will place our stairs in the middle of the front part of the terrace.



These few simple 3D models done in a short time gives us an idea of what our haunted house will look like with further modeling.

Refining the blocking

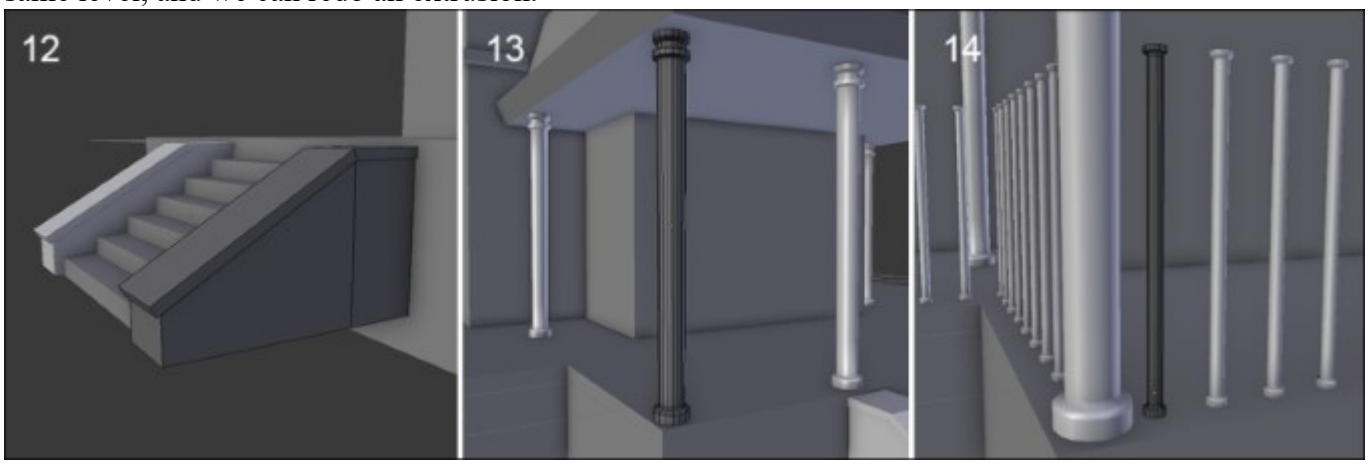
Now that we have the foundations of our model, we will go into the details by adding more defined objects.

Adding instantiated objects

If we analyze the majority of the houses, we can see that they are mainly composed of repetitive shapes such as windows and doors.

So we will use the techniques that allow us to duplicate objects by instance. This means that if we change the geometry of the source object, all the duplicates will change too. As you may have understood, this is really useful in order to save time: for instance, with UVs. Now, perform the following set of steps:

1. Let's start with the low wall around the steps. We will add a cube that we will orient with the slope of the stairs.
2. We will add an edge loop in order to break the slope. Then we will add a thinner piece that recovers the slope. To do this, we will extrude the top faces and scale them appropriately on the same level, and we can redo an extrusion.



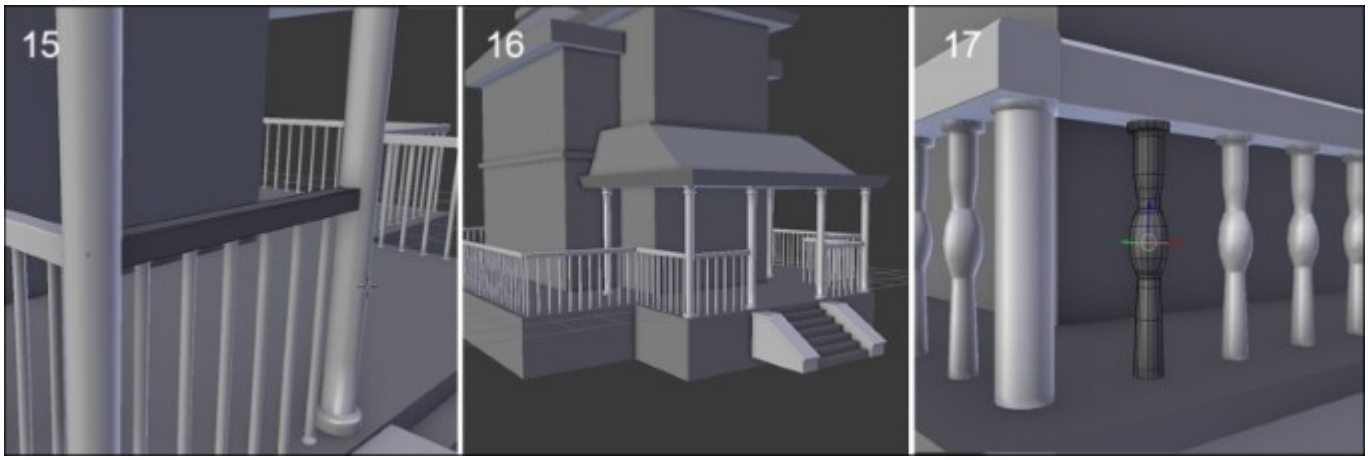
3. In order to mirror the other side of the low wall, we will center the pivot point of the stair object at its center (*Ctrl* + *Alt* + *Shift* + *C* and select **Origin to Geometry**). Now we can safely add a mirror modifier with the stairs as the mirror object (refer to step **12** of the preceding screenshot).
4. Next, we will do the columns that will support the roof that covers the front of the terrace. We will need a 16-face cylinder that is 18 cm wide with a height of 2.8 m. In the last tool options in the left 3D view panel, we will choose the **Nothing** option under **Cap Fill Type**. We will then position it on the left-hand side of the stairs, and we will duplicate it as instances with the **Duplicate Linked** (*Alt* + *D*) function.
5. In the **Object Mode**, we will place our columns at the four corners of the terrace roof. In order to add some details to the columns, we will add some loop cuts (*Ctrl* + *R*) on the top of the roof and extrude the face loops along the normals (*E* + *Alt* + *S*) (refer to step **13** of the preceding screenshot).

Note

Duplicate Linked

This tool allows you to duplicate your 3D model as instances. This means that when you do a modification in the **Edit Mode** on the source object, the transformations are applied to the other duplicated objects in real time. The UVs are also instantiated. However, when you manipulate the object in the **Object Mode**, the changes are not reflected in the other instances.

In order to break the instantiation link, we can use the **Make Single User** menu (**Call Menu** (press *U*) | **Object and Data**).

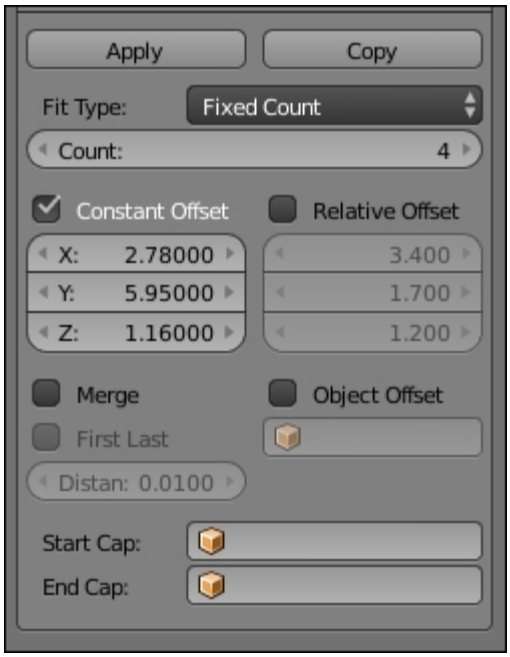


6. We will now work on the bars that delimit the terrace. We will take a new cylinder, this time thinner with a radius of 5 cm and a height of 1.2 m. We will again remove the caps that are pointless here.
7. In order to duplicate our 3D object, we are going to use the Array modifier. We will use a relative offset of 3.100 on the *x* axis. We will take advantage of the replication of the array in order to improve the shape of the bars a little bit with some loop cuts and extrusions (refer to step **14** of the previous screenshot).
8. Since the bars are along a straight line, we will duplicate them with a normal duplicate (*Shift* + *D*) to place them on each side of the terrace. We will also need to adjust the number of bars to match the surface of the terrace.

Note

The Array modifier

This modifier allows you to duplicate your 3D models with a customizable offset. You only need to choose the number of repeated objects that you need with the **Count** parameter and the distance of the offset (constant or relative) on any axis. You can also automatically merge your duplicated polygons with the **Merge** option.

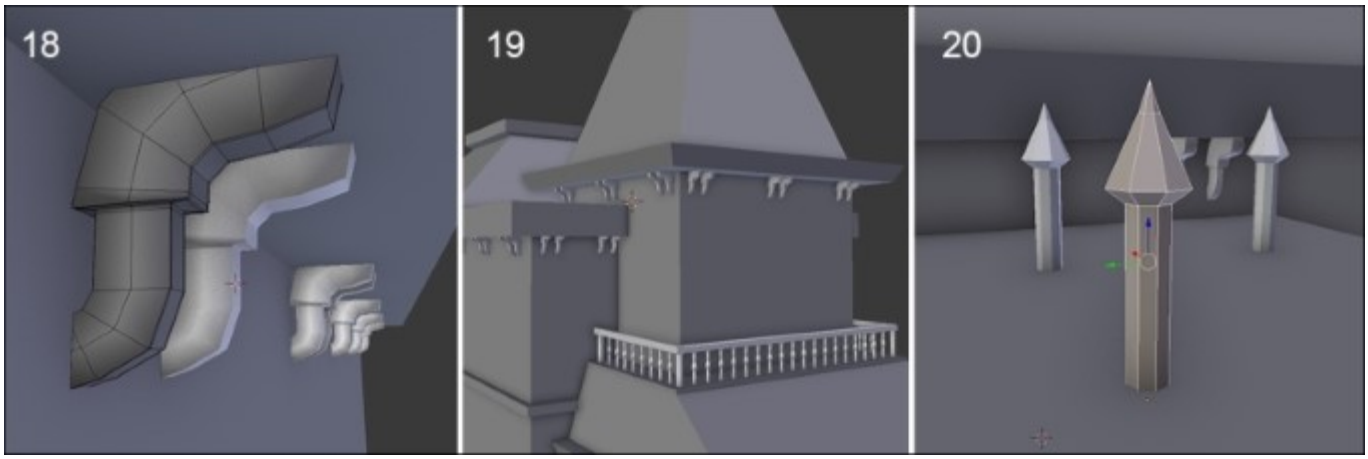


If you want to modify the geometry of a mesh, the array takes the object volume into account, so be careful. A transformation in the **Edit Mode** can change the offset.

9. We will complete this with ramps. The ramp is a simple cube scaled on the *x* axis to make it longer. We will duplicate the ramp as an instance (*Alt* + *D*) wherever needed, but remember that you need to duplicate with *Shift* + *D* if you want to do some changes in the geometry (refer to steps **15** and **16** in the previous screenshot).

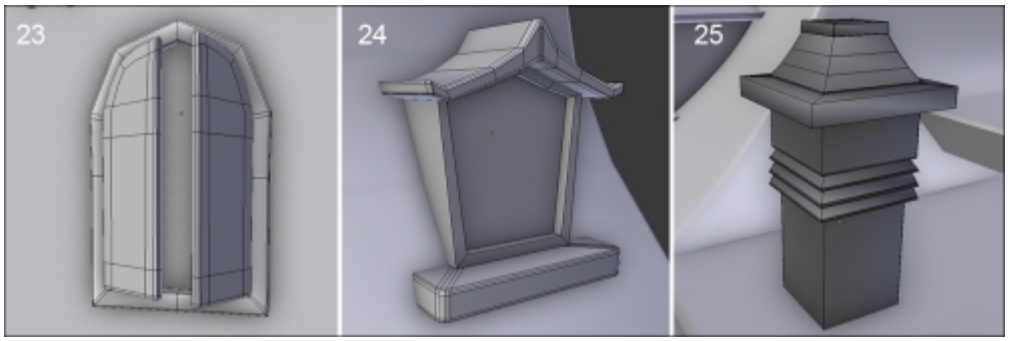
The Duplicate Linked tool is very useful, but it is not very flexible when we only want to do a transform on certain objects.

1. Let's repeat the same technique to make a balcony railing on the top of the roof that covers the front of the terrace. We are going to change the shape of bars a little (refer to step **17** of the preceding screenshot).
2. We will also use an **Array modifier** to make the wall brackets that will support the different parts of the roof. We will use one Array modifier to make a pair and another to duplicate it with a good offset (refer to steps **18** and **19**).



3. The same thing is done for the pikes on the roof that give a threatening look (refer to step **20** in the preceding screenshot).

4. The walls are a bit flat at the moment, so we are going to model a bay window with a particular shape (refer to step **21** in the preceding screenshot). We will start with a new cube (press *Shift* + *A* and select **Mesh** | **Cube**), and we will resize it on the *x* axis to form the base.
5. We duplicate it to make the top part and add an inset for the frames of the windows. When this is done, we will duplicate it as a new instance (*Alt* + *D*), and we will place it on both sides of the main block of the house.



It is also time to make a few conventional old-style windows. We will start with the windows on the front of the house (refer to step **22** in the previous screenshot). Let's start again from a cube:

1. We will delete the left side to use a **Mirror** modifier. From the front view, we will make the shape of the window with a few edge loops (*Ctrl* + *R*) and add an inset. From this model we will can make the frame and extract the shutters (refer to step **23** in the preceding screenshot).
2. We will create another window model for the roof. We will start its base with a flattened (*S* + *Z*) and beveled (*B*) cube. The central part is an extruded edge with an inset, and there is a little roof with a curved slope (refer to step **24** in the preceding screenshot).
3. Let's add a fireplace (refer to step **25** in the preceding screenshot). This is also fairly simple to model, starting from a cube that is extruded and scaled. It uses the same basic modeling techniques that we have previously covered.

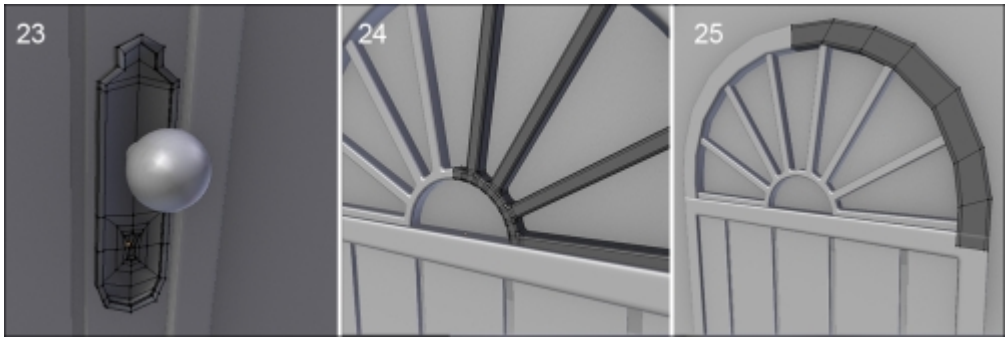
Reworking the blocking objects

Until now, we had quickly added a series of 3D objects to form a fairly rudimentary house and have a general idea of the entire model of the house. Now we will finish the modeling and review the topology of our models in order to make them more presentable.

There is still a final important object to be made: the front door. It is composed of several 3D objects. There is the frame, door, lock, handle, and there is the top frame with a decorative pattern. The frame and the door are quite similar to the windows. The door will be created as follows:

1. We will use a mirror modifier each time. The lock is quite simple, just an extruded cube with a few **insets** (*I*) (refer to step **23** in the preceding screenshot). For the decorative pattern, we will use a circle in order to get a better round shape.

   The rays are extruded and moved by hand. Don't forget to add a few edge loops when you add the **Subsurface** modifier (refer to steps **24** and **25** in the preceding screenshot).
2. We will place the address number of the house. For this, we will use the **Text tool** (*Shift* + *A* and select **Text**). We will use the basic **Bfont** font of Blender, but you can download and use any font instead. We will use the **Extrude: 2.3 cm** and **Depth: 1.2 cm** parameters to make a small bevel. We will make two small nails to hold them.



3. We will now review every asset and remove the nonvisible polygons that are useless in our case. Instantiating many of our objects has greatly facilitated the work.

4. We will also often add a **Subdivision Surface** modifier when it's needed.

   But sometimes, this is not the case, as a simple bevel will do the job as well. As always, you need to maintain your geometry with loop cuts or bevels. In both cases, we want to activate the **Smooth Shading** (in the left panel of the 3D viewport).
5. We will also detach some parts of the geometry from other objects. To do this, we will go in the **Edit Mode** and we will select the faces that we want to detach by pressing *P* and selecting **Selection**.

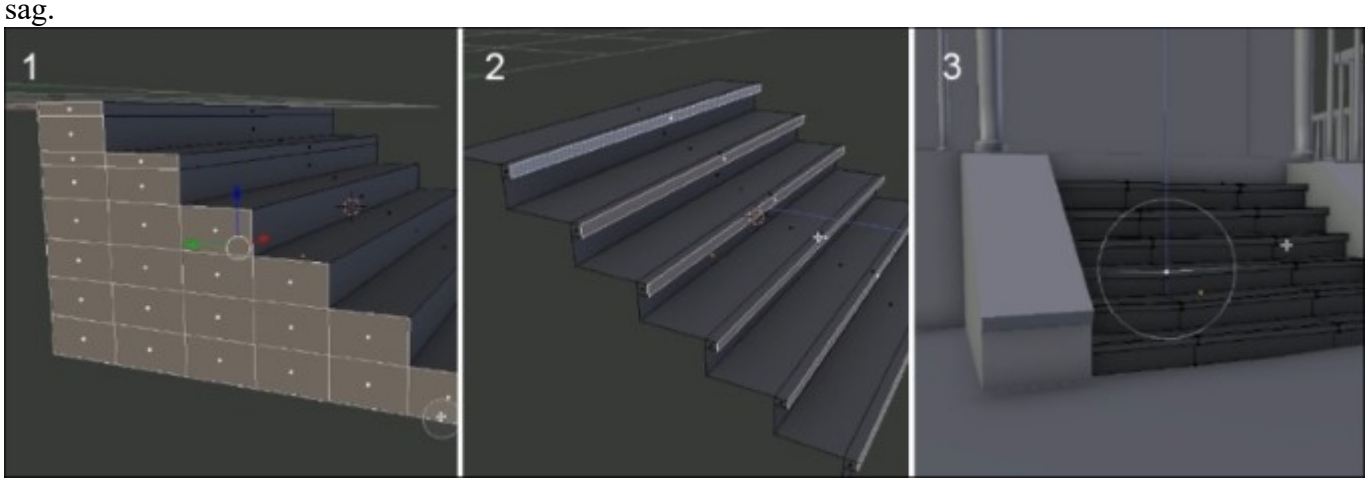
   This will allow us to rework certain objects and add the **Subdivision Surface** modifier with ease. For instance, for the roof, we will detach the center part and add tight edges near the four corners with the **Bevel** tool. We can also increase the windows' resolution by adding and placing some edge loops to form a nice round shape in the corners.

All the improvements that we've done here are, in our case, pretty easy because we don't have to get a realistic result. We are taking a more "cartoony" path, and in the next chapter, we will paint textures by hand in order to add more details.

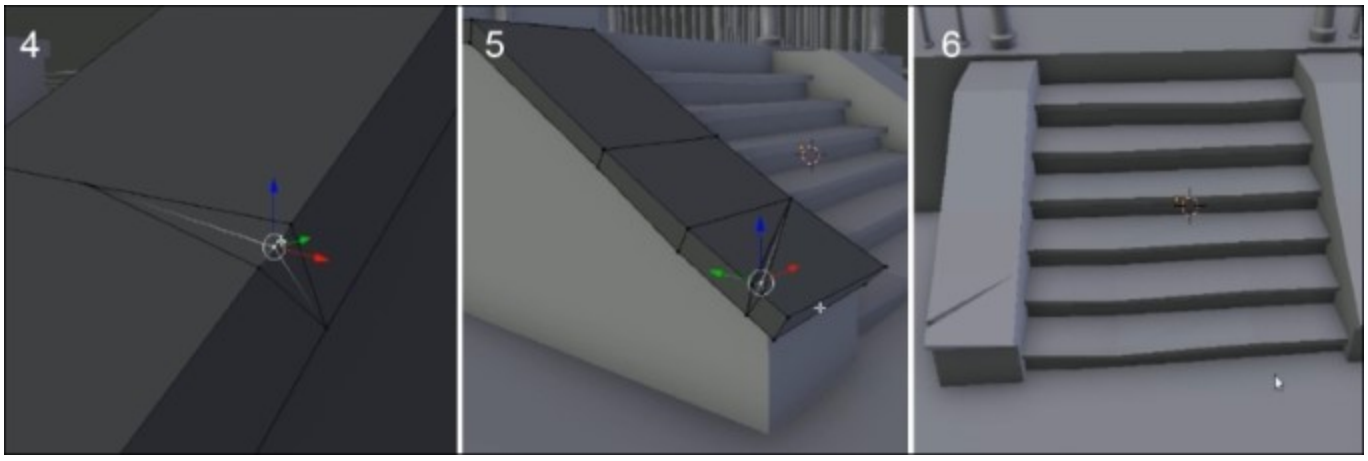
Breaking and ageing the elements

It's now time to refine some elements in order to give them an old/destroyed look. This will mainly come from the texturing work that we will do later, but we can still work on the geometry a little bit:

1. We will begin by working on the stairs. We will delete the side faces that are invisible. (Refer to step **1** in the following screenshot.)
2. Let's add loop cuts below the angle of each stair. This will allow us to extrude a face in the front direction of the top of each stair. (Refer to step **2** of the following screenshot.)
3. We will then add two edge loops at the center to add more resolution.
4. Now with **Proportional Editing** turned on (*O*), we can select some vertices on the stairs and move them to break the shape a little bit. (Refer to step **3** of the following screenshot.)
5. We will now enter the stair ramps object and delete the one on the left. We will have to delete the invisible faces, those that are in the ground and in the house.
6. After this, we can move the origin back to its center (*Ctrl* + *Alt* + *Shift* + *C*) and duplicate the object (in the **Object Mode**) with *Shift* + *D* in order to move it back to the left-hand side.
7. We can then add definition to the object on the right-hand side with the Loop Cut tool.
8. At the top of the ramp, we will use the **Knife** tool and cut around a loop cut that we had made before. We will then push this hole inwards. (Refer to step **4** in the following screenshot.)
9. We can then repeat this process elsewhere on the other object. (Refer to step **5** in the following screenshot.)
10. The stairs are now finished. (Refer to step **6** in the following screenshot.)
11. We can also add a cut in the back wall of the house and move the vertices a little bit to add some sag.



We now encourage you to go over each object and slightly move the geometry, with the **Proportional Editing** tool turned on (*O*), in a random manner in order to break each object's silhouette. You can also rotate the objects, such as the curtains. This is a house that is not new, plus it's probably abandoned!



Simulate a stack of wooden planks with physics

We are now going to add little bit more details by adding a stack simulation of wooden planks in the front of the house. But instead of placing each plank by hand, we will take advantage of the physics engine of Blender that will do the job for us. In order to keep our stack simple and manageable, we will use the instancing principle that we've seen before. Let's start with the modeling of the plank:

1. The plank will be very easy to model by starting with a cube primitive.
2. We will then rescale this cube to make it thinner and larger. Note that if you do this in the **Object Mode**, you will have to apply the scale with *Ctrl + A*.
3. In order that the plank catches the light on its edges better, we will add a small bevel to it. There are two ways of adding bevels in Blender. The first one is the one that we've already used before with *Ctrl + B*. The other method is by adding a **Bevel** modifier. That's what we've done here. Note that if you want, you can apply the modifier too.
4. We will now duplicate the planks by instancing them (*Alt + D*). You need to place them one on top of the other. In order to add a little bit of randomness, we will rotate them slightly in an unordered way.

Note

About the Bevel modifier

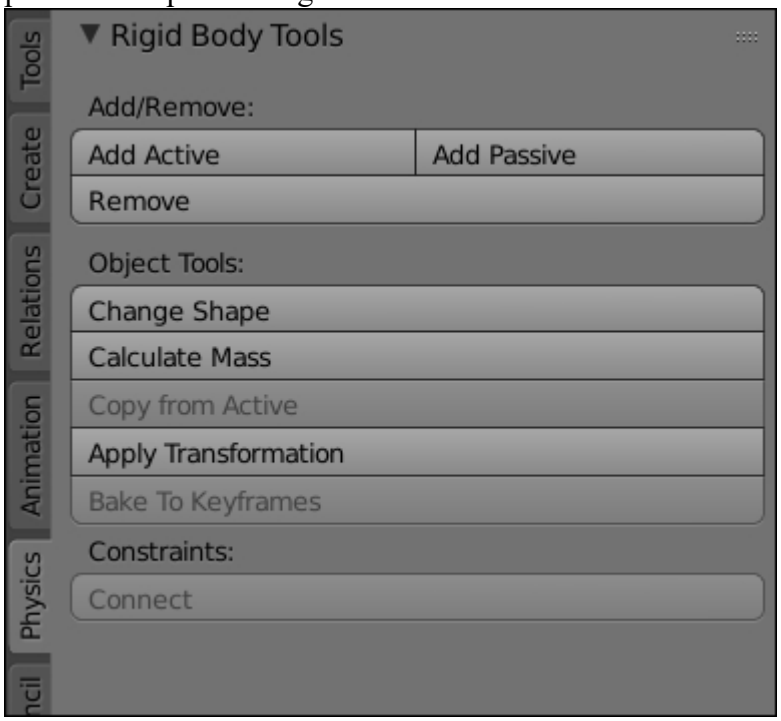
The **Bevel** modifier is nice because it is applied on the whole object, so we don't have to manage a lot of geometry while we are in the **Edit Mode**. We can adjust the **Width** slider to tighten or enlarge the effect of the bevel. The **Segments** option allows us to choose the number of cuts the bevel will be made of. The **Profile** of the bevel corresponds to the direction of the bevel; if it's negative, it will go inwards.

Creation of the simulation of a stack of planks

We will now create our simulation. In any rigid body simulation, the objects have some properties that define them. For instance, you can set their mass, velocity, or simply let gravity act on them as the sole force. In any decent physics engine, you can have static and dynamic objects. A static object is an object that, as its name implies, can't move at all but will be considered in the simulation when collisions occur.

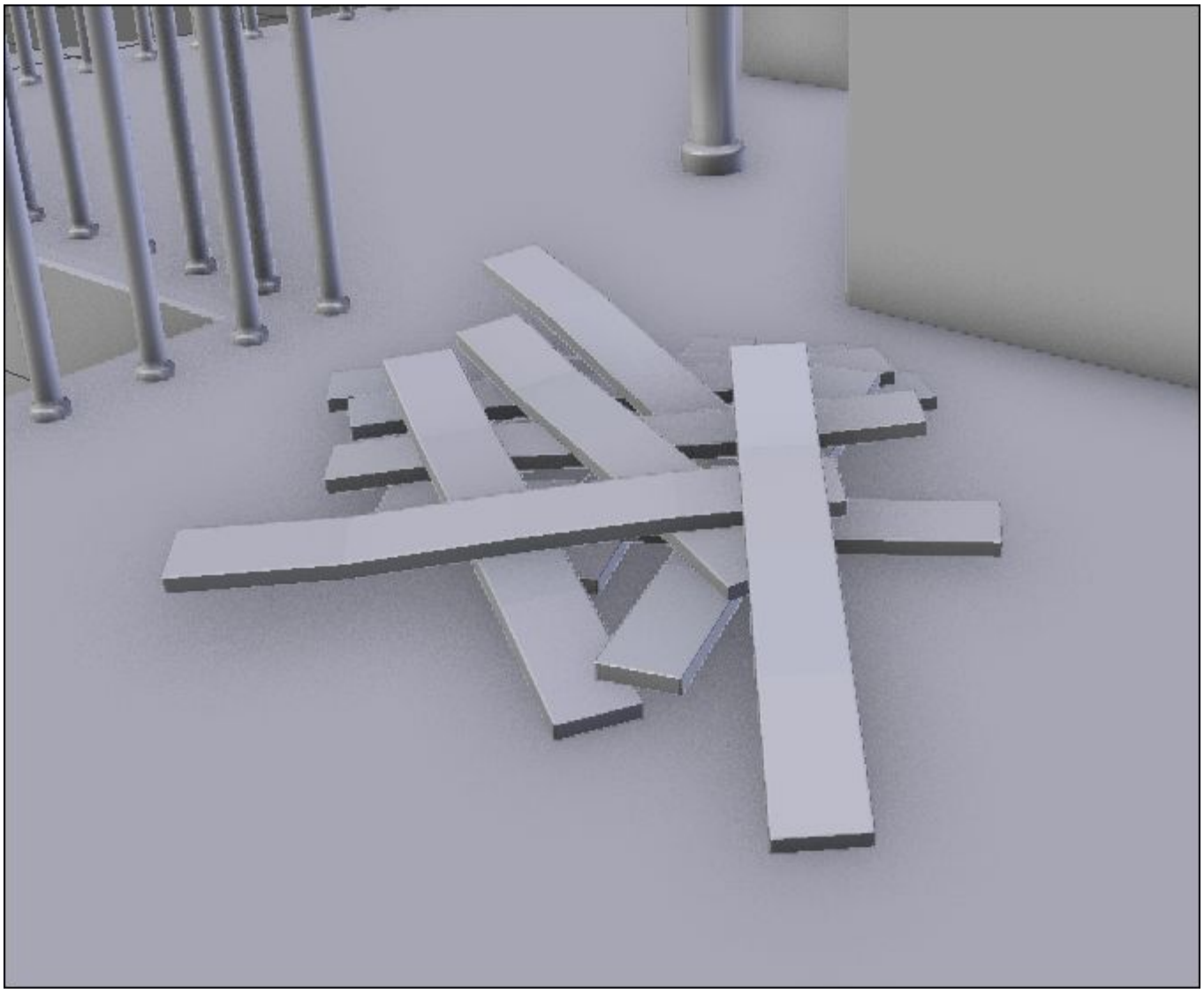
A dynamic object is an object that can receive forces. In Blender, static objects are defined as **Passive** and dynamic objects as **Active**.

1. We will select all our planks, and in the **Physics** tab, in the left 3D view panel (*T*), we will press the **Add Active** button. They will have a green outline.
2. Now we will set the ground object as passive by pressing the **Add Passive** button so that the planks don't pass through the house.



The Physics tab

3. In order to simulate our stack, we will launch the animation. To do this, we will use the *Alt + A* shortcut or press the **Play** button of the **Timeline** editor. Note that if the simulation doesn't seem to launch, you can replace the Timeline bar on the first key frame.



The final stack of wooden planks

4. As you may have seen, there is an orange line on the Timeline that tells us that a simulation has been cached, but as soon as you go backwards in time, the simulation will be removed. So after the simulation has been completed, we will have to apply the placement of each of the planks. In order to do this, we will select them and click on **Apply Transformation** in the **Physics** tab. We can now safely replace the Timeline bar at the first frame, and our stack will rest still.

Modeling the environment (8 pages)

Now that we've finished the house modeling, we will improve our scene with an environment composed of a cliff, a barrier, a cart, and some rocks.

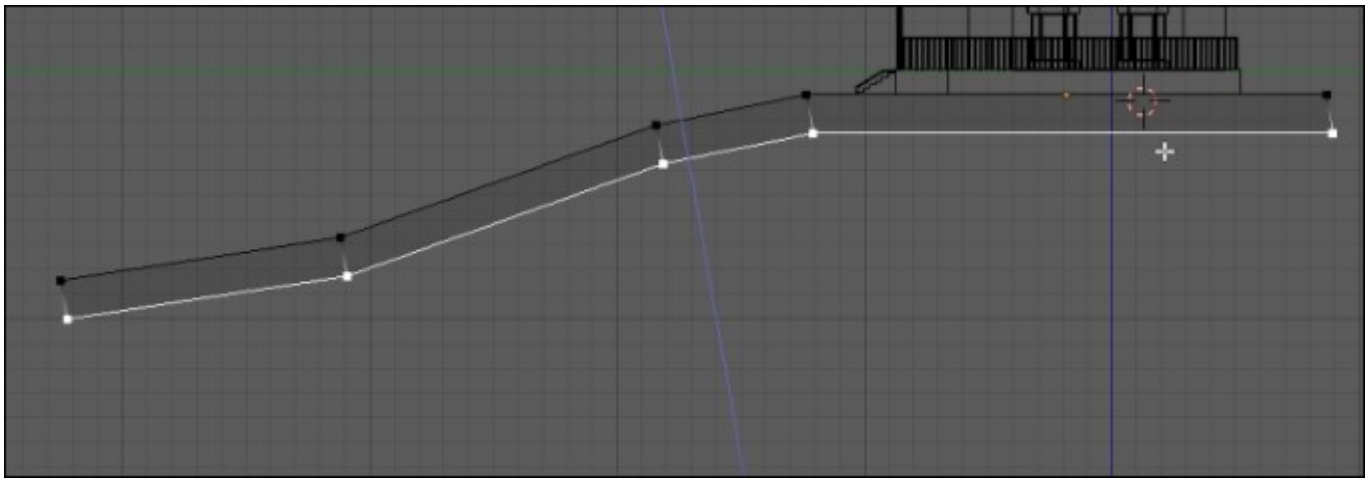
Modeling the cliff

Let's now model the cliff:

1. We will start by modeling its ground part. In order to do this, we add a plane and scale it.
2. We will then move the ground, so the house is placed above it.
3. We will use the scale tool in order to make the ground wider.
4. In the side view, we will enter the **Edit Mode** and activate the wireframe (*Z*). We will select the two vertices in the front of the plane, and by pressing *Ctrl* and the LMB, we will extrude the cliff profile to the left.
5. We will then reshape the geometry from a top view by moving the vertices with the Grab (*G*) tool. The cliff should be narrower where the house is.
6. The whole geometry will then be extruded down to form the height of the cliff.

   ### Note

   Note that you may have the normals of the face pointing inwards causing lighting mistakes. In order to recalculate them, use the *Ctrl* + *N* shortcut.



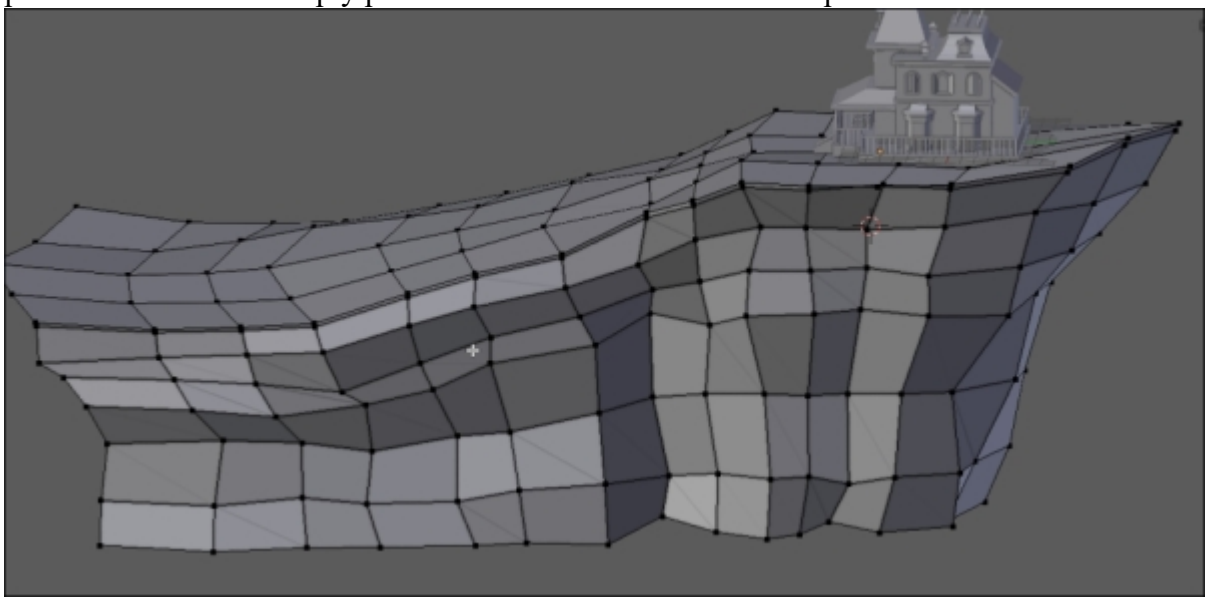
Starting the cliff modeling

7. At this point, we can add some new horizontal loop cuts (*Ctrl* + *R*) in order to add definition to our model.

Now our goal is to give the impression of a rock with the few polygons that we have. To get the shape, we will go in the top view, and with the wireframe turned on, we will select the edges of the contour of

the cliff and move them to form ridges and a valley. This will give an angular look. Remember that all the details will come with the texturing process in the next chapter, so we only have to give the overall shape here.

1. We can improve the model with some randomness. To do this, we go in the **Tool** tab of the left 3D view panel, and under the **Deform** section of the **Mesh Tools** subpanel, we click on **Randomize**. The effect of the tool can be tweaked with the **Amount** slider in the last tool option panel. This tool will simply push all the individual selected components in a random direction.



The final cliff

2. Lastly, we can arrange the global shape of our object with **Proportional Editing**. For instance, we can scale the tip of the cliff with it.

Modeling a tree with curves

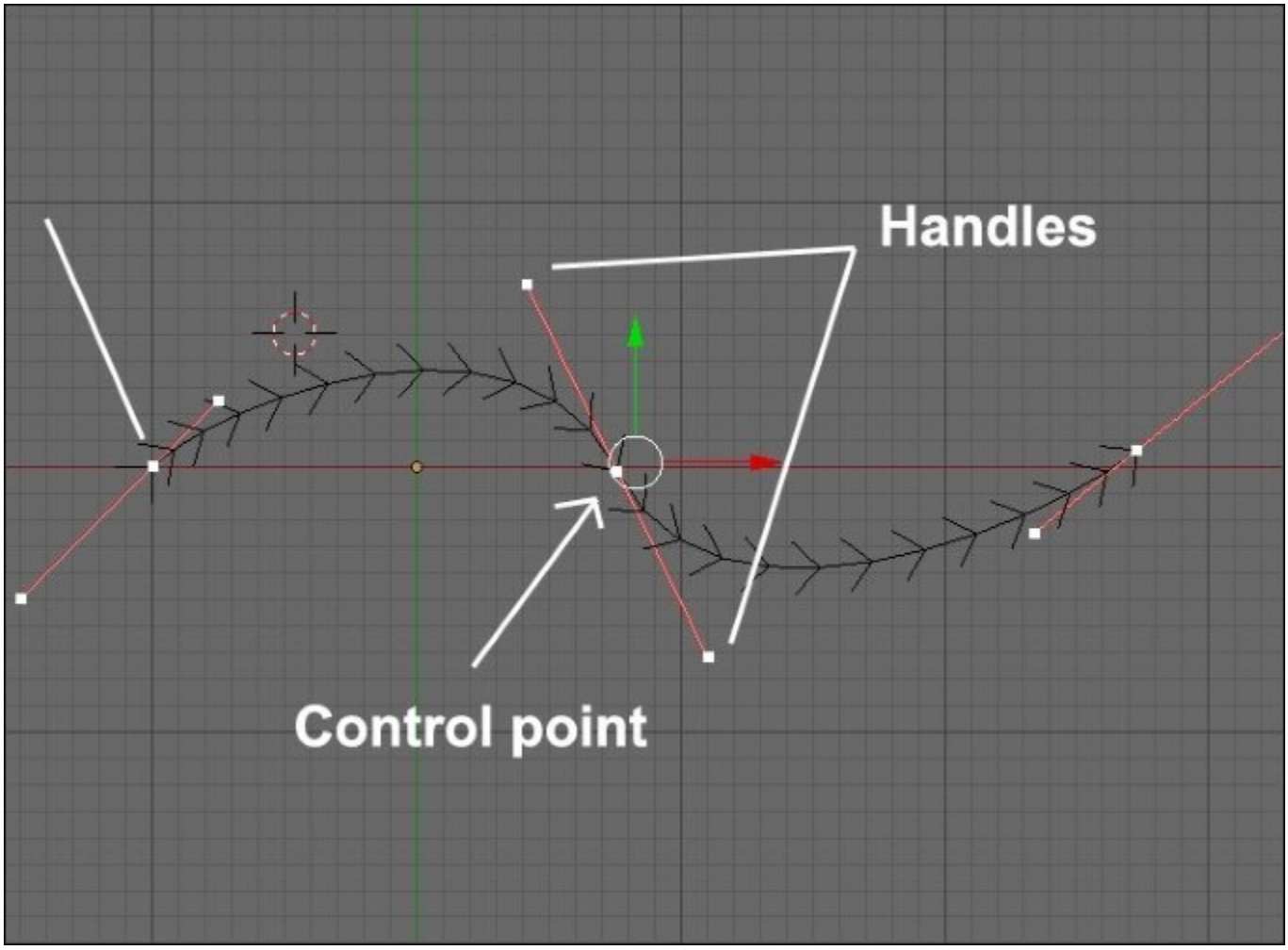
We are now going to learn a new way of modeling certain objects with curves. Curves are entities that can be useful to model ornaments, shoelaces, ropes, tree branches, and so on. In our case, we will model an entire tree with them.

Note

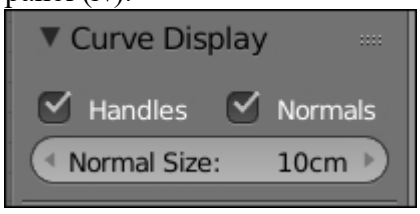
A word about curves

A curve is defined by control points that are connected together. Each point has a handle type. This can be changed by first selecting the control point that you want and by pressing the *V* key. By choosing the **Vector** type, the point will be angular. We are choosing **Automatic** or **Aligned**, so you will have Bezier handles that define the smoothness of the point. Handles always work in pairs. You can break their link by selecting the handle that you want to disconnect and by pressing *V* and select **Free**. In the beginning,

you will find that curves are hard to manipulate, but with a little bit of practice, it will become as easy as pie. But, of course, there is lot more to discover about them!



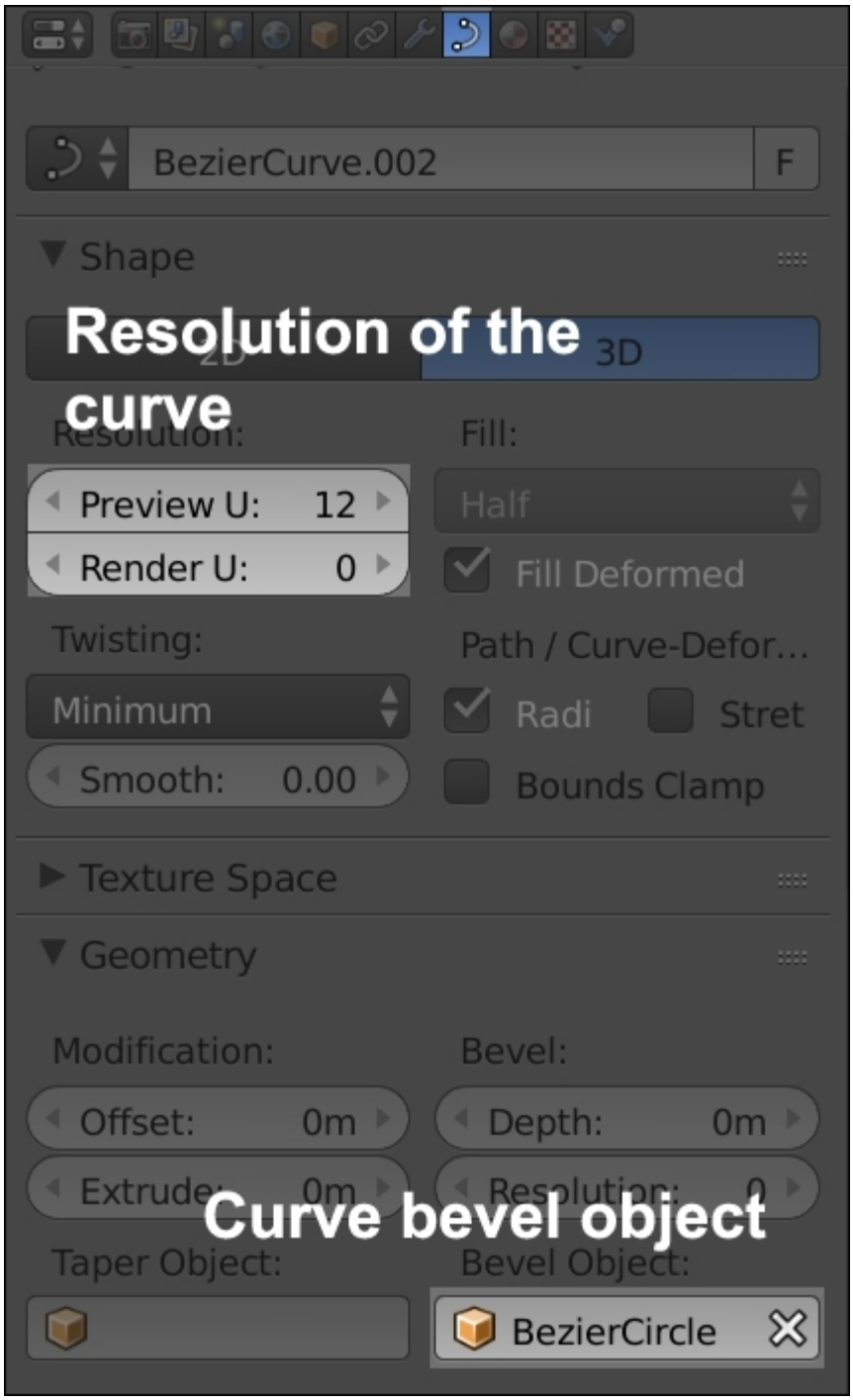
1. We will add a new curve of the Bezier type (press *Shift* + *A* and select **Curve** | **Bezier**). Then we will enter in the **Edit Mode**.
2. We will then turn off the handles and the normals of the curve. To do this, we will uncheck the **Handles** and **Normals** check boxes under the **Curve Display** subpanel in the right 3D view panel (*N*).



The Curve display options in the N panel.

3. When you start working on a shape that uses curves, it's easier to set all the control points as **Vector** (press *V* and select **Vector**).
4. Now we will leave the **Edit Mode** and add a new curve object of the bezier circle type (press *Shift* + *A* and select **Curve** | **Circle**). This circle will define the volume of our bezier curves.
5. We will select the bezier curve, and in the Properties editor, we will click on the object data icon. Under the **Shape** subpanel, we will then select the circle as the **Bevel Object**. As you can see, the curve is now getting a volume defined by the circle. If you change the circle, its scale, for instance, will change the curve volume.
6. We will then close the holes on each side of the curve by checking the **Fill caps** option under the **Bevel Object** one.
7. We can now modify our curve to form the trunk of the tree. To do this, we can simply vertically rotate the two points that we already have and extrude the tip several times with *E* or press *Ctrl* and LMB. You can also subdivide the curve by selecting at least two consecutive points and pressing *W* and select **Subdivide**.
8. Now we can change the radius of each point by changing its value in the **Transform** subpanel in the left 3D view panel. This is how we will taper the trunk.
9. We can now add new branches to our tree by simply adding a new point by pressing *Ctrl* and LMB without anything else selected. This point could then be extruded.

   We can also change the radius of the branch. If you want to quickly select a branch, you can either select one of its points and press *Ctrl* + *L*, or select one of its points and press *Ctrl* and the + numpad key several times.
10. We will now add more branches by duplicating the first one that we've created. The process simply involves the duplication and placement of a lot of branches that differ in size and radius.
11. Next, we can convert our curves to a mesh object. In order to do this, we will have to first decrease the subdivision of our branches and our circle profile object by going to their **Object data** tab in the Properties editor and by changing the **Preview U** (for the viewport) and **Render U** (for the final render) sliders. We can now select our curve object and press *Alt* + *C* and select **Mesh from Curve/Meta/Surf/Text** to convert our curve object into polygonal geometry.



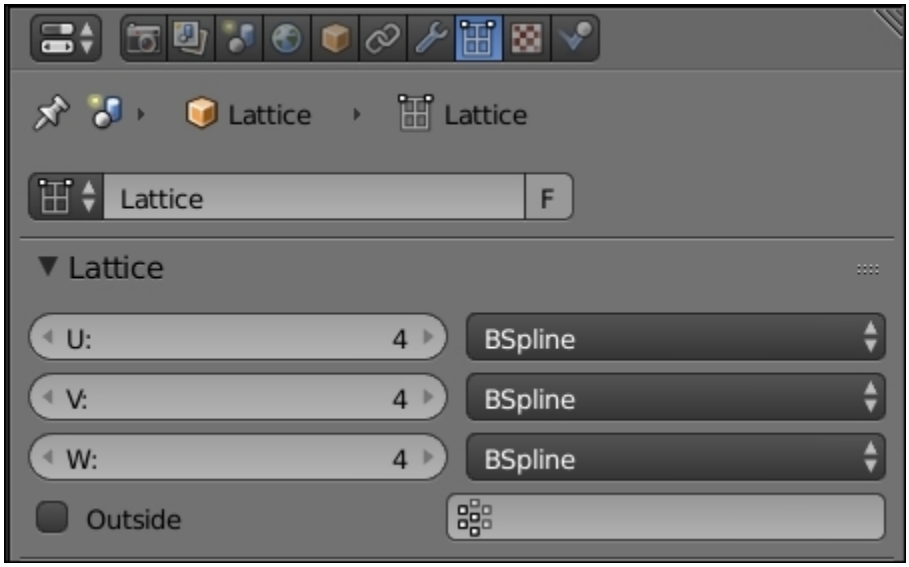
The curves option in the Properties editor

We will now show you how to deform the tree with a special type of object called a lattice. This will serve us to change the overall shape of it.

Note

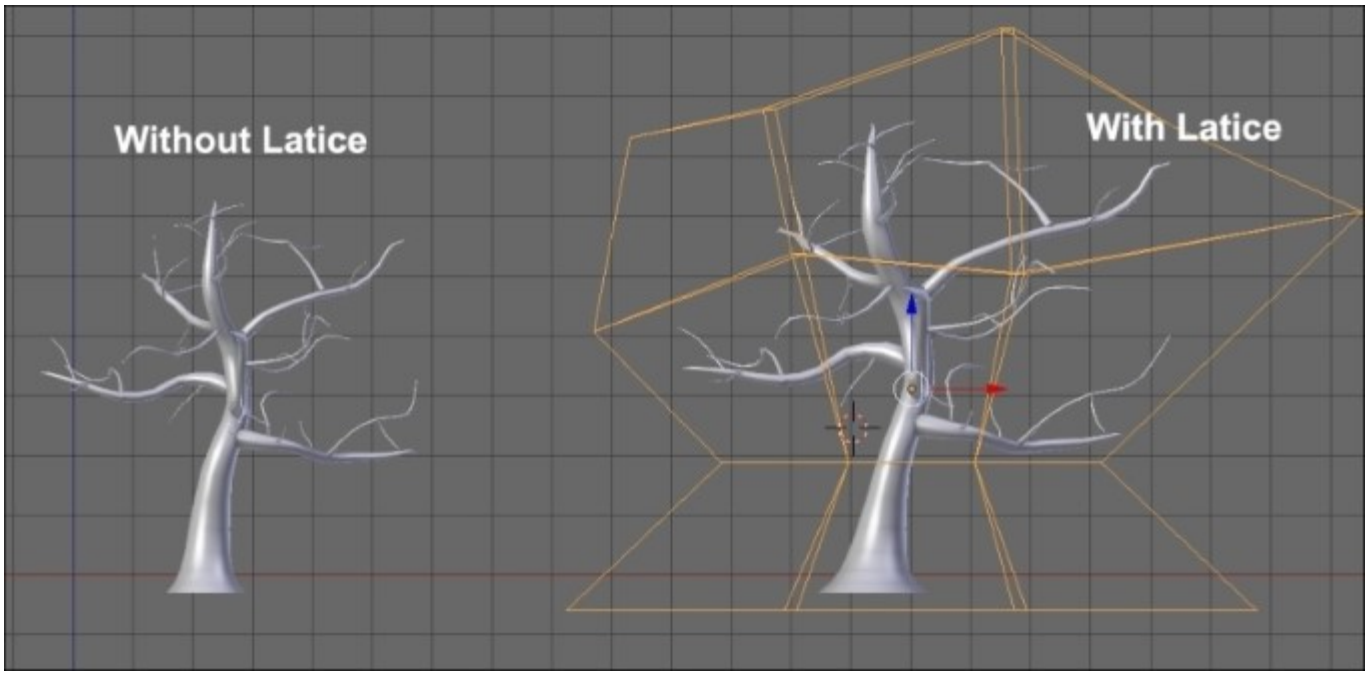
What exactly is a lattice?

A lattice is a very useful object that is bound to another object. It is a cube that can be subdivided along three axes: U, V, and W. This needs to encapsulate the object to which it is bound. Then, when we move its control points, it performs a sort of a projection that allows us to deform the bound mesh.



*The **Lattice** options in the Properties editor.*

1. We will add a new lattice by pressing *Shift + A* as usual.
2. Now that we have a new lattice, we can place it around our tree. No geometry should be outside the lattice or it will cause problems.
3. We will now increase the subdivisions of the lattice in U, V, and W to **4**. This will gives us enough control points in order to tweak our tree.
4. Now it's time to bind our lattice to our tree by simply selecting our tree and adding a new **Lattice** modifier. We will now have to specify to Blender the lattice object that we want to bind to our mesh in the **Object** parameter. In our case, we only have one lattice so it's easy to find it in the list, but if you have many lattices, don't forget to name them.
5. We can now enter **Lattice Edit Mode** and move the control points as if they were vertices.

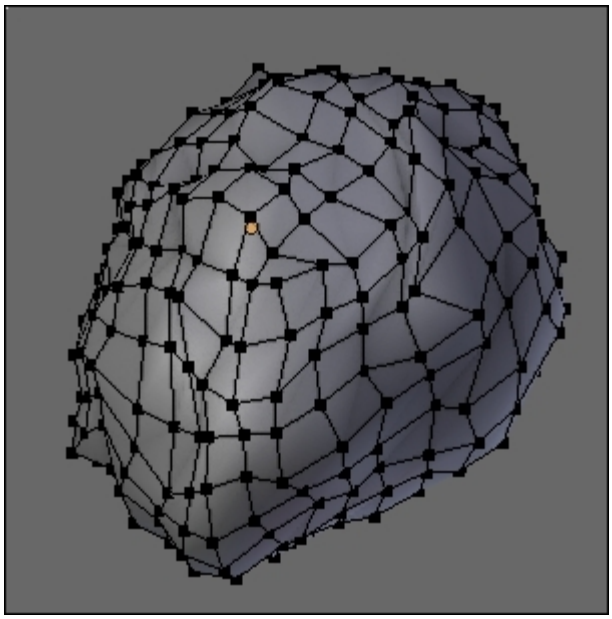


The final tree with its lattice

Enhancing the scene with a barrier, rocks, and a cart

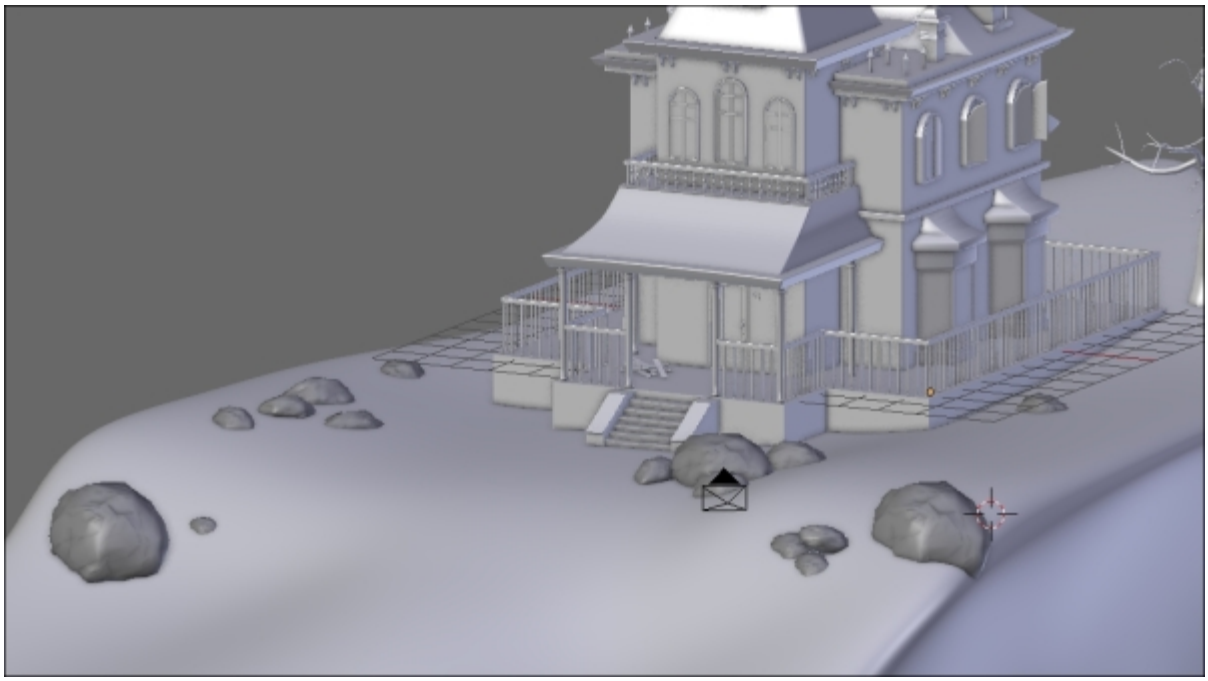
We will now enhance our scene by adding some cool assets using the techniques that we've seen in this chapter:

1. Let's start with the easiest asset, the rock. We will start this element with a cube.
2. Now in **Edit Mode**, we will use the **Subdivide smooth** option (press *W* and select **Subdivide smooth**).
3. In the last tool subpanel of the left 3D view panel, we can tweak the **Fractal** slider in order to add some randomness to the subdivision.
4. We can now repeat steps **2** and **3** several times.



A single rock

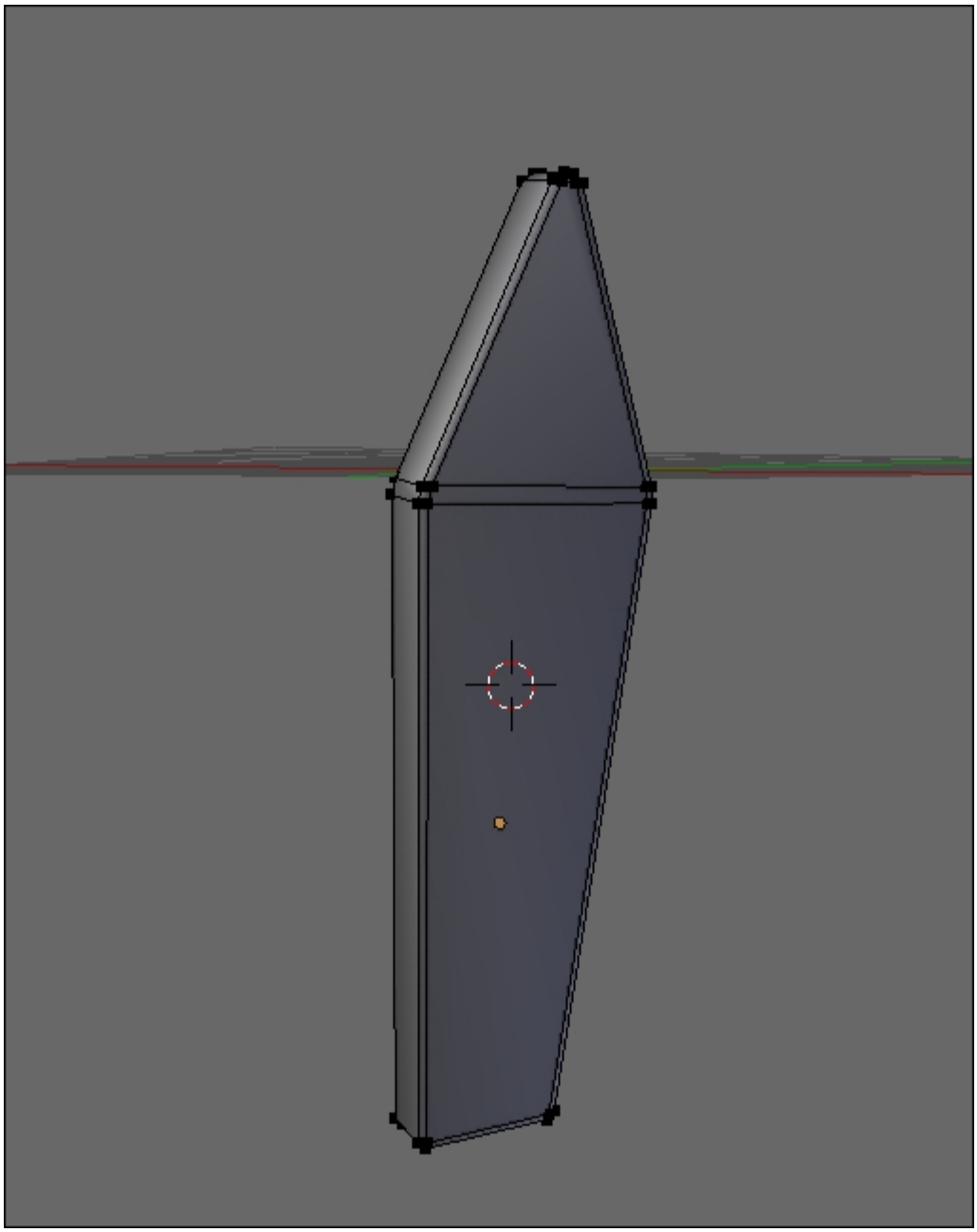
5. Now that we have a pretty spherical rock, we will use a **Lattice** modifier to shape it more like a rock. In our case, we will set our lattice with three subdivisions in U, V, and W, and after we've done the job, we will apply the modifier on the rock.
6. We can now use the instance duplication (*Alt + D*) tool in order to populate our terrain with rocks. This is a little bit of cheating, but we don't have to do any other objects in order to give the impression that there are many different rocks. Indeed, we can simply rotate and scale the rocks in a random manner! In this case, the Free Rotation tool (press *R* key twice) is quite useful.



The rocks are now placed all around the house.

We will now create a barrier in front of the house with a very nice modifier.

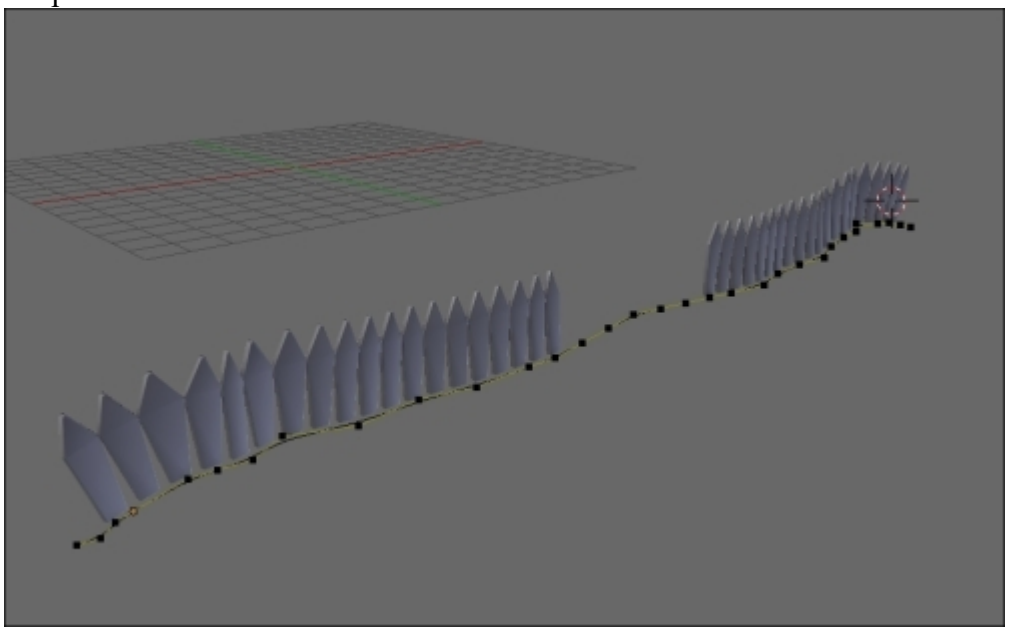
1. We will first add a new cube that we will scale on the *x* and *y* axes to make it thinner.
2. Next, we can select the top face in the **Edit Mode** and move it to change the height of the plank.
3. With the top face selected, we will extrude the top part of the plank and scale it on X.
4. We can then select all the objects (*A*) and use the **Bevel** tool (*Ctrl* + *B*) to smooth their edges.



The barrier plank

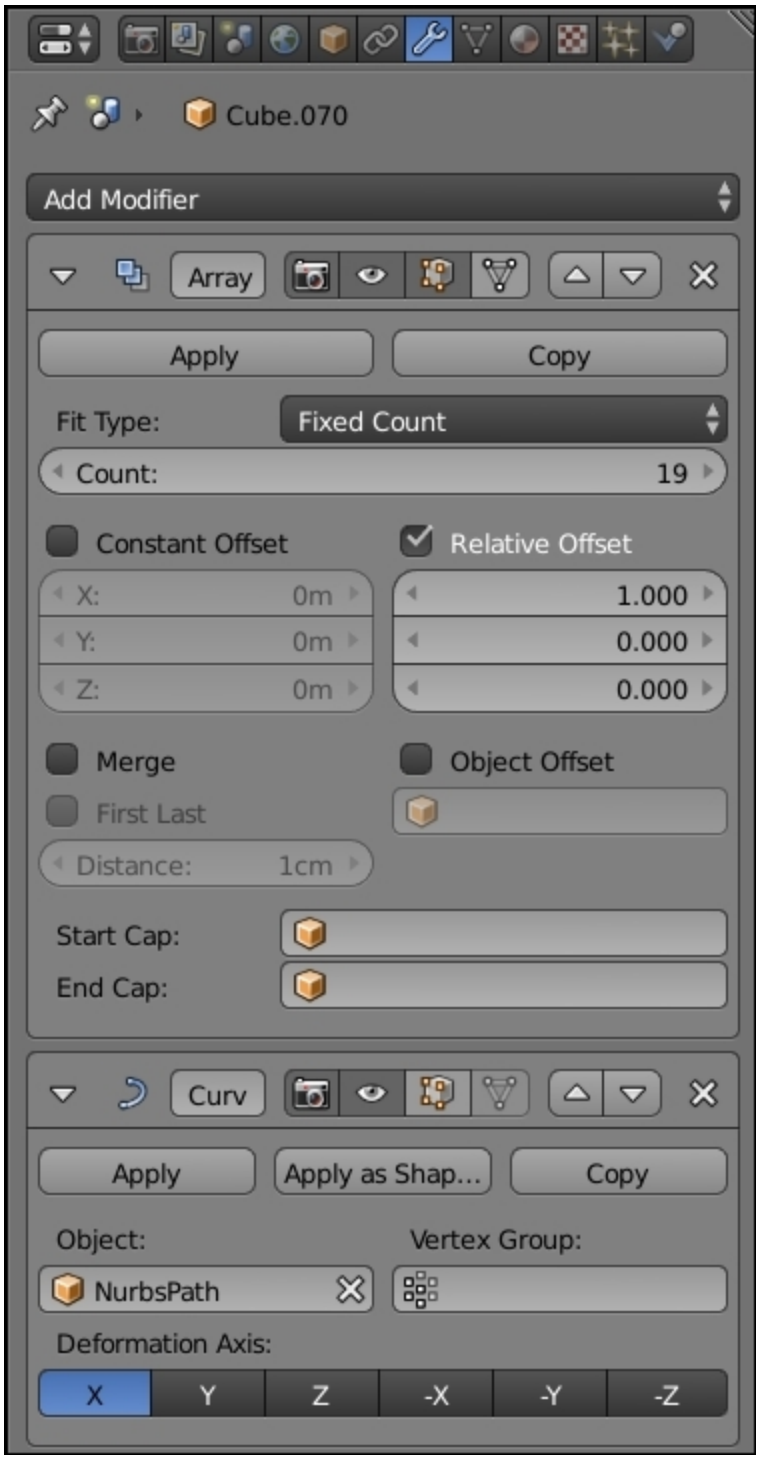
5. We can now place it on the far left of the cliff in front of the house.
6. The next thing is to use the **Array** modifier in order to do a line of planks to the middle of the house.
7. We can duplicate this barrier piece on the other side.
8. In order to break the boring alignment of these objects a little bit, we will use a path curve (*Shift* + *A* and select **Curve** | **Path**) that will guide the planks.

9. The path object looks like a curve. We can select each control point, extrude them, and subdivide them. We will extend the path object from the far left to the far right of the cliff and subdivide it several times (*W* and select **Subdivide**).
10. We can now break the path by moving each control point in a random way.
11. On the barriers, we will add **Curve** modifiers and select our path as the object. In our case, the deformation axis is *x*. Be careful that curve modifier is placed above the array of the stack. Indeed, we want to deform only one plank. As you can see, we now have our barrier following the path.



The final barriers with their curve

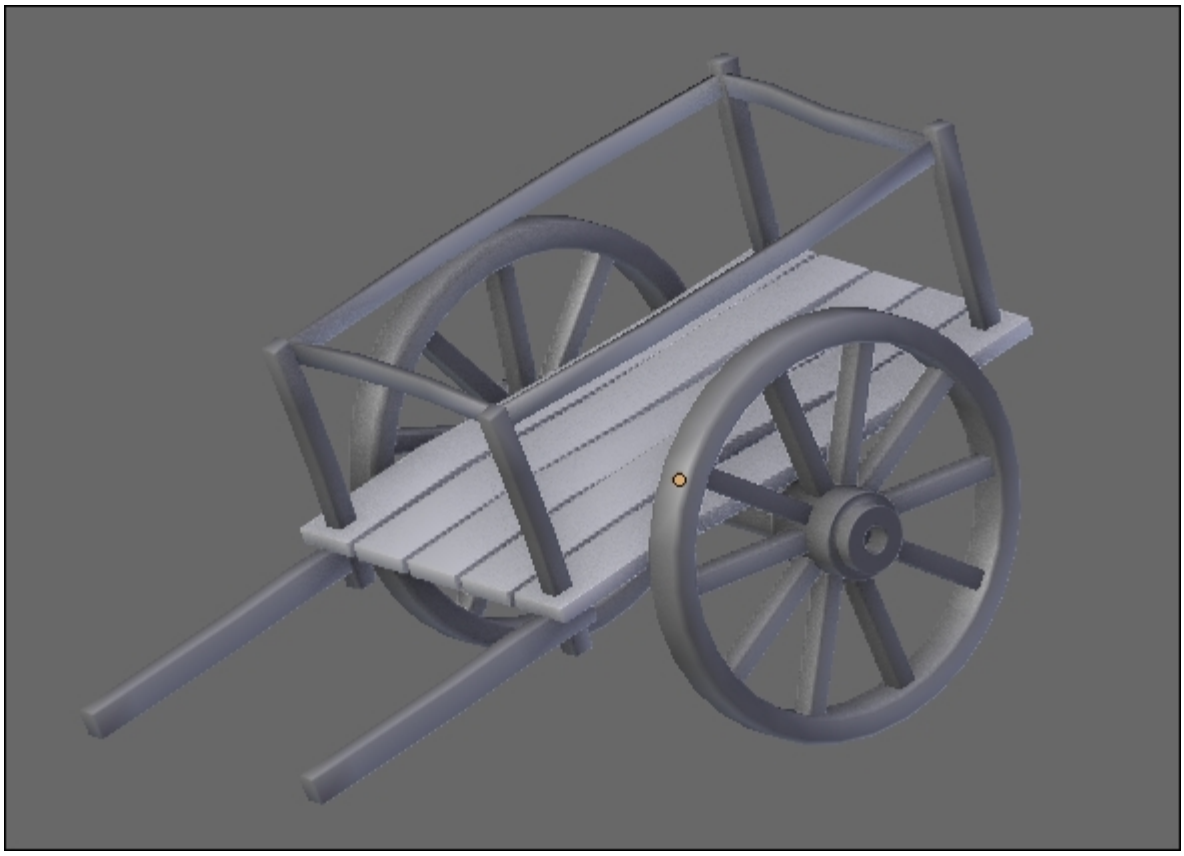
The following screenshot shows the barrier modifiers:



The barrier modifiers

It's now time to model the cart, which is a more complex object, but still simple to do with all the tools you've learned until now:

1. Let's start by duplicating one of the planks that we used in our physics simulation. This time we want to modify it without breaking the ones that are part of our simulation, so we press the *U* key and select **Object & Data** to make it a single user.
2. We can now use an **Array** modifier to make the bottom part of our cart. We can specify a little margin between each of them.
3. Now we will apply the modifier and tweak each plank so that they don't look the same.
4. We will now add a small cube and change its height. We can break its silhouette a little bit with **Loop Cut** (*Ctrl* + *R*) that we will move, and finally, add a bevel.
5. Now we can duplicate this object (*Shift* + *D*) at each corner of the cart.
6. We can now add a small plane that we will extrude by following the outline of the cart. We will add a **Solidify** modifier and apply it.
7. We can now add a new cube that we will shape into handles. As always, it's a matter of loop cuts, bevel, and placement.
8. Now, the last thing we need is two big wheels. To do this, we will add a circle, and in the **Edit Mode**, we will extrude it along the normals (press *E* + *Alt* + *S*). We can also extrude the wheel to give it a thickness. Remember to check your normals' orientations and clean them with *Ctrl* + *N* if needed.
9. Now we will add some radius. To do so, we first select one of the inner faces of the wheel, duplicate it and extrude it. We can select the whole radius and press *P* for **Separate selection**. We will also need to change its pivot point at the center of the wheel (press *Ctrl* + *Alt* + *Shift* + *C* and select **At center**).
10. We then duplicate the wheel with *Alt* + *D*, and without validating the action, we will press *R* to rotate it. This is because now we can press *Ctrl* + *R* to repeat the duplication and rotation process. What a time saver!
11. In order to cap the wheel, we will add a cylinder that will be extruded several times.
12. Finally, we can duplicate the wheel on the other side with *Alt* + *D* and add a cylinder that will connect both wheels. Remember to delete the unneeded faces on each side.



The final cart

As you can see, the items that we created are quite simple, so we encourage you to add more of them, such as a dead trunk or a scarecrow. Be creative, you have all the tools needed to do that yourself.

Organizing the scene

This is maybe the largest scene you have ever made, in terms of details and number of objects. So we will take some time to learn how to organize ourselves in order to have a more manageable scene.

Grouping objects

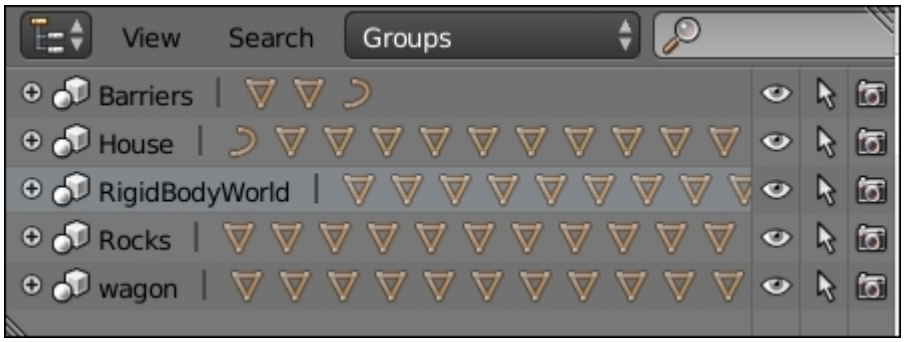
One interesting thing that Blender has to offer is the grouping tool. Let's see how it is working:

1. We will group objects that have a logical relationship between them. So we first select every part of the house, such as the foundations, the walls, the curtains, and the fireplace, and we will place them in a group by pressing *Ctrl* + *G*.
2. If we look at the last tool options in the left 3D view panel, we can enter a name for our newly created group.
3. Now we can select barriers and create a new group with them.
4. We'll leave the cliff alone, so we don't have to put it in a group.
5. The rocks will also be part of a group.
6. We will also group all the cart pieces.

Note

Using groups

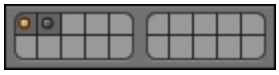
Groups are very useful as they allow you to organize a scene better. You can see all the groups that are in a scene in the outline by changing the **All Scenes** drop-down menu to **Groups**. Here you can select them, disable their render, and hide them. You can also select one object that is part of a group in 3D view and press *Shift* + *G* and select **Group** to select all the other objects of the same group. You can add or remove an object from a specific group in the **Object** tab of the Properties editor under the **Groups** subpanel.



Working with layers

Another thing that can help your scene organization are the layers. They are located in the 3D View header. You can move objects on specific layers with the *M* key. A popup will show you all the layers.

You can move an object on multiple layers by pressing shift and selecting the layers that you want. The layers that have objects are marked with a little circle. You can display multiple layers at a time by pressing the *Shift* key and clicking on them in the header bar. Note that you can also see the layers in **Scene** tab of the Properties editor. In our case, the house is on layer one and the other elements on layer two.



Layer one is selected, but we can see that there are objects on layer two

Summary

You have completed the first part of the Haunted House project! You have learned some tricks to save time, such as how to instantiate objects or how to use the array modifier. You have also learned another modeling method with curves. You will clearly see the purpose of duplicating objects by instance in the next chapter where we will UV unwrap the objects. But don't think that it's over with the modeling process here. We can later change our objects after the texturing part in order to break the likeness between each instantiated object. Let's texture our scene!

Chapter 6. Haunted House – Putting Colors on It

This chapter will be devoted to the texture creation pipeline. We will explain new ways to you for the UV unwrapping processes, such as the **Project From View** or the **Smart UV** method. Then we will cover the basis of the powerful Texture Paint tool in order to create hand-painted textures for our house and environment. You will also learn about the tiling method in order to save time. In order to enhance the final set, we will show you how to create transparent textures, such as grass or grunge to age the house. In this chapter, you will not learn how each object is unwrapped and painted step by step, but you will gain a thorough understanding of the process. Finally, you will learn how to produce a test render with Blender Internal. This will be a good transition to the next chapter. So let's dive into the texture creation process with Blender!

In this chapter, we will cover the following topics:

- Learning the different ways to project UVs on an object
- Painting your model with the Texture Paint tool
- Creating hand-painted and tile-able textures
- Baking diffused textures on proper UVs
- Making transparent textures
- Creating a draft render in Blender Internal

Unwrapping UVs

We will now cover the UV unwrapping process that will later allow us to add textures to our objects.

Using Project From View

In order to quickly unwrap the UVs of the walls of the house, the shape and size of the wall still being accurate, we can use the **Project From View** method.

For this to work, we need to align the walls on the axis of the world coordinates (X, Y, Z). Indeed, we are going to make use of the angle of the 3D View of Blender to project the UVs. So, for each wall, we are going to rotate our view so that the wall that we are going to project is flat. This method only works for objects that are flat and aligned, so don't use it for organic shapes:

1. We will enter the Orthographic mode (*5*) and then enter the view that corresponds to the face that we want to unwrap. For instance, in order to unwrap the UV of the front wall we go into the front view (*1*).
2. In the **Edit Mode** and in the **Face Mode**, we will select all the polygons of the 3D model that we want to unwrap and be careful with the beveled parts.
3. In the **UV Mapping** menu (*U*), we will select the **Project From View** option.

   We will obtain a perfect UV Island that is well proportioned according to the mesh. Usually, we don't need to tweak UVs with this method, except when we are willing to scale them. To check the size of our UVs, we will use a UV grid texture.
4. We will create a new editor by selecting **Split Area**, then we will go to **UV/Image Editor**.

5. We will need to create a new image. So we will chose **Image** | **New Image** and **Generated Type** | **UV Grid**.

This UV grid allows us to adjust the island's size at the uniform scale for all walls of the house.

We will use **Project From View** for all the walls as well as the big flattened surface such as the platform of the terrace. For the later one, we will avoid having seams that are too visible:

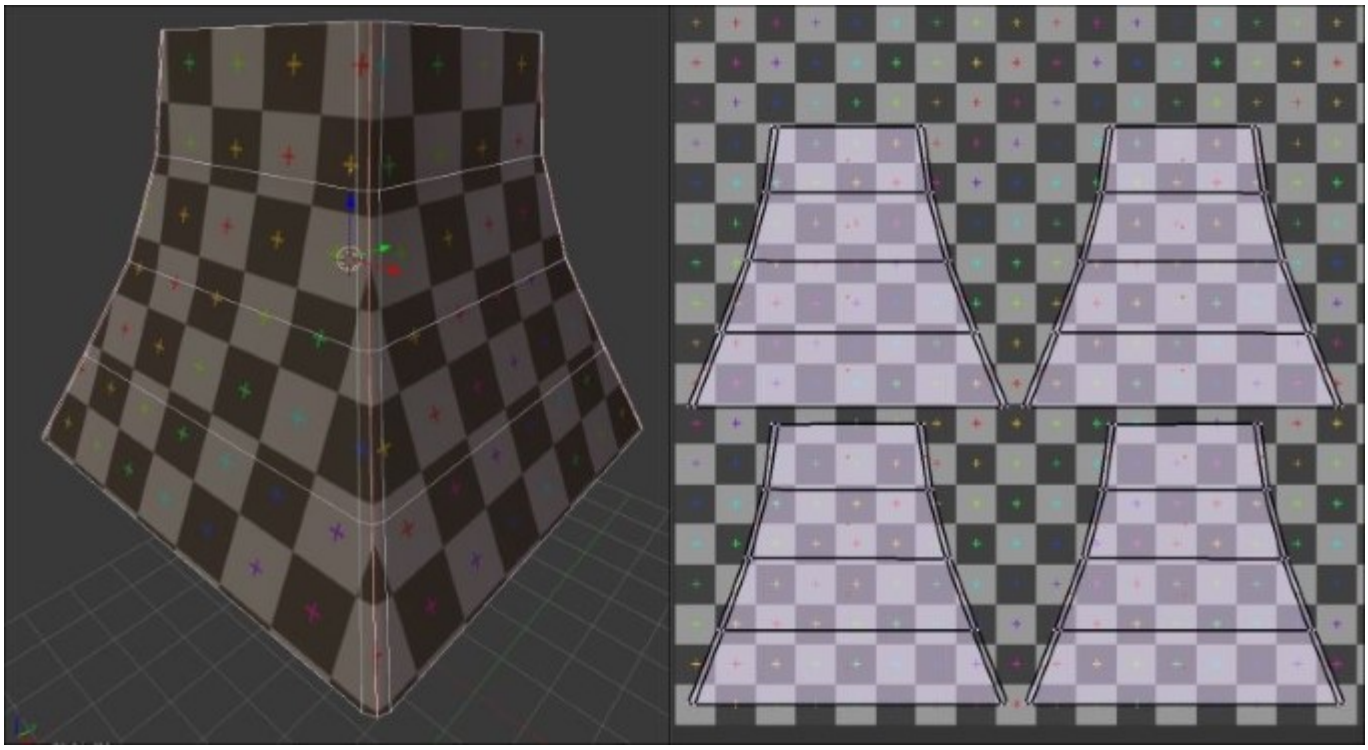
1. We will select all the top faces with the bevel part. We will then go into the top view (**7**), and we will do a **Project From View** unwrap.
2. Still being in the **Project From View**, we will select and unwrap the polygons of the sides, one after the other.
3. To make some seams invisible, we need to weld the vertices of some of the side islands of the platform with the central island. In order to weld the vertices in the **UV/Image Editor**, we will select them and press *W* | **Weld**. This acts like the merge tool in the 3D Viewport but in the UV space.

Unwrapping the rest of the house

For the other parts of the house—that is, the front door, windows, fireplaces, roofs, staircase, columns, bars, railing, and other little objects that form the house—we will place the seams to demarcate the UV islands like we did before with the Alien Character – Creating a Proper Topology and Transferring the Sculpt Details in Chapter 4, *Alien Character – Creating a Proper Topology and Transferring the Sculpt Details*. We keep using the UV Grid to check the scale.

We will start with a pretty simple object, that is, the roof of the front block of the house:

1. We will place the seams to delimit the four sides evenly. We will select the desired edges with a right-click, and then we select **Mark Seam** in the **Edges Menu** (*Ctrl* + *E*).
2. We will make an automatic unwrap (press *U* and select **Unwrap**) that allows us to reveal the UV without any deformation, but to check this we will use the **Stretch** option in the right panel (press *N* and select **Display** | **UVs** | **Stretch**). You will recall that blue means there are no noteworthy deformations.
3. We must keep a little space between the different UV islands. Indeed, we will have a small gap that exceeds the islands later with the baking process. We will need to provide sufficient space in order to not see the seams on the 3D mesh displaying the textures.
4. We will reduce our UV islands and try to keep the same scale for the walls and the platform of the terrace. The position is not important for the moment.



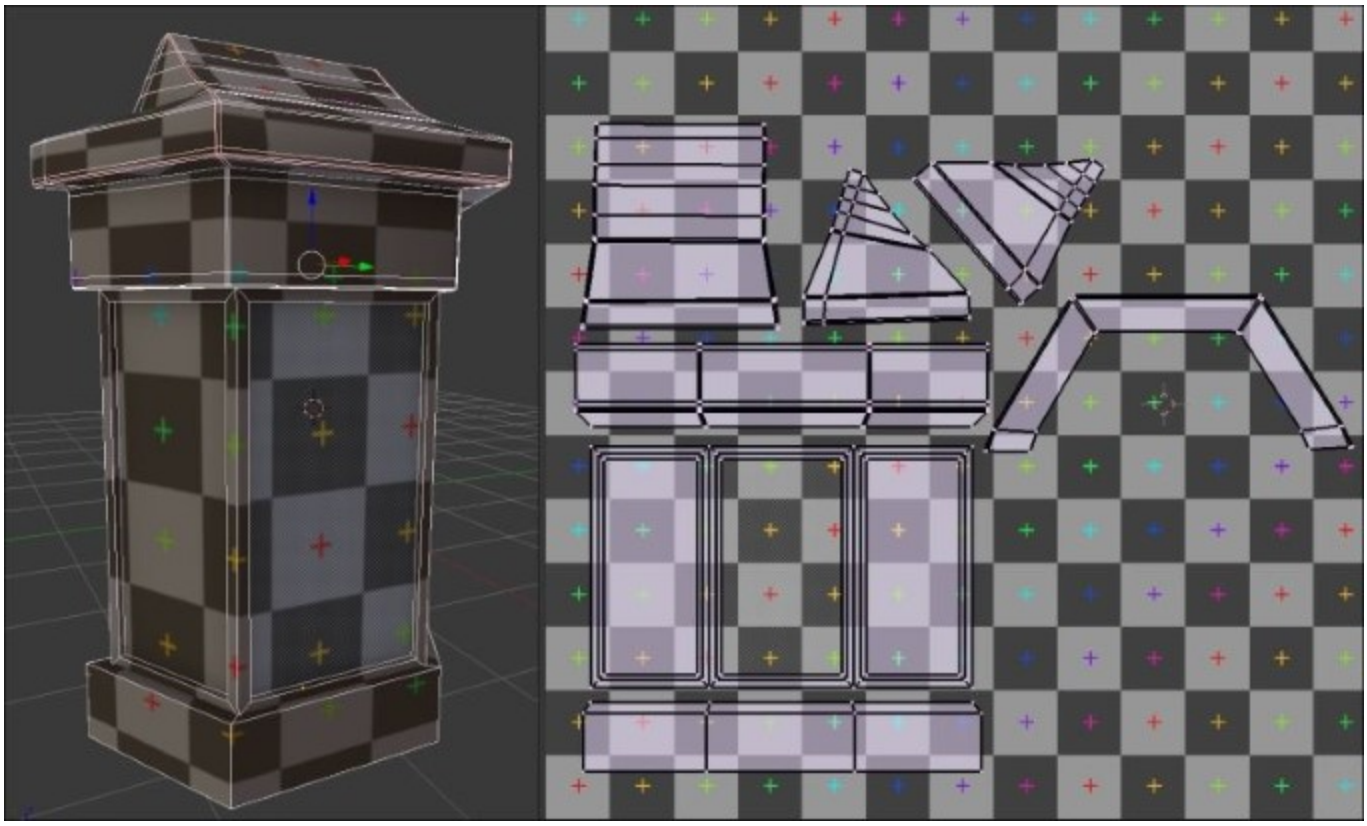
We will continue with a slightly more complex object: the bay window.

You may have observed that the object has formed recurrences. It is angular. So we will avoid laying the seams directly on steep angles and take advantage of the beveled geometry if possible. We will also try to get a UV continuity on the window frame, the lower part and the upper part of the window frame. We must, therefore, make some adjustments after the UV unwrap.

The goal by doing the UV is to try to reproduce a flattened shape of the 3D mesh with the least distortion possible. This is done as follows:

1. We must place the seams at the roof level of the bay window. We will only use seams on this part of the object. We will cut the roof into three parts and three others for the bottom part.
2. As we have detached the other parts of the bay window before, we don't need to place the seams. If, in your case, all the faces are welded, take the time to place the seams.
3. We will select the entire object, and we will perform an **Unwrap** operation (press *U* and select **Unwrap**). This gives us seven UV islands that we can place in a corner for the moment.

We don't have to select all the faces of an object to unwrap the UVs; we can select and unwrap only a few selected faces, but we will lose the scale based on the other faces. This forces us to make some adjustments.



1. We will check whether there are deformations with the **Stretch** option (press *N* and under **Display** select **UVs** and **Stretch**).
2. Now we will align the edges that need to be like the window frame. We will select the edges that must be vertically aligned, and we will scale them on the *x* axis (use the *S*, *X*, and *0* shortcuts).
3. Similarly for the edges that need to be horizontally aligned, we will select them and we will scale them on the *y* axis (use the *S*, *Y*, and *0* shortcuts).
4. Once the object is aligned, we will place a UV Grid texture to check the scale.

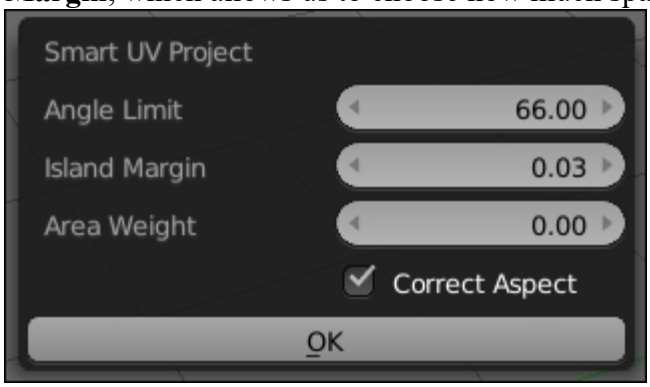
Aligning the edges is a process that can be long but it is important to do this for objects such as pipes or angular structures. There are add-ons that can save us some time, such as **Quad Unwrap**.

In this way, we will continue unwrapping the UVs of other objects that compose the house.

The tree with the Smart UV Project

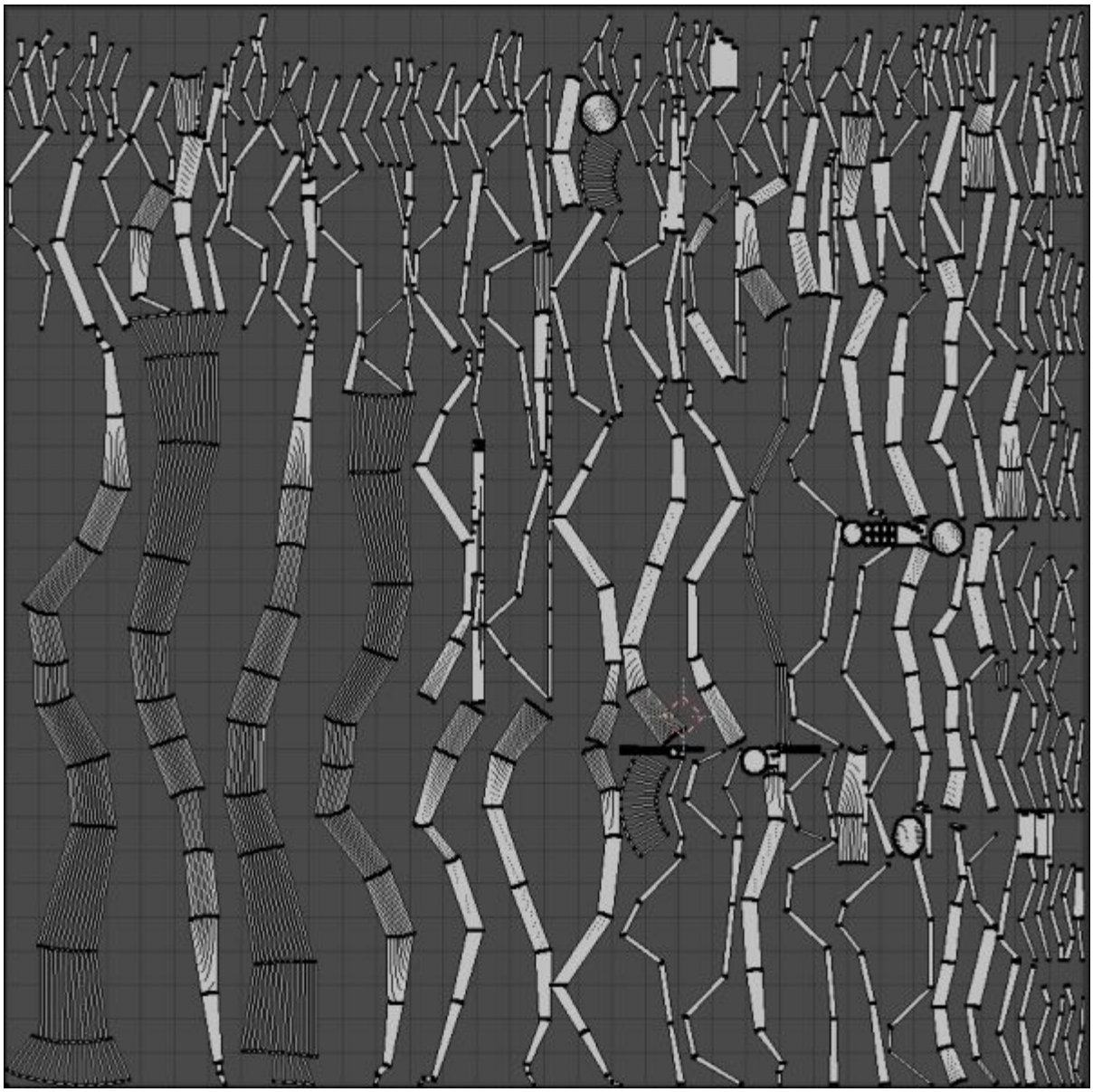
We will now learn a new unwrapping technique with the tree. The tree will be quite far in the scene, so we don't need to have perfect UVs on it. This is why we are going to use an automatic method that Blender provides called the **Smart UV Project**. This is done as follows:

1. In order to unwrap an object with the **Smart UV Project** method, we don't have to put seams on it. So, in order to UV unwrap our tree, we will need to select it entirely in the **Edit Mode**, press *U*, and select **Smart UV Project**.
2. As you can see, we now have the ability to tweak some options. The most useful is **Island Margin**, which allows us to choose how much space there is between each UV Island.



The Smart UV Project options

3. Now we can see our new UVs in the **UV/Image Editor**.



The Smart UVs of the tree

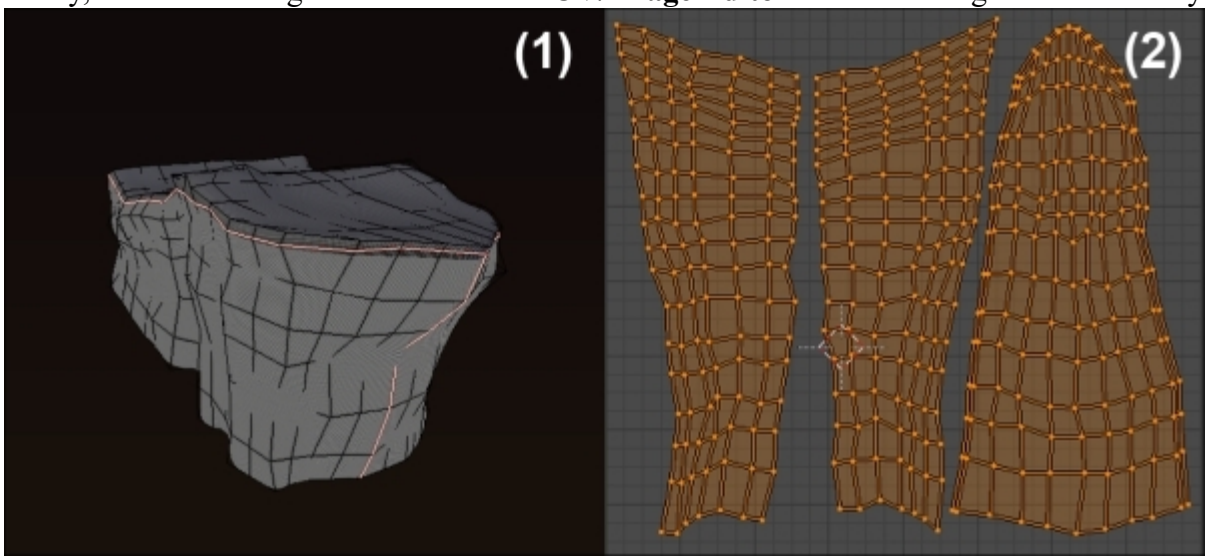
This method is quite useful but not very accurate. It generates a lot of seams that can cause visual artifacts. This is why it is only interesting for objects that are far away, small, blurred (in the depth of field), or with a small number of polygons. In the case of our tree, the wood texture will also be quite repetitive, so this will do the trick!

Unwrapping the rest of the environment

There are other methods to unwrap objects such as cube, sphere, or cylinder projections, but in our case, they won't be very useful. In order to unwrap the rest of the environment, we will use techniques that we

have already seen in the Alien project. If you don't remember how to put seams on an object, we encourage you to read Chapter 4, *Alien Character – Creating a Proper Topology and Transferring the Sculpt Details*, again. To refresh our mind, we will just do the cliff together as follows:

1. The cliff will be separated into three islands. The first seam separates the top where the ground will be from the lower section. We simply select a horizontal edge loop near the top and mark it as a seam (*Ctrl* + *E* and select **Mark seam**).
2. We will now separate the sides into two islands by placing a vertical seam from the front tip to the bottom of the cliff.
3. We will then select our whole object, press *U*, and select **Unwrap**.
4. Lastly, we can rearrange our islands in the **UV/Image Editor** in order to align them vertically.



The seams and the UVs of the cliff

Now that you are nearly a pro, you can unwrap the rock and the barrier alone by following the same method.

Tiling UVs

Now comes the interesting part! We will show you a very useful technique in order to save time with the texturing process, so we won't have to paint our objects entirely. So let's learn a little bit more about tiling.

What is tiling for?

The main goal of tiling is to allow a texture to repeat itself on the mesh. For instance, in the case of a very large wall, it would be very tedious to paint each brick one by one over the entire wall. So what is preferable to do is to use a "tile-able" texture and scale the UV islands. Scaling the islands will simply repeat the texture. Later, we will show you how to paint a tile-able texture within Blender. If you remember the UV unwrapping part of the alien project, we had mentioned that all the islands will be

placed in the UV finite space. This was because we didn't want repeated textures. But in this project, we can use this to our advantage. But it doesn't mean that we won't keep our UV islands in the finite UV space too. This is why we are going to need different UV coordinates on each object.

The UV layers

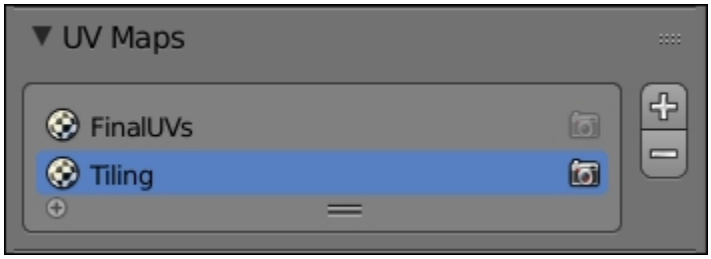
As we said before, we need to find a way to keep our proper UV (in the UV "square") and create a new set of UVs. Blender allows us to use multiple UV layers on each object. We can use a layer when we want to have another UV layout while maintaining the UV that we've already done before. It is similar to storing different UVs on the same object. We will show you how to create a new layer that will contain the scaled islands for tiling. As this will be the same for many objects, we will just show you the method for the wall of the house, and let you repeat the process for the rest of the objects:

1. We will select the wall and enter **Edit Mode**.
2. We will then click on the **Object Data** icon in the **Properties** editor, and under the **UV Maps** subpanel, we will click on the + (plus) icon. This will create a copy of the current UV set of our object.
3. We will now need to change the UVs on this particular layer. Be sure that it is selected, and in the **UV/Image Editor**, scale the current UVs with the *S* key. You can always check the tiling effect with a checker. Our wall has now two sets of UVs.

Note

The UV layers options

There are very few options to manage our different sets of UVs. The first one is the little camera icon that tells Blender which UV set should be used at render time and the second one is the ability to rename a set by double clicking on its name. The only thing that you need to think about is which set to select.



Now that you know the technique, you can add a **Tiling** UV set for each object.

Adding colors

Now that we have the proper UVs on our objects, let's dive into the fun part, that is, the texturing. It is the more artistic part of the process, so let's start by discovering the Texture Paint tool of Blender.

Basics of the Texture Paint tool

The **Texture Paint** tool is a mode that allows you to paint directly on a 3D object in the 3D Viewport while applying color to the texture. This requires having textures with a sufficiently high resolution. One of the interesting points is to paint on a 3D polygonal mesh with a low density.

To observe the paint of our textures in 2D, we need to split the 3D Viewport in two and switch the second type editor to **UV/Image Editor**.

To activate this, we first select an object, click on the **Mode** drop-down menu in the **Header**, and switch to the **Texture Paint** mode.

If you don't have any UVs, a message will warn you in the left panel (*T*) of the 3D Viewport. You can generate UVs automatically with the **Add Simple UVs** option, but it is much better to unwrap them yourself as we saw earlier.

In the **Slots** tab, there is an important parameter, that is, the **Painting Mode**. It gives a choice between two options. The **Material** option allows us to paint automatically linked textures to a material in Blender Internal. The **Image** option allows us to paint the texture without necessarily having a material linked to the object. For this first approach of the Texture Paint tool, we will be especially interested in the **Material** option.

If in **Texture viewport** Shading Mode (*Z*) your object displays a pink color and you see the message **Missing Data** in the left panel (*T*). Select **Tools**, to correct this; you will need to click on the **Add Paint Slot** option. Here, several texture types are available. This will automatically create a texture corresponding to a slot of the material with the required settings during the painting phase.

We can start testing a **Diffuse Color** map. Several options are proposed. They are the same as when we create a new texture. You can rename the texture and choose the height, the width, and the color with an alpha value. You can also choose whether you want an alpha layer (it is the opacity), the type of texture to generate, and finally, the 32-bit float option. Press **OK** to create this texture. A new material is then automatically created if there are none of them. You can visualize it in the **Material** editor on the right-hand side of the work space.

To modify this, you can change the name with a double left-click on the name.

It is possible to create several stacked textures one above the other like layers in the material. You must select the one that you want to paint in the **Slots** tab of the left panel. The bottom slot is the one that is first visible. You can also choose the Blend Type to mix pixels. There are the usual Blend Types (add, subtract, multiply, and so on.) that we can find in every decent image editing software. The **Slots** tab allows us to also change the UV layers, which can be very useful.

Now that you know the basics to generate and manage a texture for painting, we will look at the brushes.

Discovering the brushes

As in the **Sculpt Mode** that we saw previously, we have multiple brushes in order to paint our texture. They all have some specific purpose that we will test on a simple sphere object in a new scene file. Be aware that the goal here is not to do something beautiful but to test our brushes.

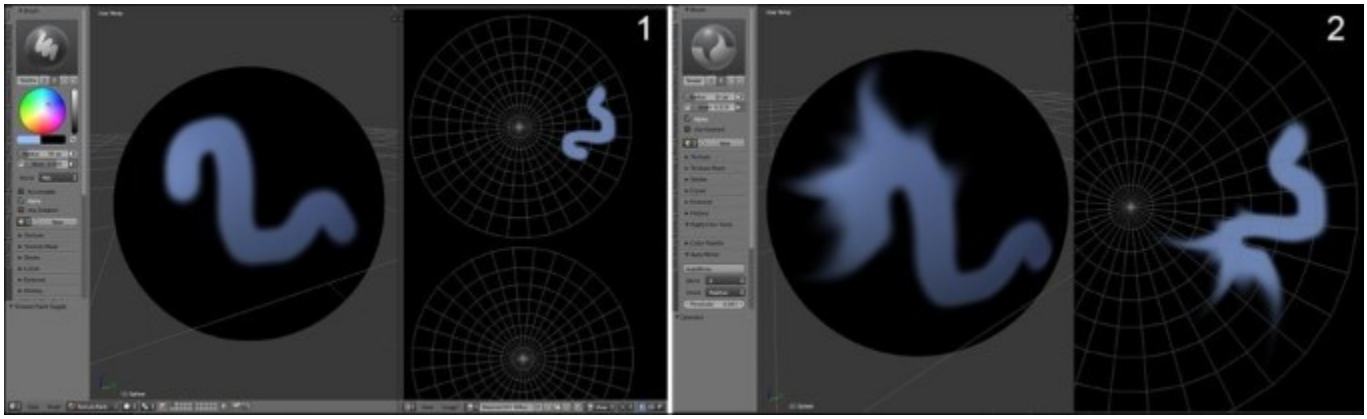
The TexDraw brush

This is the brush that allows us to paint the desired color in a localized manner.

You can use the blender mode in order to create effects. For instance, the **Add** mode is very useful for lighting texturing effect (refer to **1** on the following screenshot).

The Smear brush

The Smear brush allows us to move the color while blurring it. It is very useful to create some blown or flame painting effects. If you change the strength parameter to a higher value, you can stretch your paint to a higher distance (refer to **2** on the following screenshot).

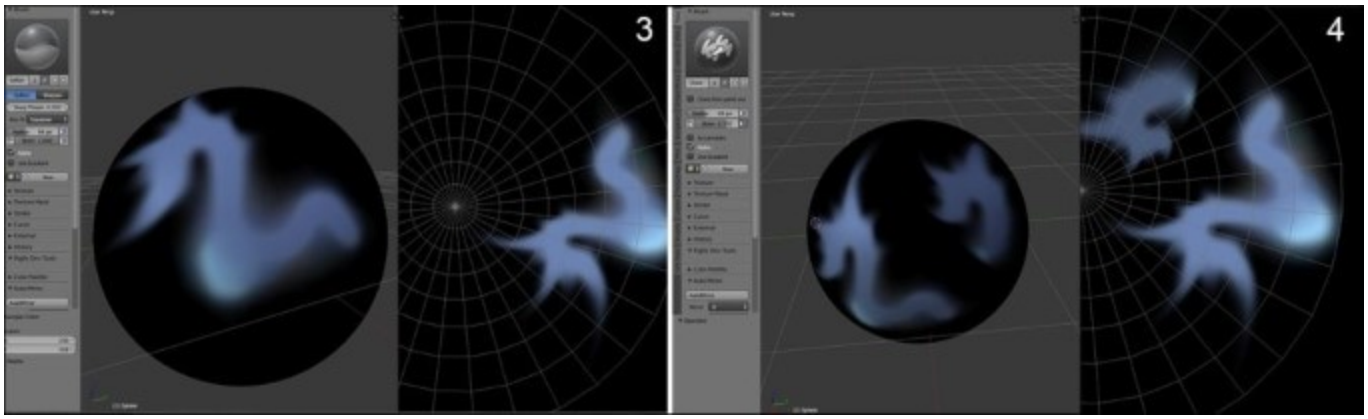


The Soften brush

This brush allows us to blur the painting. It is useful to mix the colors and create gradients (refer to **3** in the following screenshot).

The Clone brush

This brush allows you to copy a specific zone on another place. This is very useful when you need to fill some untextured space or when you want to correct the seams. You select the zone that you want to copy by placing the 3D cursor on it with *Ctrl* and LMB (refer to **4** in the following screenshot).

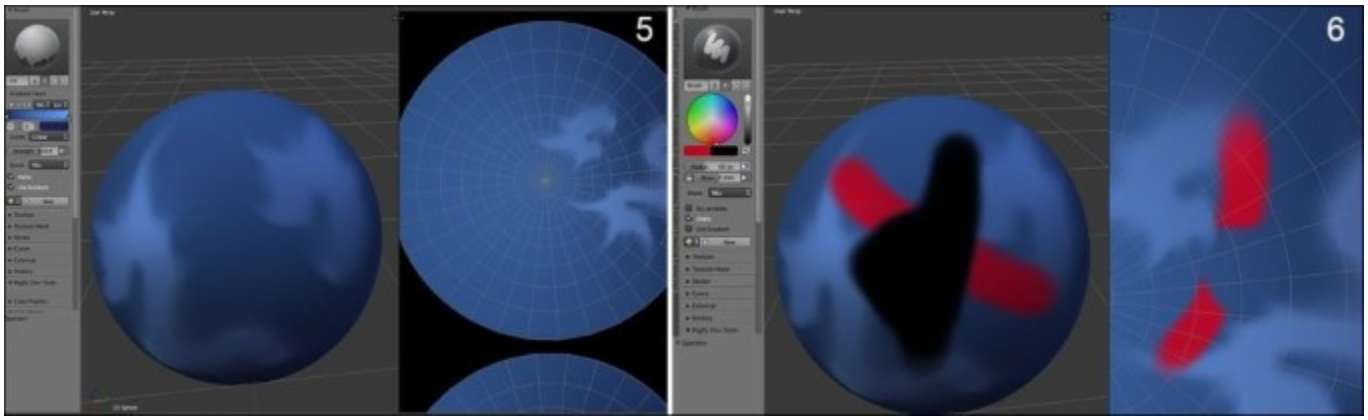


The Fill brush

This is a new brush that Blender has had since version 2.72. This brush allows us to fill the whole object with the selected color. With the **Use Gradient** option, you can do a gradient that stretches over the whole object. Remember to set the strength parameter to **1** to have a sufficient opacity. A line under the mouse cursor will inform you where the start and the end of the gradient will be. You can also use the **Multiply Blend** mode while using it (refer to **5** in the following screenshot).

The Mask brush

As with the **Sculpt Mode**, it is possible to mark a zone that you want to avoid painting. To do this, you will create a stencil image. Don't worry, Blender will ask you to create the image as soon as you create a mask, if it can't find one. You only need to click on the **New** button or select a preexisting image in the .blend file and validate the image settings like we are used to. To clear a masked part, press *Ctrl* and LMB. To remove your mask, you can remove the mask option in the **Slot** tab. Be aware that the masks are not visible in **Material Viewport Shading Mode** (refer to **6** in the following screenshot).



If you have a pen tablet, you can check out the small button on the right of the radius and strength parameters (an icon with a hand). This allows you to vary the amplitude of the parameter according to the pressure sensitivity of your stylus.

The Stroke option

The **Stroke** option allows us to completely modify the brushes' behavior. It is, therefore, important to focus on this for a little while.

First of all, there is the **Stroke Method** option that allows us to choose among several methods for applying the colors:

- **Space**: This is the basic method with a variable dot space.
- **Curve**: This is a new method since Blender 2.72 that allows us to paint in a well-defined curve with controllable points. Press *Ctrl* and left-click to create the points defining the curve. Each point can be controlled with the transform tools: Grab (*G*), Scale (*S*), and Rotate (*R*). In order to apply your painting, you need to press *Return*.
- **Line**: This method simply allows you to draw lines. You must do a pushed left-click to draw the line from one point to another. The paint is then projected onto the 3D mesh.
- **Anchored**: This allows you to drag your stroke. You first need to select the placement of the stroke or the texture you want to paint, and then, without releasing the mouse, you will be able to control its scale. This method is especially interesting when projecting a texture.
- **Airbrush**: This method could be used to project a multitude of little spots, for instance, you can change the radius so that it is smaller with **Rate** of **0.10** and **Jitter** of **2**. It is useful to create skin textures, for example.
- **Drag Dots**: This allows you to paint points or spots by placing them one by one.
- **Dots**: With a **Jitter** parameter at **0**, you get a textured line to paint. With a **Jitter** parameter at **1**, you will get a multitude of spots.

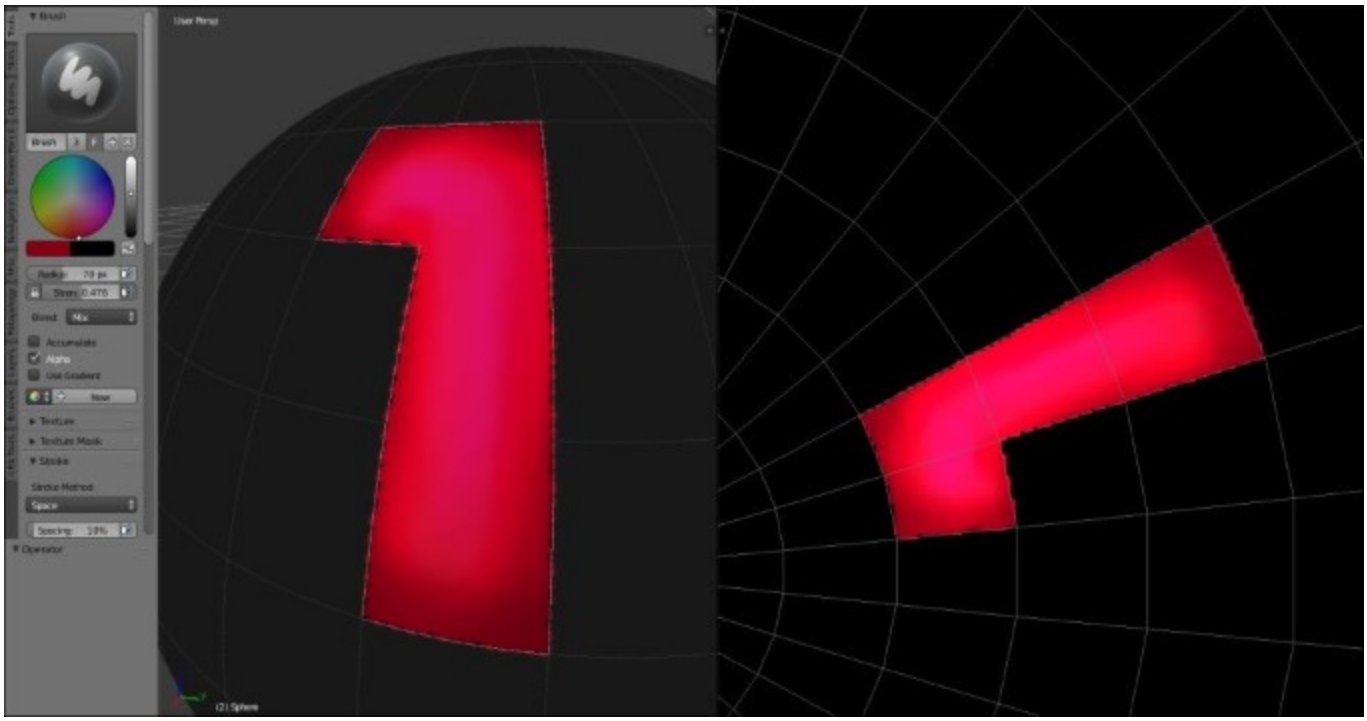
There is another key element that will determine the settings of your brush. It is the curve located just below the **Stroke** tab. It works exactly in the same manner as the **Sculpt** mode that we saw previously. Depending on whether you want a hard or thin brush to paint the details, remember to use and test several curve profiles. There are already several predefined shapes that can meet your needs.

Delimiting the zones of painting according to the geometry

So that we paint in a precise manner, it is possible to limit the zone that we want to paint by selecting polygons.

After you have selected the desired faces in the **Edit Mode**, you can go to **Texture Paint** and check the **Face Selection Masking for Painting** button on the left-hand side of the layers in the 3D View header. The icon shows a small cube with a checker pattern on a side.

You can now paint without fear of overflow.



Painting directly on the texture

If for any reason you have difficulties when painting directly on the mesh in the 3D View, you can also paint on the texture.

You simply need to select your texture in the **UV/Image Editor**, click on the **Mode** drop-down menu, and choose **Paint**. The **View Mode** is the one by default.

You have all the painting tools that you already know in the left panel (*N*). For your comfort, you can always set your view in full screen with *Shift* + *Space*.

Painting the scene

We are now ready to apply what we've learned previously about the **Texture Paint** tool on our haunted house. Let's start!

Laying down the colors

For any image that uses colors, it is necessary to lay down a color palette. This means that we will need to find the colors that will make up our image. In our case, for the house, we have chosen the following color codes:

- R: 0.259 – G: 0.208 – B:0.149
- R: 0.180 – G: 0.141 – B: 0.102
- R: 0.149 – G: 0.102 – B: 0.082
- R: 0.337 – G: 0.318 – B: 0.310

- R: 0.188 – G: 0.145 – B: 0.055
- R: 0.251 – G: 0.192 – B: 0.075
- R: 0.780 – G: 0.596 – B: 0.231
- R: 0.212 – G: 0.267 – B: 0.373
- R: 0.176 – G: 0.063 – B: 0.067

We have the ability to create a color palette by clicking on the + (plus) button near the color wheel. However, in order to have an idea of the whole color scheme, we will start by fulfilling the 3D mesh that we had unwrapped object with the colors. This is done as follows:

1. We will select one of our objects.
2. We will split our screen into two, and we will open **UV/Image Editor**, if it's not already the case.
3. In the **Edit Mode**, we will move the UV in a corner. We will keep a space for a margin, and we will not place our UVs too close to the border.
4. We will create a new diffuse map with a resolution of 4096x4096.
5. With the Fill brush, we will apply the corresponding base color for the object.
6. We will select another 3D mesh.
7. In the **Edit Mode**, we will select **Diffuse Map** for this mesh, and we will move the UV near the UVs of the previous mesh. This will be easy as you will see the previous fill.
8. Again, with the fill brush, we will apply our color for this new object.

We redo these steps for all the objects. Since our objects' UVs are proportionally scaled, their size should be sufficient in order to place the maximum number of objects on the same diffuse map.

Be careful to not select the tiling layer for the UVs while filling your objects.

Tiled textures

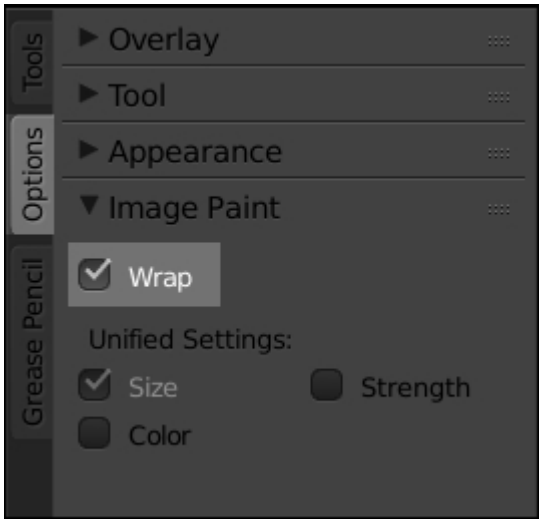
It's now time to take advantage of our tiled UVs by painting our own tiled textures by hand! In this section we are going to show you how to create the roof texture step by step, and as the process will be very similar for the wood plank, ground, brick wall, and rock texture, we will only give you some advice in order to get a nice result. So let's get started by setting up our painting environment.

The settings of our workspace

One of the strengths of the Blender painting tool is to be able to paint in the **UV/Image Editor** in such a way that the strokes that you paint repeat themselves on the borders.

1. We will first open the **UV/Image Editor**, and make it full screen by pressing *Ctrl* + *Space* while hovering over it. After that we will create a new 1024 x 1024 Texture with a greyish dark blue color (**Image** | **New Image**).
2. We now need to enter the **Paint** mode by choosing it in the header drop-down menu. By default the mode is set to **View**.
3. Now we will activate the **Wrap** option in the **Option** tab of the **UV/Image Editor** (*T*). This allows our stroke to repeat to the other side while painting on an edge.
4. Another nice feature is to check the **Repeat** option located in the Right panel (**N**) under the **Display** subpanel. It allows us to see the tiling effect.

5. Let's try this by painting on our texture with the basic brush and a different color. As you can see our strokes are repeated and we can see the tiling effect! Press *Ctrl* + *Z* to undo your testing strokes.



*The **Wrap** option in the **Options** tab*

Advice for a good tiled texture

Before starting the painting of our roof texture, we will give you some good advice that can lead you with a nice tiled texture. We first need to remember that the goal of a tiled texture is to give the impression of a pattern that repeats on a surface but in real life, even with a perfect wall pattern for instance, we can see differences between each brick. That's why we need to have a pretty homogenous texture.

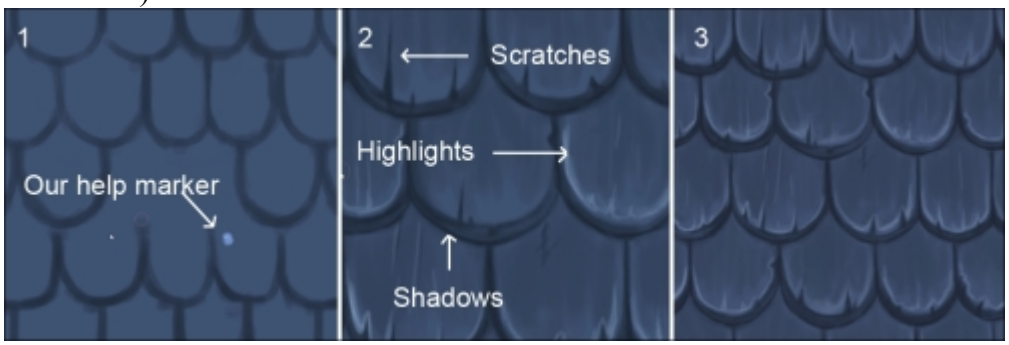
We will need to balance the contrast of our tints so they don't disturb our eyes after the tiling. Another important thing to remember is that the pattern should be repetitive in some way. We cannot paint a computer keyboard texture in a tileable manner for instance, because the keys are not the same size and don't contain the same letters. But it can work with a lot of things such as a brick wall, concrete, wood, and so on. We also need to think about the scale of the elements that compose the pattern. For instance, in the case of our roof tiles, we don't want to have one that is very small compared to the others; it will break the illusion of repetition. So now that we know the pitfalls of the tiled texture art we can start working on our roof-tiled texture.

Painting the roof-tile texture

Let's start our roof tile texture from the texture that we've created in the **UV/Image Editor** in the previous section.

1. Before starting to paint our texture we will change our curve to be a little bit pointier. We can easily select a curve preset in the **Curve** subpanel.

2. The first thing that we need to lay down is the tile pattern. In order to trace that we will use the same tint as the background color that we chose when creating the texture but darker. We then re-size our brush size (*F*) and start to paint a row of 'U' shapes for the first top tiles. We need to space them proportionally according to the size of our texture. For the next rows we will do the same thing with a little offset. The top-left part of the 'U' shape needs to touch the middle of the above ones. Note that if you are lost while having the repeat option activated, you can always help yourself with small helping markers that you can erase later (refer to **1** in the following screenshot).
3. Now that we have the basic pattern drawn, we can start to add a little bit more detail with the shadows. For this we are going to select a darker color, but still in the blue shades. Because shadows will be faded, we are going to decrease our strength a little bit (*Shift* + *F*). Here, the shadows that we are going to paint are projected because of the tiles that are placed above. This will act like contact shadows (refer to **2** in the following screenshot).
4. Now that we have our shadows we can start to add some scratches on the tip of each tile. You need to remember the fact that it needs to be as homogeneous as possible, so don't paint big scratches or it will break the tiling effect (refer to **2** in the following screenshot).
5. The last thing that we are lacking here is some highlights. When painting highlights we use a white color and decrease our strength and size. We then slightly paint the highlights where we think we have hard edges. For instance we can emphasize the scratches (refer to **2** in the following screenshot).
6. That's all, you've now completed your first hand-painted texture (refer to **3** in the following screenshot).



Steps for the roof tile texture creation

Quick tips for other kinds of hand-painted tiled textures

We aren't going to show you step by step how to do each tiled texture as it would require a lot of space and it would be a very repetitive task. Indeed when we are doing such a texture we first always create a texture with a flat color, and then lay down the pattern with a darker color. Once we are satisfied with the pattern and the way it tiles, we add the shadows, more details, and finally the highlights (the specularity).

As you can see on the wood texture, it is quite difficult to add the ribs and having a good tiling, so later we will need to take this problem into account on the objects that will receive the texture. But we can't

add ribs on wood or it will look strange. For the ground we can add a little bit more detail, such as small rocks and crackles. The bricks are quite easy to do, but if you feel you can add more detail, you can easily paint moss between each brick.

*Examples of other tiled textures painted in the **UV/Image Editor***

Baking our tiled textures

We are now going to project our tiled textures on other textures that correspond to the UVs of our different objects.

Why bake?

As we saw it with the Alien character, texture baking is very useful in order to capture relief, shadow, or color information. In the case of our haunted house, we are going to capture the color information of the tiled textures in order to have them on a large texture with the proper UVs. This lets us achieve our tiled patterns on one big map in order to add all the tweaks that we want later on. We could for instance paint the window contact shadows, add some grunge, and age our objects.

In our scene we aren't going to bake everything. So some objects are still going to use their tiling UV layer. It will simplify our work and still leave us with a nice result.

How to do it?

To obtain a successful bake, the manipulations will be quite similar to what we've done with the normal and ambient occlusion map of the alien. We will start by doing the baking of the walls.

1. We select our walls joined as one object and we go into **Edit Mode**.
2. In the UV Map subpanel under the **Data** tab we click on the camera icon on the right of the Tiling layer. This will tell Blender to use this layer for the render and baking process. We then select the first layer in order to project the details on it with the proper UVs that are normalized.
3. Still being in **Edit Mode**, we create a new map that we call **Color_walls_01** with a 4096 x 4096 resolution. We also un-check the **Alpha** checkbox. This image will contain the result of our bake.
4. We now go to the **Bake** tab from the **Render** tab.
5. Under the **Bake** button we select **Texture** as the **Bake Mode.**

6. In the **Margin** option we choose **10px.**
7. Repeat those steps with the objects that share the same UV space.
8. We can now click on **Bake** in order to start the process. Voilà! Your baked texture is now ready to be placed as a diffuse color texture on a material.

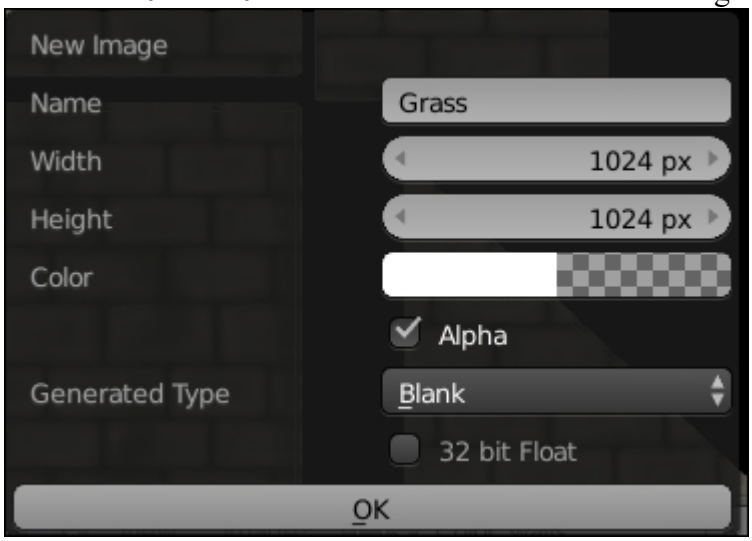
Creating transparent textures

One thing we haven't learnt until now is how we can produce texture with an alpha channel. Indeed this could be very useful in order to add some details on the previously baked texture (grunge or leaks, for instance) or even grass.

The grass texture

Usually, when doing grass, fur, or hair, we use the integrated particle hair system of Blender, but in our case we will show you a technique that can do the job as well and can save us render time. It will also accommodate very well with the style of our scene. This technique will simply consists of a plane mesh on which some grass strands will be projected; using the alpha we will be able to just render the strands. Note that this is a very common technique in the video game industry. So let's start our grass texture by first setting up our transparent texture!

1. We will first go in the **UV/Image Editor** and create a new texture. In the color setting of the texture, we will change the alpha channel to **0** in order to have a full transparent image. We then leave the 1024 x 1024 resolution and validate our settings.

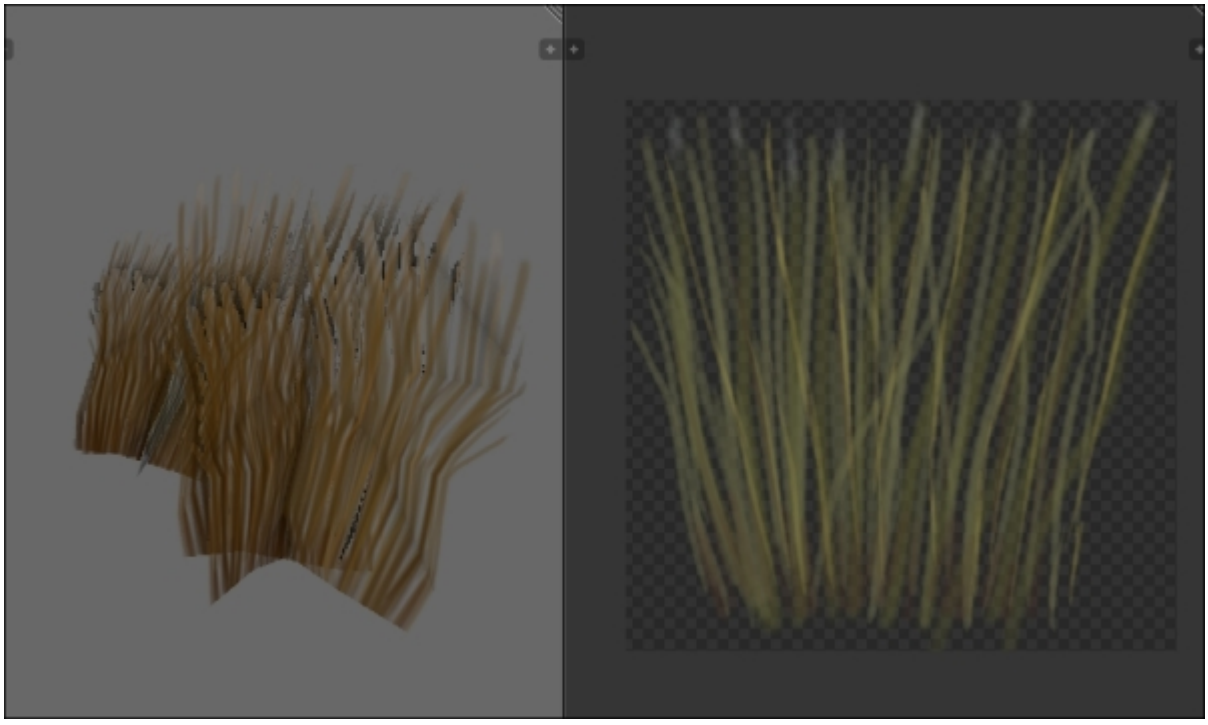


The grass texture settings

2. We can now use a pointy curve and start to paint some strands starting from the bottom of the texture to the top with a de-saturated green. We really need to think about painting a dense grass mound.
3. In order to add more realism to our grass texture, we can add some touch of yellow. We also need to add some white on the tip of each strand. It is quite important to use a reference when

painting some textures; it helps to develop our sense of perspective and come with more believable results.

4. Remember to save your texture on your hard drive or you will lose it (**Image** | **Save as image**)!
5. We can now place our texture on a new plane (*Shift* + *A*). We do a quick UV on it by simply pressing *U* and selecting **Unwrap** in the **Edit Mode**.
6. We will now create a new material that will use our texture. To do that, go to the **Material** icon in the **Properties** editor and press the plus icon. If you already have a default material, you can delete it with the minus icon.
7. So our material can understand the alpha channel, we will have to activate the check box of the **Transparency** subpanel and select the **Z-Transparency** mode with its Alpha value set to **0**.
8. Now we will tell our material to use our grass texture. To do this we click on the texture icon of the **Properties** editor and click on the first available texture slot. Under the **Image** subpanel we click on the far left drop-down menu and select our grass texture. The last thing we need to do is to activate and set the **Alpha** slider under the **Influence** subpanel to **1.0**.
9. We can have a preview of our texture in the 3D Viewport by activating the GLSL mode in the right panel of the 3D View (*N*) under **Shading**. Note that you will need to be in the **Texture** Shading mode (located under the **Viewport shading** drop-down menu in the 3D View header) and you also need to have lights.
10. We can now duplicate the plane object as an instance to have more grass. Note that you can also add a subdivision to the plane and in the last tool options you can change the F**ractal** slider in order to add a little bit of randomness. Remember that the render in the viewport is a preview, not the final render.



*The final grass texture in the viewport (left) and in the **UV/Image Editor** (right)*

Note

More about the color wheel window

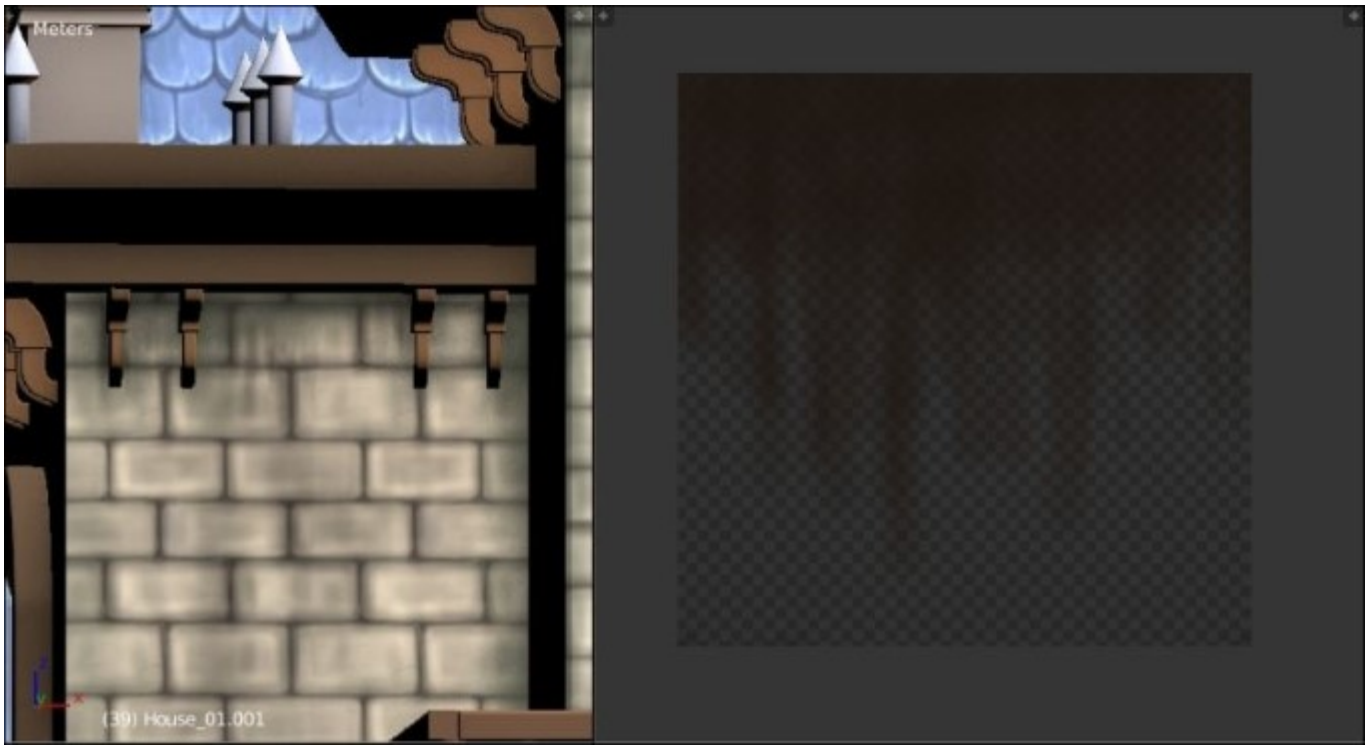
When selecting a color in Blender we have many options. You can of course select the color that you want with the color circle or by changing the slider's values. In **RGB Mode** we can act on each red, green, and blue component plus on the Alpha channel. In **HSV Mode** we can change the hue (the tint), the saturation, and the value of the color. If you put the saturation down to **0** the color will be on a gray scale. The **Hex Mode** allows you to type a hexadecimal value such as FFFFFF (white) or FF0000 (red). Hexadecimal simply means that instead of counting from 0 to 9 we count from 0 to F. It represents 16 possible values. The easy thing to remember when dealing with hexadecimal colors is that the first two digits represent the Red value, the next two digits represent the green, and the last two represent the Blue: RR GG BB. FF is the full color, 00 means no color. For instance 00FF00 is full green.

The grunge texture

The grunge texture will be useful in order to add details on the wall texture of the house. The technical process is the same as the grass texture. For the painting we simply use a dark brown color and paint some vertical leaks from the top to the middle of the texture.

Now we can stamp this texture on our wall.

1. We select the wall and ensure that its baked texture is selected in the **UV/Image Editor** while being in **Edit Mode**. Another thing you may want to do if you are still in **GLSL** mode is to create a new material with the baked texture set.
2. In order to paint our leaks we will use the **Anchored** stroke method located in the **Stroke** subpanel. It allows us to precisely place our leaks near the top and the bottom of the wall.



*The grunge placed on the house in the viewport (on the left) and the grunge in the **UV/Image Editor** (on the right)*

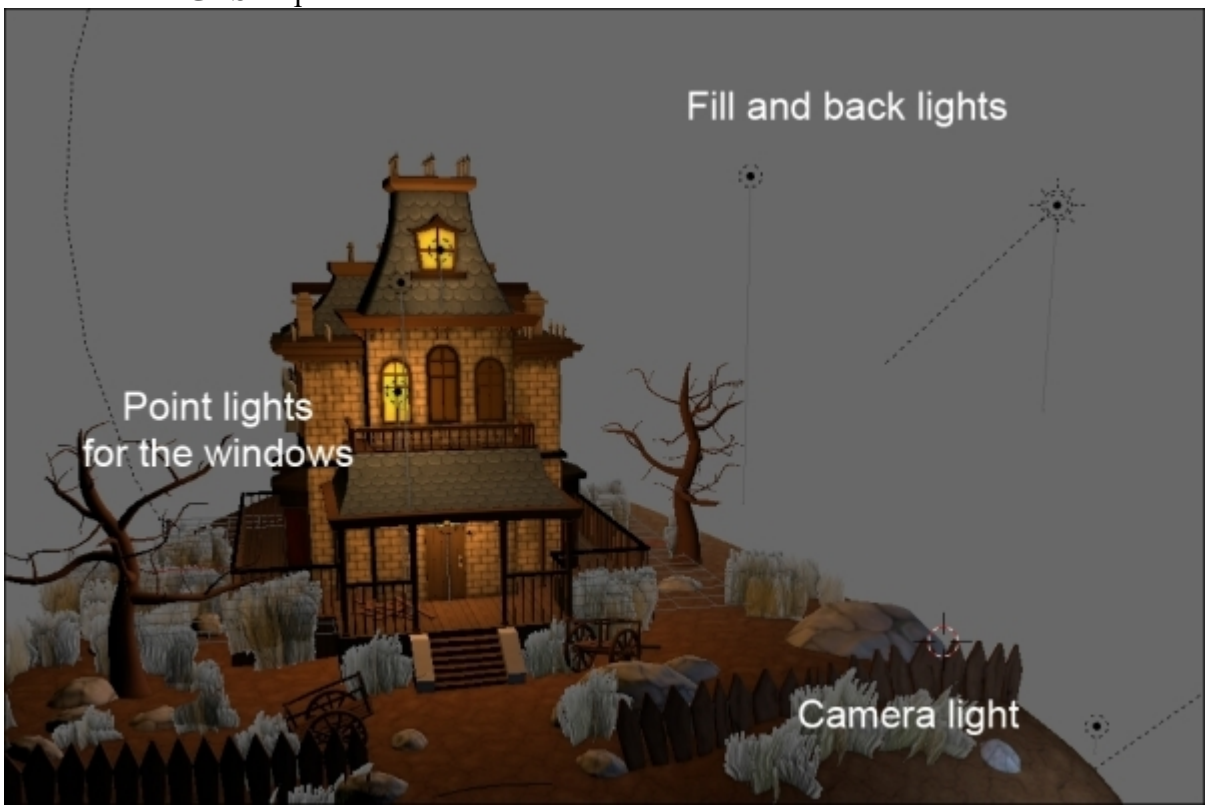
Doing a quick render with Blender Internal

Welcome to the bonus section! Here we are going to do a quick render with the Blender Internal render engine to get an idea of what the whole scene will look like. Note that this is totally optional as we are going to create a render with cycles later. In order to do our render, we will need to add a material for each object in the scene like we did for the walls but with their corresponding textures.

Setting lights

Now that we have all our materials created we will need to turn up the lights! Of course if you don't have light you won't see anything like in the real world.

1. We add a sun lamp (press *Shift* + *A* and select **Lamp** | **Sun**) and change its color to a grayish yellow in the light settings situated in the **Properties** editor. We leave its energy to a value of **1**. We place it behind the house and rotate it so it hits the back of the house.
2. The next light we will add is point light. This one is going to be much more intense so we bump up the energy value to **20** and also change its color to a light yellow. This one will fill the scene a little bit more.
3. Now we need to add lights behind the house to fake the lighting of the windows. In total we added three yellowish point lights with a value of **10** for their energy.
4. The last point light we can add will be near the camera, so we would have to place it correctly after choosing our point of view. This light allows us to see a little bit more of the close environment.
5. If you want to see the lighting effect in the viewport, it is best to turn off the **Texture Shading** mode and the **GLSL** option.



The final light placement

Placing the camera

We can now choose our point of view, by moving our camera. If you don't have a camera, you can add one (press *Shift* + *A* and select **Camera**). You can also add many cameras and switch between them by selecting the one you want to look through and pressing *Ctrl* + *P*. In order to look through your camera in the viewport you can press the *0* numpad key. Usually it's a good habit to change the focal length of your camera in the Camera setting tab in the **Properties** editor (click on the Camera icon). In our case we wanted a fleeting camera so we set a focal length of 20mm.

Setting the environment (sky and mist)

Now in order to improve our render we will change the sky color and add a mist.

1. To do that we go into the world settings in the **Properties** editor (click on the earth icon).
2. Under the World subpanel we check the **Paper sky** and **Blend sky** option; we change the **Horizon color** to a dark brown color and the **Zenith color** to an even darker color. It's not realistic, but this image is not intended to be realistic.
3. We can now check the **Ambient Occlusion** checkbox and set it in the **Multiply** mode.
4. The **Environment Lighting** option will serves to light the scene with the **Sky Color** option (and not **White**, the default option).
5. Lastly we can activate the **Mist** subpanel and change the **Depth** parameter to **30m**. We also set the start option to **0**. You may have to tweak those values in order to match your own scene.
6. Now you can press the **Render** button in the **Render** tab of the Properties editor or press *F12*. Note that if you want to improve your render with some automatic compositing you can go to the **Scene** tab of the **Properties** editor and under **Color Management** you can use a look preset; you won't have to re-render your scene! You can also tweak **Exposure**, **Gamma** and change the **CRGB** (Contrast, Red, Green and Blue) curve by clicking on the **Curve** checkbox. As you can see, you render is displayed in a **UV/Image Editor**, so you can save your image (**Image** | **Save Image as**). Congratulations, you've done your first render of the haunted house!

The world settings

The final haunted house after the Blender Internal render will look like the following screenshot:

Summary

In this chapter you completed the UVs and the textures of the scene. You have learned a technique to paint textures by hand with the Texture Paint tool of Blender and how to create and use tileable textures. You also learned more about baking. Now that we have introduced the rendering process with the Blender Internal render engine, we can start to learn more about the other render engine called Cycles, which has different approach. So let's dive into Cycles!

Chapter 7. Haunted House – Adding Materials and Lights in Cycles

This chapter will be devoted to the Cycles render engine. You will learn how to achieve a convincing render of the haunted house by understanding the different types of light work and by creating complex materials using the previously made textures. You will learn some nice tricks such as how to produce normal maps of our hand-painted textures without leaving Blender or how to create realistic-looking grass. You will also discover how to use the Cycles baking tool. In order to conclude our project, we will show you how to integrate a mist effect in the final composition.

In this chapter, we will cover the following topics:

- Understanding the essential settings of Cycles
- Using lights
- Painting and using an Image Base Lighting
- Creating basic materials with nodes
- Using procedural textures
- Baking textures in Cycles for real-time rendering

Understanding the basic settings of Cycles

To switch to the Cycles render engine, you must select it in the list of proposed engines that Blender offers in the menu bar. We will see in the first part of this chapter some of the very useful settings that should be known while using Cycles.

The sampling

If you directly try to make a render with Cycles without changing the parameters of Blender, you will certainly see some noise in the image. To make this less visible, one of the first things to do is change the sampling settings. Unlike Blender Internal, Cycles is a Raytracer Engine. While rendering, Cycles will send rays from the camera in order to generate pixels. The noise is due to a small amount of the samples. Cycles, therefore, needs more samples; the more sampling, the more accurate the final render.

The following sampling settings are in Properties editor. Just select **Render** | **Sampling**:

- **Render samples**: This is the number of samples for your renders. The more samples you add, the longer the rendering time will be.
- **Preview samples**: This is the number of samples Blender will calculate to preview your scene in the 3D viewport in Rendered Viewport Shading. A value between 20 and 50 samples is correct. This value depends on the performance of your computer.

The Render sample must be higher than the Preview sample.

Note

The GPU device

If you have a fairly recent CUDA®-compatible graphic card, you can opt for GPU rendering. This allows you to make renders very quickly and visualize your scene in the 3D viewport nearly in real time.

For this, go to **User Preferences** | **System** | **Computer Device** and select **CUDA**. Then, in **Properties**, go to **Render** | **Device** and select **GPU**.

Note that the most recent AMD GPU has been supported since Blender 2.75.

Clamp direct and indirect

This allows us to clamp the intensity of the rays of light launched from the camera. This can also help to reduce the noise effect, but it blurs the pixels together.

Light path settings

You will find the following light path settings in the menu:

- **Max and Min Bounces**: This is the number of minimum and maximum bounces a ray of light can do in the scene to render a pixel. This mimics the way photons bounce from objects in real life. The higher the value of the maximum bounce, the greater the precision will be, and the quality of the rendering will increase. A high value of minimum bounce will also improve the quality but may considerably increase the rendering time. It is advisable to set the same minimum and maximum value.
- **Filter Glossy**: This will blur glossy reflections and reduce the noise effect. You can put a 1.0 value.
- **Reflective and refractive caustics**: Caustics are light effects related to transparent and reflexive materials. We can observe this light effect with diamonds, for instance. This uses a lot of resources and generates some noise. By unchecking these two options, you can completely turn off these effects during rendering. In the case of our haunted house scene, we are going to disable them.

Performances

You will find the following performance settings in the menu:

- **Viewport BVH Type**: There are two types of this: **Dynamic** and **Static**. This is a way to let Cycles remember some of its rendering calculations for the next render. The **Static** option is highly recommended to optimize the rendering time, if you have no more polygonal modifications in your scene. Otherwise, you can use the **Dynamic** mode.
- **The Tiles**: This helps to manage the pixel groups that are to be rendered. So you can control their size along the *x* and *y* axes and also control the method to render the image (if you prefer to start rendering through the center of the image, or from left to right, and so on). The size of the pixel group to be rendered should be chosen according to your machine settings. If you are rendering with your CPU, you can choose a smaller tile size than if you were rendering with your GPU.

Note

For more information, you can have a look at the official Blender manual at these addresses:

http://www.blender.org/manual/render/cycles/settings/integrator.html

http://wiki.blender.org/index.php/Dev:2.6/Source/Render

Lighting

We are now going to look at a very important aspect of the rendering process: the lighting. Without lights, you won't see any objects, as in the real world. Good lighting can be hard to achieve, but it can give a nice atmosphere to the scene. One of the things that is true with a scene with good light is that you won't even notice the lights as they look like natural lighting. In order to get our lighting job done correctly, we are going to add a basic shader to every object in our scene.

Creating a testing material

Let's add a very basic material with Cycles in order to see the effects of the lights. In order to better understand the lights in the next section, we are going to create a blank scene and test our lights on a cube that is laid on a plane:

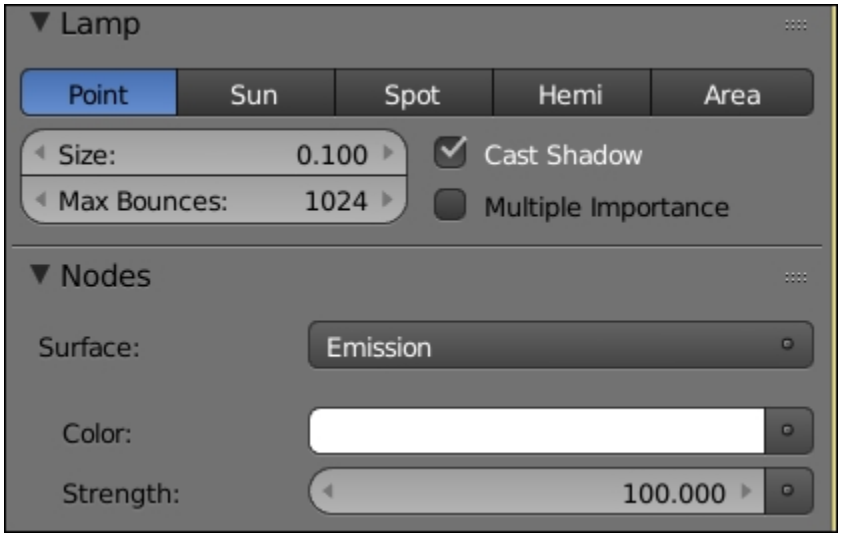
1. Let's start by creating a new scene and by adding a plane scaled ten times (**S > 10**) under the default cube.
2. We also want to delete the default light and turn the Cycles render engine on.
3. We aren't going to explore material creation in depth for now, so we advise you to follow these steps in order; later you will have more information about this process. We need to select the cube and open the **Material** tab of the **Properties** editor.
4. Now we will create a new material slot by clicking on the **New** button.
5. Under the **Surface** subpanel, we will click on the color and change its value to **1.0** in order to have a full white color. That's all for the material. It will be a handy material to test the different types of light.
6. The last thing we need to do is to add the same shader to the plane. To do this, we can copy the material of the cube. We will first select the object (or objects) to which we want to copy the material, in our case, the plane, and then we will select the object that owns the material that we want to copy. Then we will press *Ctrl* + *L* and select **Material**. The plane should now have the same white material.

Understanding the different types of light

The goal of this section is to understand how each type of light can affect our objects in a scene. We will give you a brief explanation and a short preview of what effects they can provide you with:

1. Before our tests, we will need to change the world shader that contributes to the lighting of the scene. If you press *Shift* + *Z* or change the **Viewport** Shading mode to **Rendered** in the 3D View header, you can see what the scene will look like, but you will see it in real-time in the viewport. Here you can clearly see the objects even if we don't have any lights. That's because the background color acts as if there was an ambient lighting.
2. If we go into the world settings of the Properties editor and change the Surface color to black, we will not see anything.
3. We can now add a light and have a better understanding of its effect without being disturbed by the world shader.

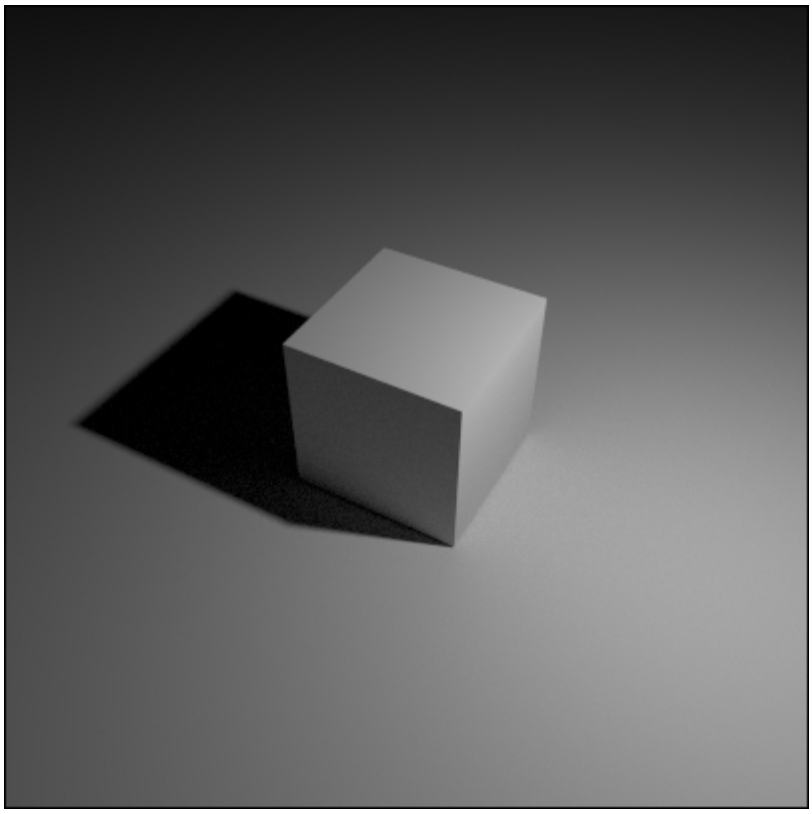
The settings of the lights can be found under the **Object Data** tab of the Properties editor (a yellow dot with a ray icon) while the light is selected. There are five types of lights (but four work in Cycles) that have many options in common: **Size** influences the hardness of the shadows they produce on objects and **Max Bounces** tells Blender the maximum number of bounces the light rays can travel. They also have the ability to cast shadows with the **Cast Shadow** checkbox turned on. Of course, they also have a strength that you can tweak in the **Nodes** subpanel. Note that, if you can't see the strength option, you need to click the **Use Node** button.



The shared light options

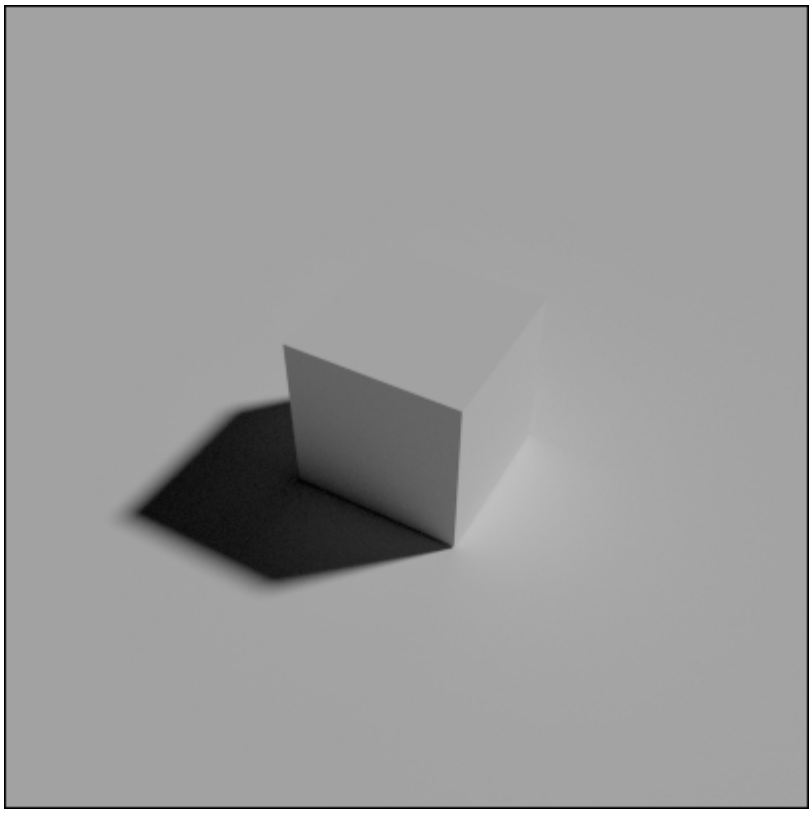
The different types of light are as follows:

- **Point**: As its name implies, it emits lights according to its position. It emits light rays in all directions, but these rays are limited to a certain distance from the center of the light. We often call it a spherical lamp.



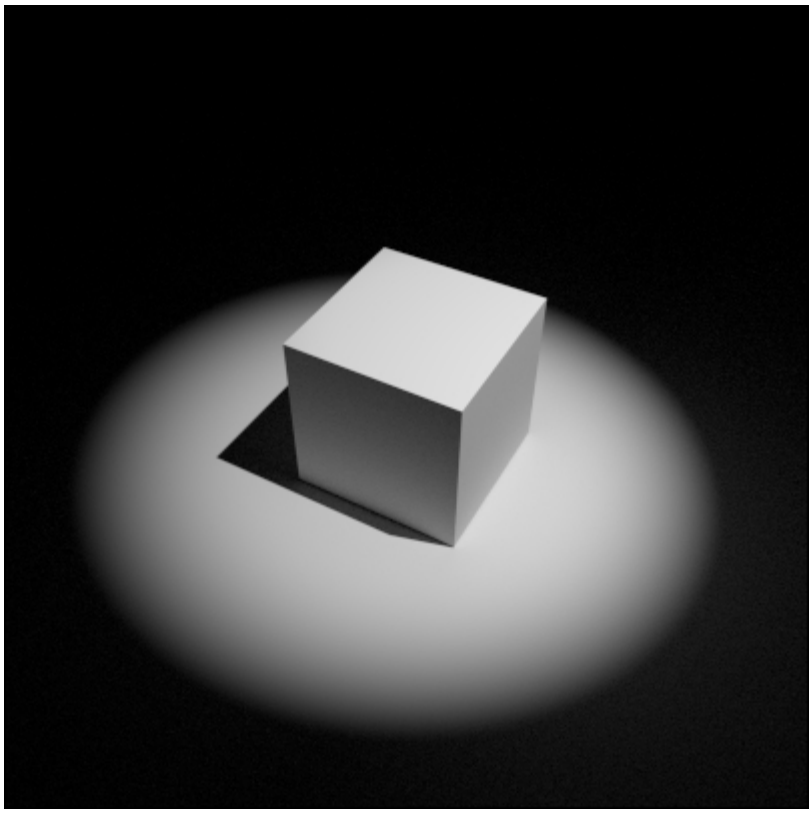
A point light with a strength of 500 and a size of 0.1

- **Sun**: As you can imagine, this type of light represents the way the sun works. As the sun is very far away from earth, we can admit the way we perceive its rays as parallel. So in Blender, the sun produces parallel rays, and we also don't care about its location, but only its orientation matters. With a small size, you can quickly represent the lighting of a bright day. You will mostly use it as a global light as it lights the entire scene. Also, notice that its strength should be less than the point light that we saw previously.



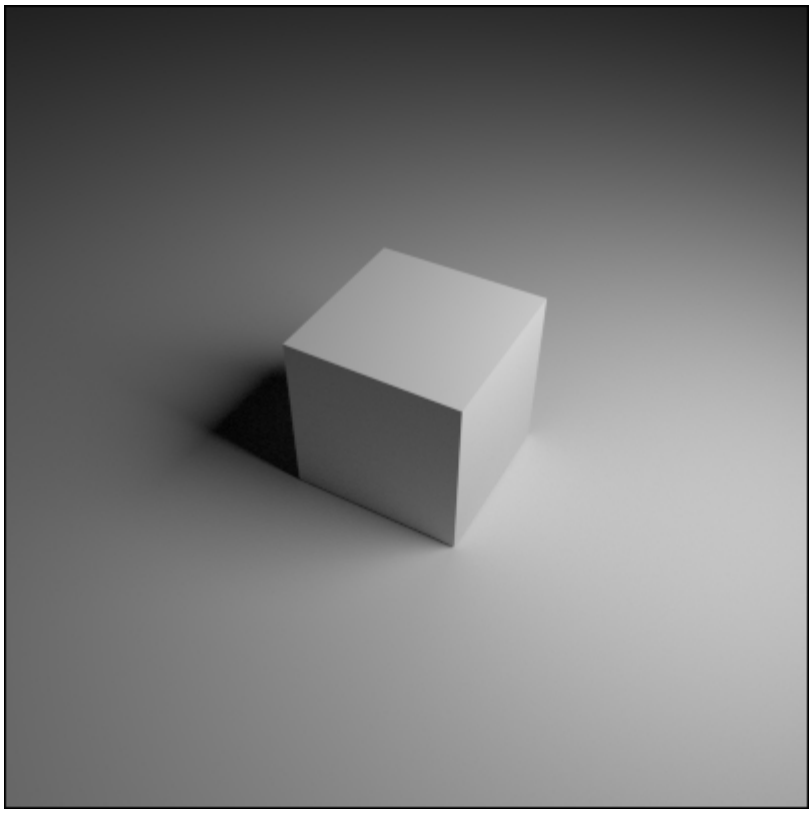
A 45-degree angle on Y and Z sun light with a strength of 2 and a size of 0.05

- **Spot**: The spot light is a conic light. It looks like the lamps used on stage in order to light the show presenter. The lighting that it will produce depends on its location, direction, and the spot shape. You can change its shape in the **Spot Shape** subpanel, and **size** will determine the size of the circle of light influence. The **blend** will define the hardness of the circle shadow. The circle size and the strength of the spot light will depend on its distance from the objects.



A 45-degree angle on a Y spot light with a strength of 5000, a size of 0.5, a shape size of 30 degree, and a blend of 0.8

- **Hemi**: For now, the Hemi type of light is not supported in Cycles. If you use it, it should react like a sun lamp.
- **Area**: This is one of the more common lights. It emits light rays from a plane according to a direction represented by a dotted line. It could be squared or rectangular. Its size, like for the other types, will affect the hardness of the shadows. The strength of the light will also depend on its distance from the objects. With this light, you can achieve a very precise lighting, so we strongly advise you to test this in order to be familiar with the way it reacts.

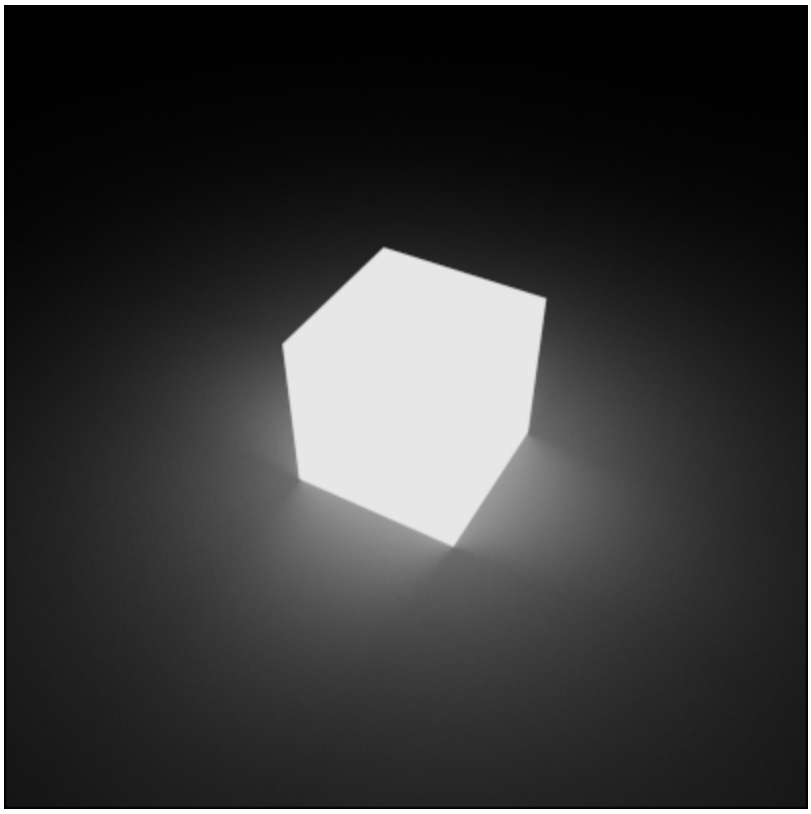


An area light with a strength of 500 and a square size of 5

Another option that many Blender users appreciate is using an emission shader to act as a light on an object (a plane, for instance). The emission shader is, in fact, the base shader of the other types of lights.

1. First let's add a plane.
2. Then, under the **Material** tab of the **Properties** editor, we will add a new material slot.
3. Change the diffuse surface shader from **Diffuse BSDF** to **Emission**.
4. As you may notice, if the plane is in the camera field, you can see it. We don't want this, so go into the **Object** tab of the **Properties** editor, and under the **Ray visibility** subpanel, uncheck **Camera**.

You may find this easier, but, in fact, with this method we lose a lot of control. The main problem we've found with this method is that we can't control the way the rays are emitted, for instance, with the area light. This can be useful when you want visible objects to emit light, but this is not very good for precise lighting.



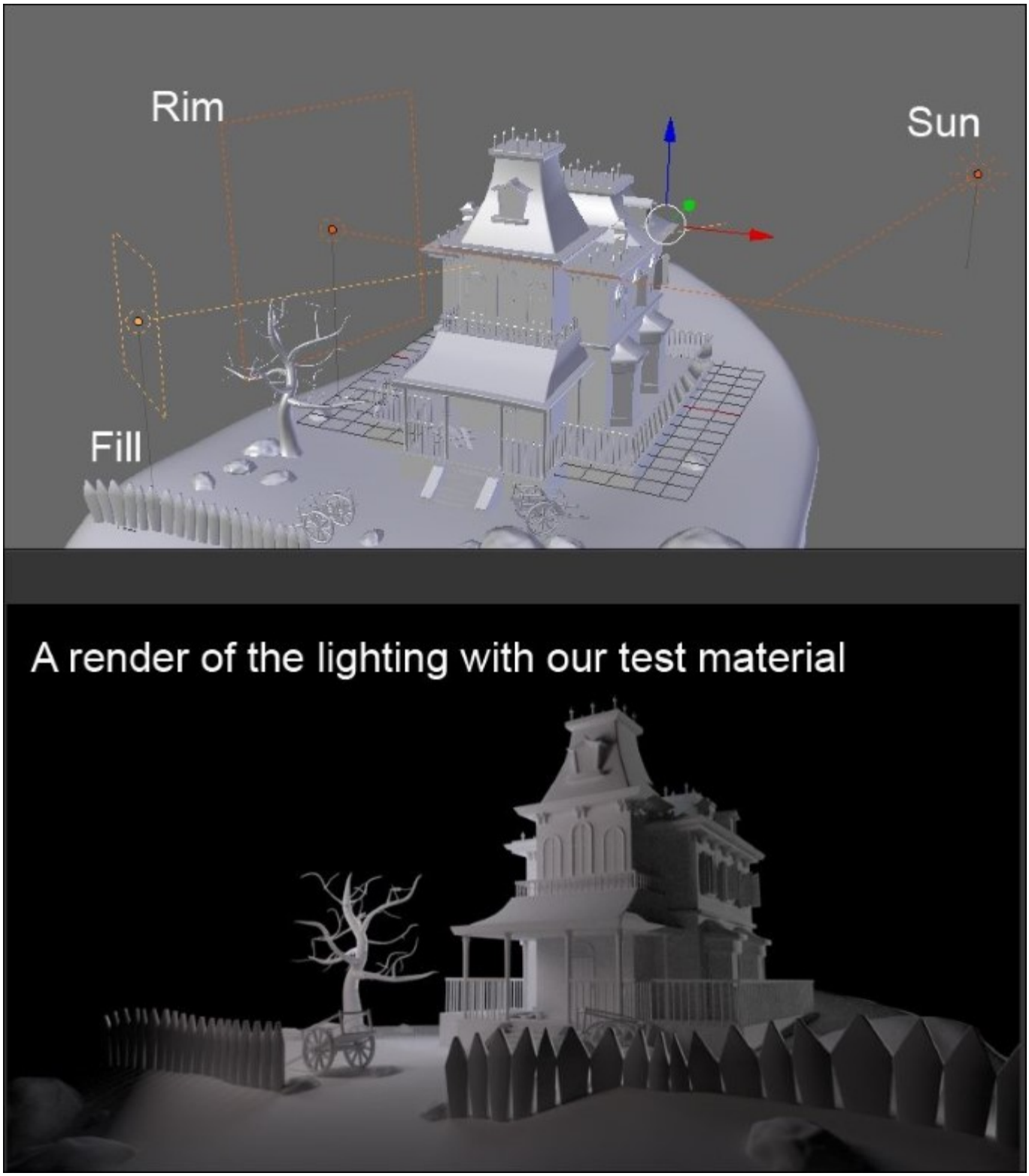
The cube with an emission shader

Lighting our scene

As you can see from the previous part, there are many types of light that we can manipulate in order to achieve a nice lighting effect. But in the case of our haunted house, we are only going to use area and sun lights because the other types of light are often used in specific situations.

1. We will start by opening our haunted house scene and saving it in another name (`HauntedHouseCyles.blend`, for instance).
2. Now we can delete all the lights that we used in the Blender Internal render.
3. While doing a lighting effect, it's a good to have an idea of the volume of your objects with a neutral material. So we will select one of the objects in the scene, remove its existing material, and create a new material in the **Material** tab of the Properties editor. As we did for our testing scene, we will change the color value to **1.0**.
4. Rename the material as `Clay`.
5. Now we will select all the objects in scene (*A*), and reselect the object that is the clay material while pressing *Shift* in order to make it the active object.
6. Press *Ctrl* + *L* and select **Material**. All objects will now share the same material.
7. We can now split our interface in two. One of the 3D views will display the camera point of the view (the *0* numpad key) and will be rendered in real time (*Shift* + *Z*) with a preview sampling of 50 (decrease it if you don't have a powerful computer).

8. The first light to be added is the sun. We will orient it, so it lights the right-hand side of the house. It will be nearly horizontal. Our goal here is to have a dawn lighting. The sun has a size of 5 mm in order to have harsh shadows and a strength of 1.0.
9. The next light that we will add will fill up the front of the house a little bit. It will be an area light that is a slightly tilted down and located on the front left side of the house. We want smooth shadows, so we will change its size to 5 m. Its intensity will be around 400. We can also change its color to be a little bit yellowish.
10. The last light will act as rim light. It will be an area light that comes from the back of the house on the left-hand side. Its size will be 10 m and its strength around 700. We have also tinted it a little bit towards blue.



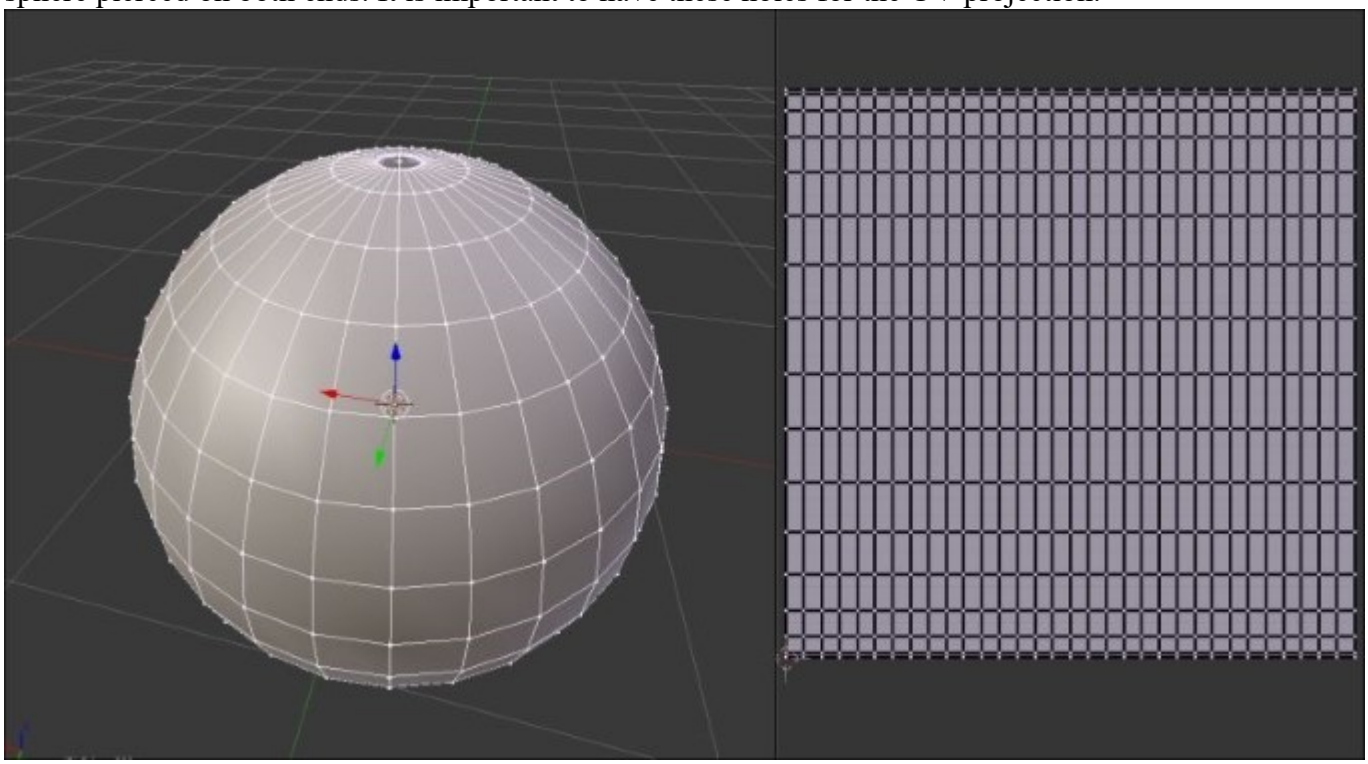
The lighting of the haunted house scene

11. That's all for the basic lights. The light settings are in constant evolution during the whole image creation pipeline, so don't be afraid to change them according to your needs later. Note that we are missing an environment lighting that we are going to set up in the next part of the chapter.

Painting and using an Image Base Lighting

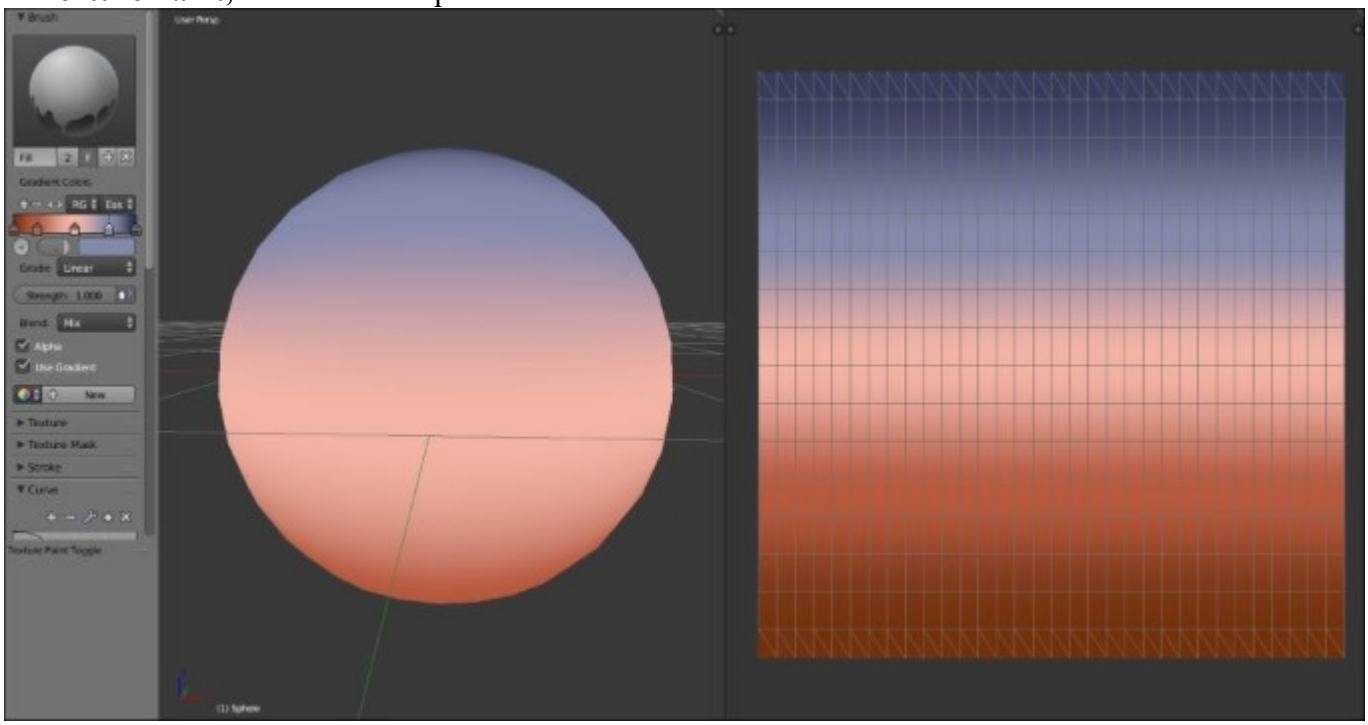
An **Image Base Lighting** (**IBL**) is a very convenient technique that allows us to use the hue and the light intensity of an image to lighten up a 3D scene. This can be a picture of a real place taken with a camera. HDR images provide very realistic results and may be enough to light a 3D scene, but for our haunted house scene, we will paint it directly in Blender with Texture Paint. This technique allows us to do complex lighting in less time and will enrich the lighting that we have prepared previously. We will start by seeing how to prepare the painting phase of a customized IBL:

1. We will open a new scene in Blender.
2. We will split the working environment in half with a **UV/Image Editor** on the right-hand side and a 3D View on the left-hand side.
3. We will add a UV Sphere at the center of the world (*Shift* + *A* and select **Mesh** | **UV Sphere**) on which we will paint the sky.
4. We will delete the vertices at the two poles of the sphere. We will make a scale extrusion (*E* and *S*) of the edges and slightly reposition them again for a well-rounded look. We will obtain a sphere pierced on both ends. It is important to have these holes for the UV projection.



5. We will select all the polygons of the sphere (*A*), then we will apply **Unwrap Cylinder Projection** (*U*). In the **Cylinder Projection** options on the left panel (*T*) of the 3D Viewport, we will change the **Direction** parameter by selecting the **Align to Object** option. This allows us to get straight UVs that occupy the most space on the entire UV Square.

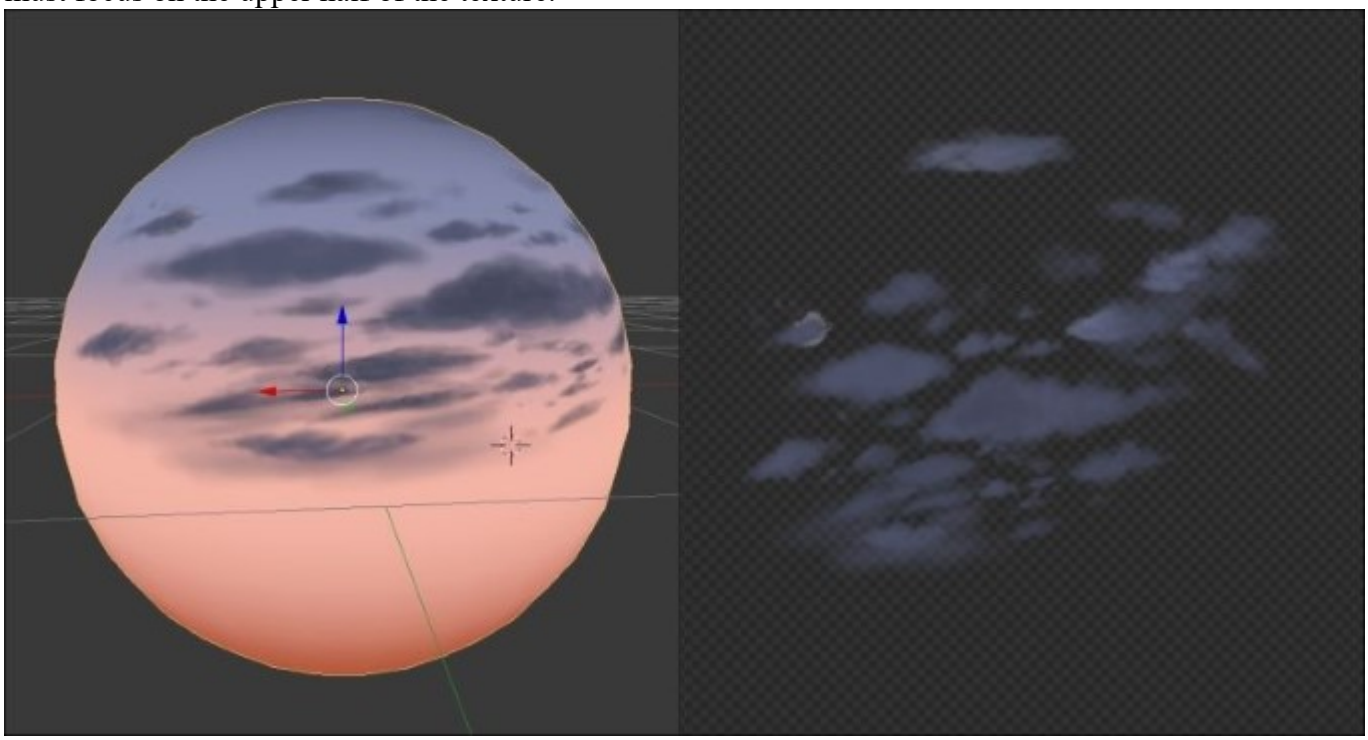
6. Once the UVs are created, we will select the edge loops that form the holes at the poles of the sphere, and we will merge them each in turn to form a complete sphere. This will form triangles in the UV, but it does not matter.
7. We will again select all the polygons of our sphere, and we will then add a new texture by clicking on the **+ New** button in the **UV Image** Editor.
8. In Blender Internal Renderer mode, we will create a new material on which we will place our IBL texture. To better visualize the texture, we will check the **Shadeless** option (**Material** | **Shading** | **Shadeless**).
9. In **Texture Paint,** we can start to paint.



10. We will use Fill Brush with the **Use Gradient** option in order to prepare the gradients of the sky. We will use the following Gradient Colors from left to right. The color marker number **1** on the far left is **R: 0.173**, **G: 0.030**, and **B: 0.003**. The color marker number **2** is located at **0.18**, and its color is **R to 0.481**, **G to 0.101**, and **B to 0.048**. The color marker number **3** is at position **0.5**, and its color is **R to 0.903**, **G to 0.456, and B to 0.375**. The color marker number **4** is located at **0.78**, and its color is **R to 0.232**, **G to 0.254,** and **B to 0.411,** and the last marker at the far right is the color **R: 0.027, G: 0.032,** and **B: 0.085**. We will then apply the gradient upwards on the sphere.
11. On the texture in the **UV/Image** Editor, we can see some black near the poles, which may interfere with the lighting calculation. Thus, in the **Paint Mode**, we will take the nearest color of the black triangles by pressing the *S* shortcut (without clicking), and we will fill the black triangles with **Fill Brush** (without Gradient).
12. When this is finished, we can save this image as **IBL_Sky**.
13. In the **Texture Paint** mode, in the **Slots** tab, we will add a **Diffuse Color** texture that will be transparent this time. For this, we will check the **Alpha** box, and we will change the alpha value

of the **default fill colo**r to 1 in the **Texture Creation** menu. This will allow us to create clouds on another texture while keeping our sky visible.

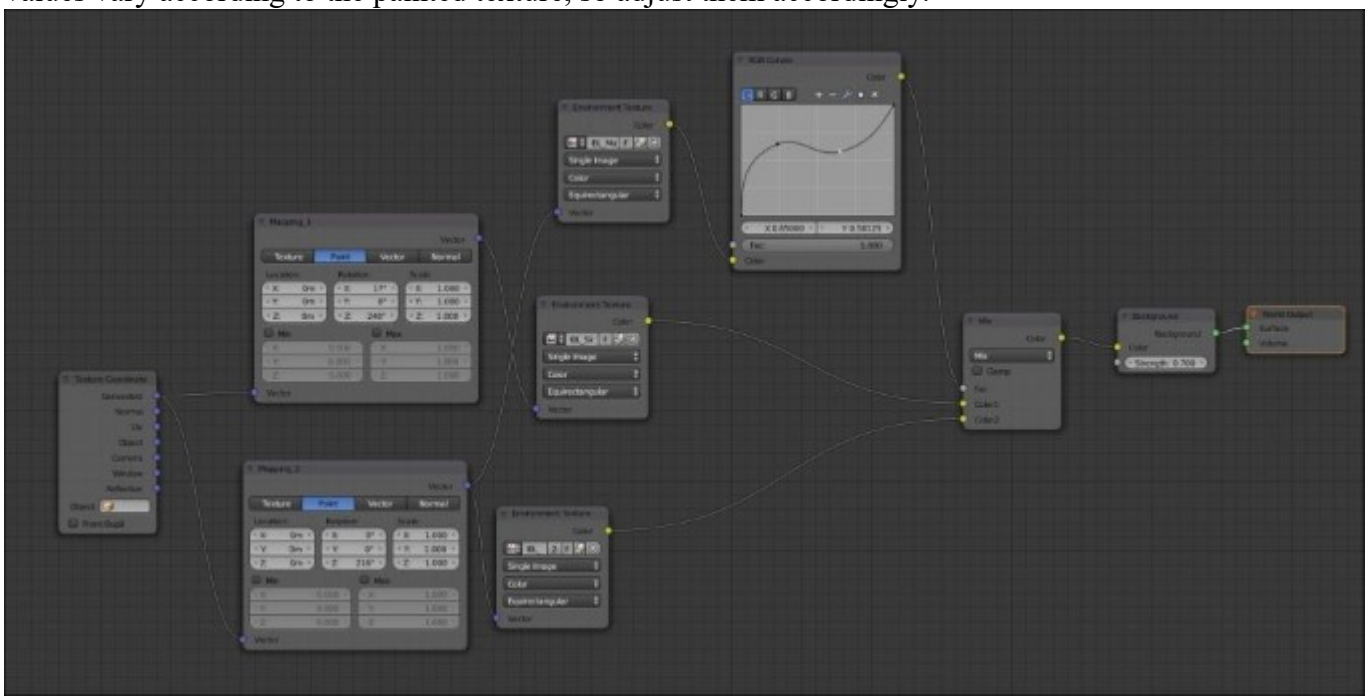
14. With the **TexDraw** brush and the **R: 0.644, G: 0.271, B: 0,420** color, we will draw a few clouds. We must think that there will be only the upper half that will be displayed on the framing of the haunted house. This part will have the greatest importance for the lighting. So we must focus on the upper half of the texture.



15. We will save the image as **IBL_Cloud**.
16. From this cloud texture, we will make a mask that allows us to properly mix the sky and the clouds. For this, we must save our image, with the **BW** (Black and White) option and not in RGBA, by naming it as **IBL_Mas**k.
17. We will then return to the haunted house scene, and in the **Node Editor**, we will click on the **World** icon that is represented by an earth, and we will check the **Use Node** option.
18. We have two nodes that appear: **Background** and **World Output**. We will add an Environment Texture node (press *Shift* + *A* and select **Texture** | **Environment Texture**).
19. We will duplicate the **Environment Texture** node twice (*Shift* + *D*). We will place them one above the other and to the left.
20. In each **Environment Texture** node, we will open the IBL textures created previously.
21. We will add a **Mix RGB** node (press *Shift* + *A* and select **Color** | **Mix RGB**) that will allow us to mix our textures. We will connect the **IBL_Sky** Color Texture Image Output socket to the **Color1** input socket of **Mix Shader**, the **IBL_Cloud** Texture Image Color Output socket to the **Color2** input socket of **Mix Shader,** and the **IBL_Mask** Color Output Socket to the **Fac** input socket. We will keep **Mix** as the Blending mode.
22. We will add a **Mapping** node (press *Shift* + *A* and select **Vector** | **Mapping**) that we will duplicate once, and we will position them one above the other on the left-hand side of the **Environment Textures** node. We will rename them as **Mapping_1** and **Mapping_2**. We will

connect **Mapping_1** to the **IBL_Sky** Texture Image node and **Mapping_2** to the two other **Environment Textures**.

23. We will add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) that we will position at the left. We will connect the **Generated** socket of the **Texture Coordinate** node to the **Vector Input** socket of the two **Mapping** nodes
24. To get a better contrast for the **IBL_Mask**, we will place a **RGB Curves** node (*Shift* + *A* and select **Color** | **RGB Curves**) between the **Environment Texture** node and **Mix RGB**. We will set the following two points: the first point at the **X= 0.24** and **Y= 0.65** position, and the second point at the **X= 0.65** and **Y= 0.58** position.
25. We have all the necessary nodes. To finish, we will need to modify the mapping of the **IBL_Sky** texture. Therefore, we will modify the **X= 6°**, **Y= 23.9°**, and **Z= 0°** rotation parameters. These values vary according to the painted texture, so adjust them accordingly.



To visualize your **Image Base Lighting** (**IBL**) better, you can display it in the viewport. In the **Solid** mode, in the right panel of the 3D Viewport, check the **World Background** option (**Display** | **World Background**).

Creating materials with nodes

It's now time to discover the material creation process with Cycles. In this section, we are going to create the basic shaders that are composed of our previously painted textures. The shaders won't be at their final stage here. Later, we are going to improve them with normal maps.

Creating the materials of the house, the rocks, and the tree

Let's start with the wall shader of the house:

1. We will first select the corresponding object.
2. We are going to duplicate the clay shader that we had added in order to test our lighting in the previous section. As you can see, it is used by 68 objects in the scene. If you click on the 68 button on the right-hand side of the material name in the material tab of the Properties editor, you will duplicate the shader and make it unique. At this time, we can now rename it as `HouseWall`.
3. We are now going to switch to the **Node Editor** in order to have more control on our shader. In fact, we can do everything in the Properties editor, but it will be quite hard to manage with a complex shader. So open a new editor and change it to a **Node Editor**.
4. As you can see, we already have **Diffuse BSDF** plugged into the **Surface** input of the **Material Output** node.
5. A diffuse shader has no shine on it. It looks flat. In real life, every surface is at least a little specular, so we are going to mix our diffuse shader with another shader that will bring us the shiny effect. To do this, we will first add a **Glossy BSDF** shader (press *Shift* + *A* and select **Shader** | **Glossy BSDF**) and place it under the diffuse shader. Don't connect it for the moment.
6. In order to mix the two shaders together, we will use a **Mix Shader** node (*Shift* + *A* and select **Shader** | **Mix Shader**). As you can see, this node has two shader inputs. Plug the BSDF output (green dot) of the diffuse shader to the first shader input of the **Mix** shader, and the BSDF output of the Glossy shader to the second shader input of the **Mix** shader. Now plug the **Mix** shader output to the surface input of the **Material Output** node. As you can see, both shaders have been mixed together. You can now use the **Factor** slider in order to choose which one is predominating. If you put a value of **0**, you will only use the shader connected to the first input (the diffuse), and if you put a value of 1, you will only use the shader connected to the second input (the glossy one).
7. The blend between these shaders is not going to look right with any value, so we are going to connect a **Fresnel** node to the **fac** input (press *Shift* + *A* and select **Input** | **Fresnel**).

   ### Note

   About the Fac input

   The role of the fac input is to control how both shaders will be mixed. Usually, the **Fac** input needs to be fed with black and white information where the amount of black tells us how much the first shader will be used, and the amount of white tells us how much the second shader will be used for the final output.

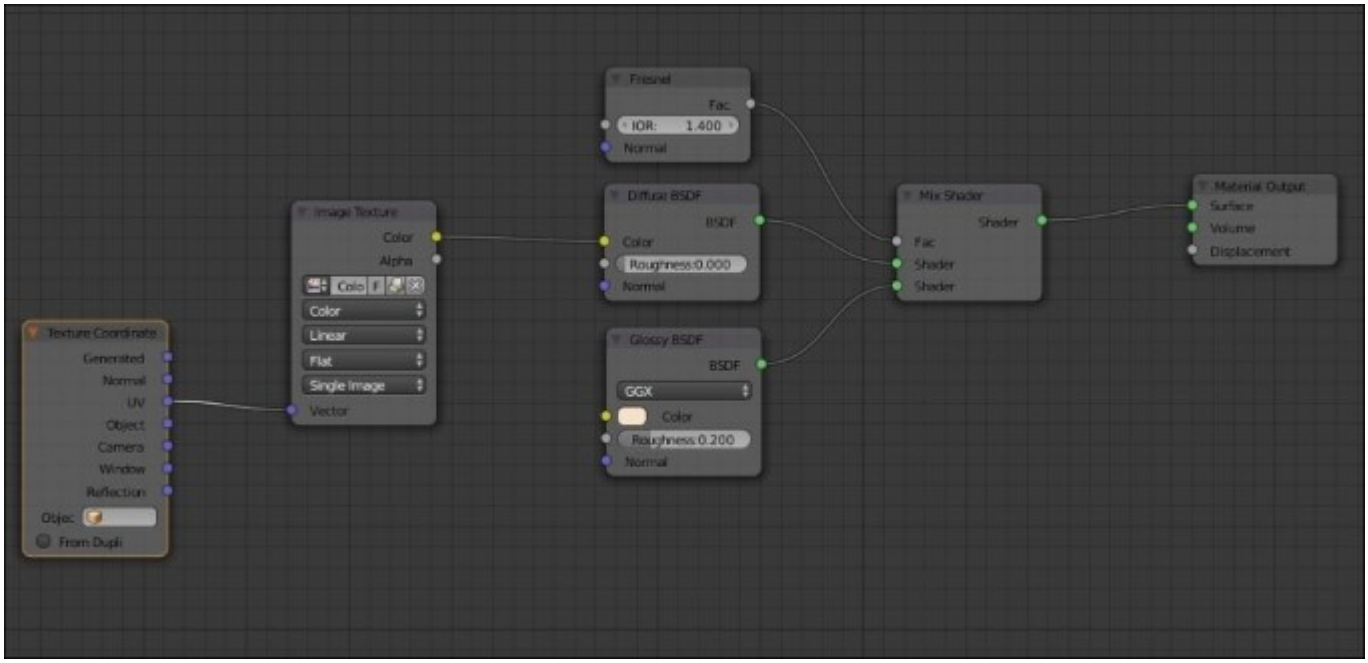
8. We can now change the value of the Fresnel to **1.4**.

Note

About the Fresnel

The Fresnel node will produce a black and white texture according to the volume of the geometry. It will be calculated according to the light ray's incidence. We usually use a Fresnel node in order to catch the highlights better. You can use a pretty interesting add-on called node wrangler that allows you to quickly see the result of each node without shadows. In order to use it, right-click on the node you want to see while pressing *Ctrl* and *Shift*.

9. We will now change the glossy color to a yellowish tint. Don't forget to turn on real-time shading in order to have a preview of what this will look like in the render. You can also drag a rectangle to the zone you want to preview with the *Shift* + *B* shortcut while being in camera view. If you want to remove the rectangle zone, drag a new zone to the outer zone of the camera in the camera view.



The base of our wall shader

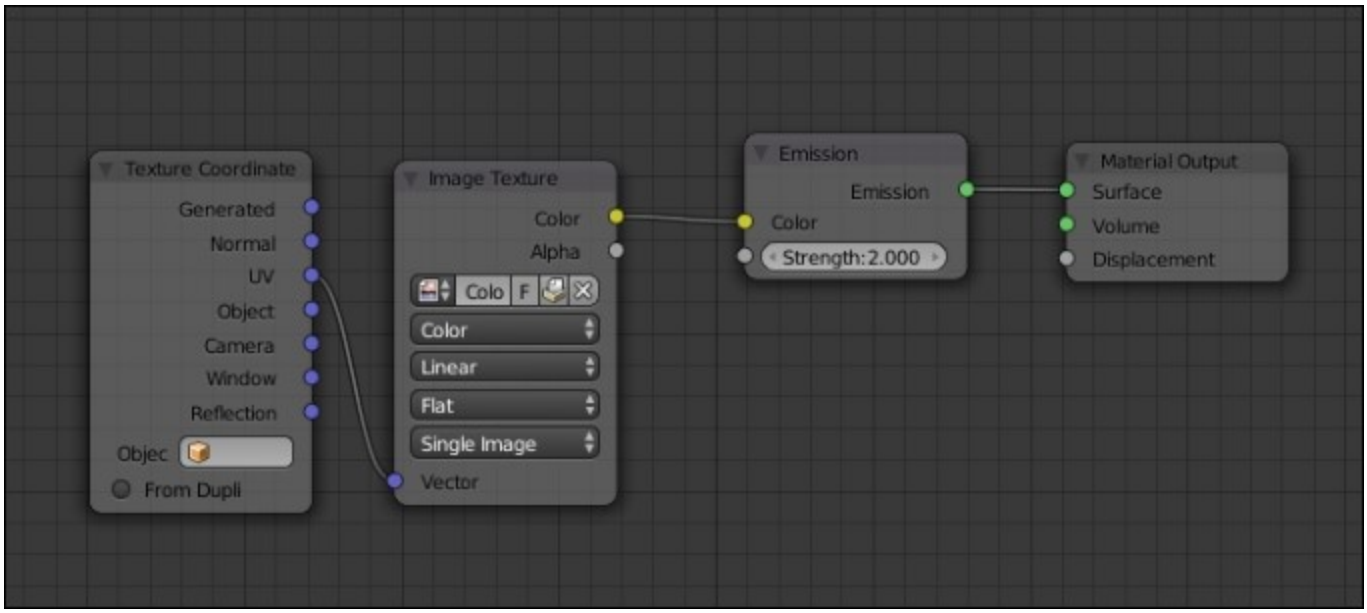
The last thing we will do with this material for now is plug it into our texture:

1. We will add a new Image Texture node (press *Shift* + *A* and select **Texture** | **Image Texture**).
2. We will need to connect the **Color** output of this node to the **Diffuse BSDF** color input.
3. It's also a good idea to add a **Texture Coordinate** (press *Shift* + *A* and select **Input** | **Texture Coordinate**) node and plug the UV slot to the **Vector** input of the **Image Texture** node.

By default, the Vector inputs are set to be UV, but with this node, we can clearly see the mapping method used for the textures.

1. We can now select one of the roofs and change its clay shader to the one we created because the roof shader will be nearly the same. Now, in order to break the link between the roof and the wall shader, we can press the button with the number of objects that share the same material in order to copy the material.
2. We will rename this shader to `HouseRoof1`.
3. The only thing we need to do for now is to change the texture of the Image Texture node to the corresponding roof texture.
4. We can now select all the objects that need to share the same material (the other blue roofs), and finally, we select the roof that has the shader that we want to share, press *Ctrl* + *L*, and select **Material**.
5. We will now repeat the process of creating a new material by copying it from the previous one, changing its name and texture information, and linking it to its corresponding objects.

We will now have a shader on the rocks, the tree, and all the different objects that make up the house. The only shader that will be different is the one on the top window. It needs to emit light. In order to do this, we will copy the previous material, delete the diffuse, glossy, and mix shader (*X* or *Delete*), replace them with an **Emission** shader (press *Shift* + *A* and select **Shader** | **Emission**), and plug the window texture to this. Now the top light is going to emit light! You can tweak the emission value if you want more light. As you can see, we need to do this for the other windows as well, but in the case of the other windows, we can't do this simply because they are not planes and their color information is located on the wall texture. That's why we need to paint a mask.

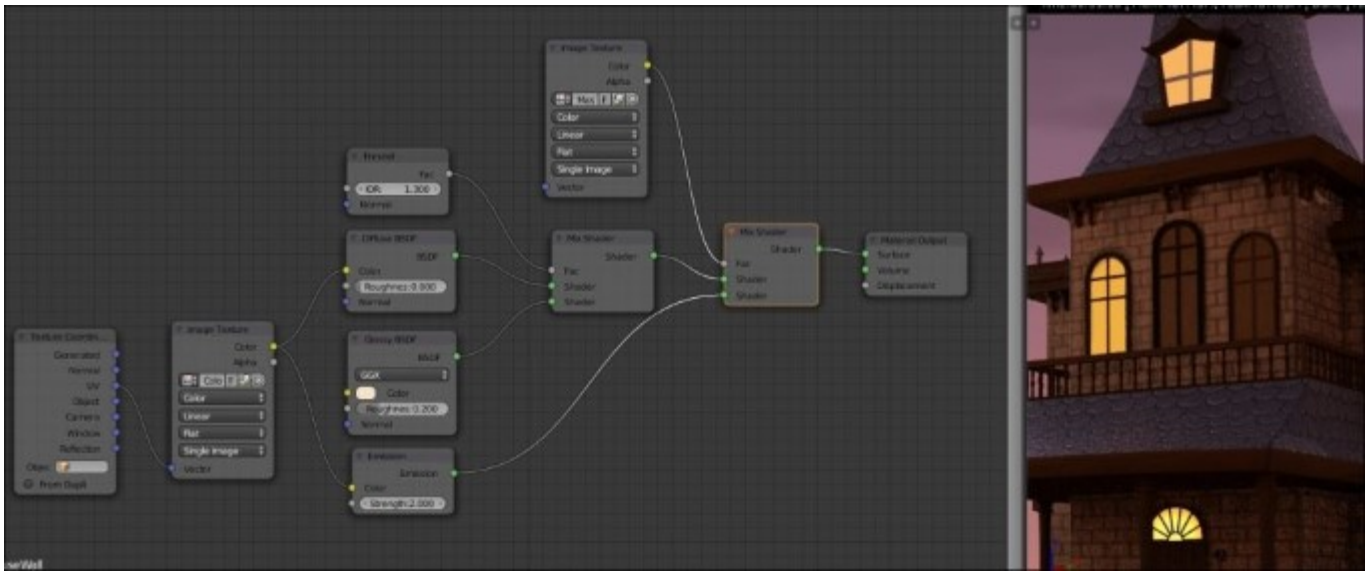


The top window shader

Adding a mask for the windows

We are now going to improve our wall material by creating a mask that will separate the windows that are shining from the rest. These windows are going to be painted white and the rest will be black. So when we plug the mask in the **Fac** input of a Mix material node, we will be able to choose an emission shader for the white parts.

1. We are going to paint our map in the **Blender Internal** context. Note that we can actually use the **Texture Paint** mode while being in Cycles, but this implies that we add and select a texture node that uses the texture that we want to paint.
2. So we will select the wall object, and in the **UV Image** editor, we will create a new 1024 x 1014 black texture. Usually, masks don't need large resolutions.
3. Now in pure white paint the windows that shine (the light yellow ones on the color map).
4. Let's go back to our wall material and add a mix shader just before the output node. If you want to save time, you can drag the node on the connection line of the previous **Mix** shader and the **Output** node. This will automatically do the connections for you.
5. The first shader input is already used by our old **Mix** shader. Now we are going to add a new **Texture** node with our mask and plug its output to the **Fac** input of our new **Mix** shader.
6. The second shader input will be fed with an **Emission** shader (press *Shift* + *A* and select **Shader | Emission**). In the **Color** input of this node, we will plug our color map (the same as in **Diffuse** shader).



The wall material with the mask on the left-hand side and the result in the real-time rendered 3D view.

7. Now we can increase the strength of the Emission shader to 2.0. As you can see, now our shining windows (and only them) emit light!

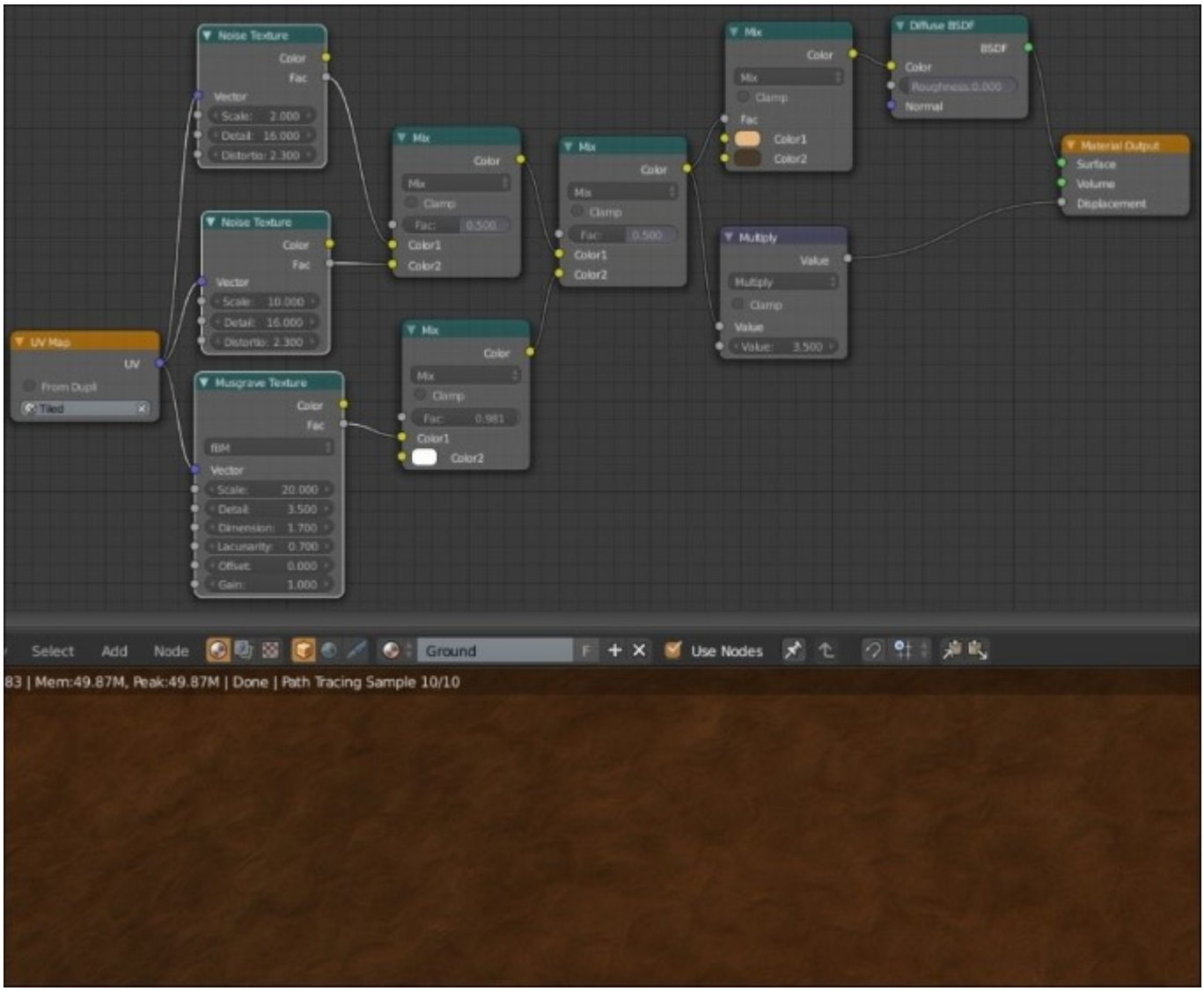
Using procedural textures

One thing that could be very interesting when creating materials is generating their textures procedurally. In this render, we are going to replace the hand-painted ground by a procedural material. This is done as follows:

1. We will select the cliff and create a new material for it.
2. The next thing is to add a **Diffuse BSDF** node. The color input of this diffuse material will be fed with a mix of procedural textures.
3. Let's add a Noise texture node (*Shift* + *A* and select **Textures** | **Noise**) and duplicate it (*Shift* + *D*). The first one will have a scale of 2.0, and the second one will have a scale of 10.0. Our goal is to have a mix of both levels of noise. Both of them will use the Tiled UV layer, so add a **UV Map** node (press *Shift* + *A* and select **Input** | **UV Map**), select the correct map, and feed the **Vector** input of the noise textures with the **UV Map** node output.
4. As these nodes are textures and not shaders, we aren't going to use the **Mix** Shader node but the **MixRGB** node instead. So we will add one of these nodes (press *Shift* + *A* and select **Color** | **MixRGB**), and feed the inputs with their noise **Fac** output. Don't use the color output as we want a black and white mix here. Remember that you can always test your results with the Node Wrangler add-on (press *Shift* + *Ctrl* and right-click on any node).
5. We are now going to mix this result with a **Musgrave** node (press *Shift* + *A* and select **Texture** | **Musgrave**). We also need the Tiled UV for the vector input. We will set the **Scale** to **20.0**, the **Detail** to **3.5**, and the **Dimension** to **1.7**. Its effect will be pronounced in the final result, so we are going to mix it with white. To do this, we will add the **MixRGB** node and plug the first **color** input with the **Fac** output of the **Musgrave** Texture. The second color slot can be changed

to white. We are going to change the **Fac** slider of the **MixRBG** node to 0.98; this will make Musgrave very subtle.

6. We can now mix our noises and the **Musgrave** results together with one more **MixRGB** node.
7. If we plug this directly to the color input of the diffuse, we will get a black and white result. In order to introduce color, we will need another **MixRGB** node, but instead of feeding the color inputs we are going to plug our texture in the **Fac** input and choose two brownish colors. Now we can plug the result to the color input of the Diffuse shader.
8. Lastly, we can plug the black and white texture to the displacement input of the Material output node. In order to raise the displacement effect, we can place a **Math** node in-between (press *Shift* + *A* and select **Converter** | **Math**). We can change its operation to **Multiply** and use a **3.5** value.



The ground material with the procedural texture made with a noise and Musgrave combination on the left-hand side and, the result in the real time rendered 3D view.

Making and applying normal maps in Cycles

As we saw it previously with the alien character, normal map allows us to simulate a relief on a 3D mesh very efficiently. It would be good to generate a few of them in order to add some relief to our scene. We will explore a method to easily generate normal maps from tiled, hand-painted textures:

1. We will open a new Blender scene.
2. We will delete the cube (*X*) and then we will add a plane in the middle of the scene (press *Shift* + *A* and select **Mesh** | **Plane**) that we name as **Plane-1**.
3. We will split the screen in two parts to open the **UV/Image Editor** at the right-hand side.
4. We will add a **Multires** modifier to our plane and a **Displace** modifier. We will place the Multires modifier above the Displace modifier.
5. In the Texture tab, we will create a new texture by pressing **New,** and then we will load the tiled texture of wood, that is, **WoodTilePlank.png**.
6. We will check that the texture is loaded in the **Displace** modifier.
7. In the **Multires** modifier, we will check the **Simple** mode, and we will click on **subdivide** until we get to **level 8**. The more the mesh is subdivided, the more the Displace effect is accurate, but we must pay attention to the RAM of the computer.
8. We will modify **Strength** to **0.25**. This can vary depending on the texture. We must avoid important deformations on the mesh.
9. We will duplicate the plane (*Shift* + *D*), and then we will delete the modifiers of this new plane that we name as **Plane-2**.
10. We will select all the faces (**A**) of **Plane-2** in the **Edit Mode**, and we will do an unwrap (**U** | **Unwrap**) with a square shape taking all the UV surfaces. Then, we will add a new image. In the **UVMap Editor**, we will click on **New Image** (**Image** | **New Image**). We will name it as **WoodTilePlank_NM.png**.
11. We will move **Plane-2** to the same height as **Plane-1**.
12. Enter the top view (the *7* numpad key) for a better view of the mesh.

If the Displace doesn't give exactly the effect we want, we can make a few modifications with the **Sculpt** mode after applying the displace modifier by pushing on **Apply**. This is what we will do with the rock and the tile roof textures.

We can now bake a normal map as follows:

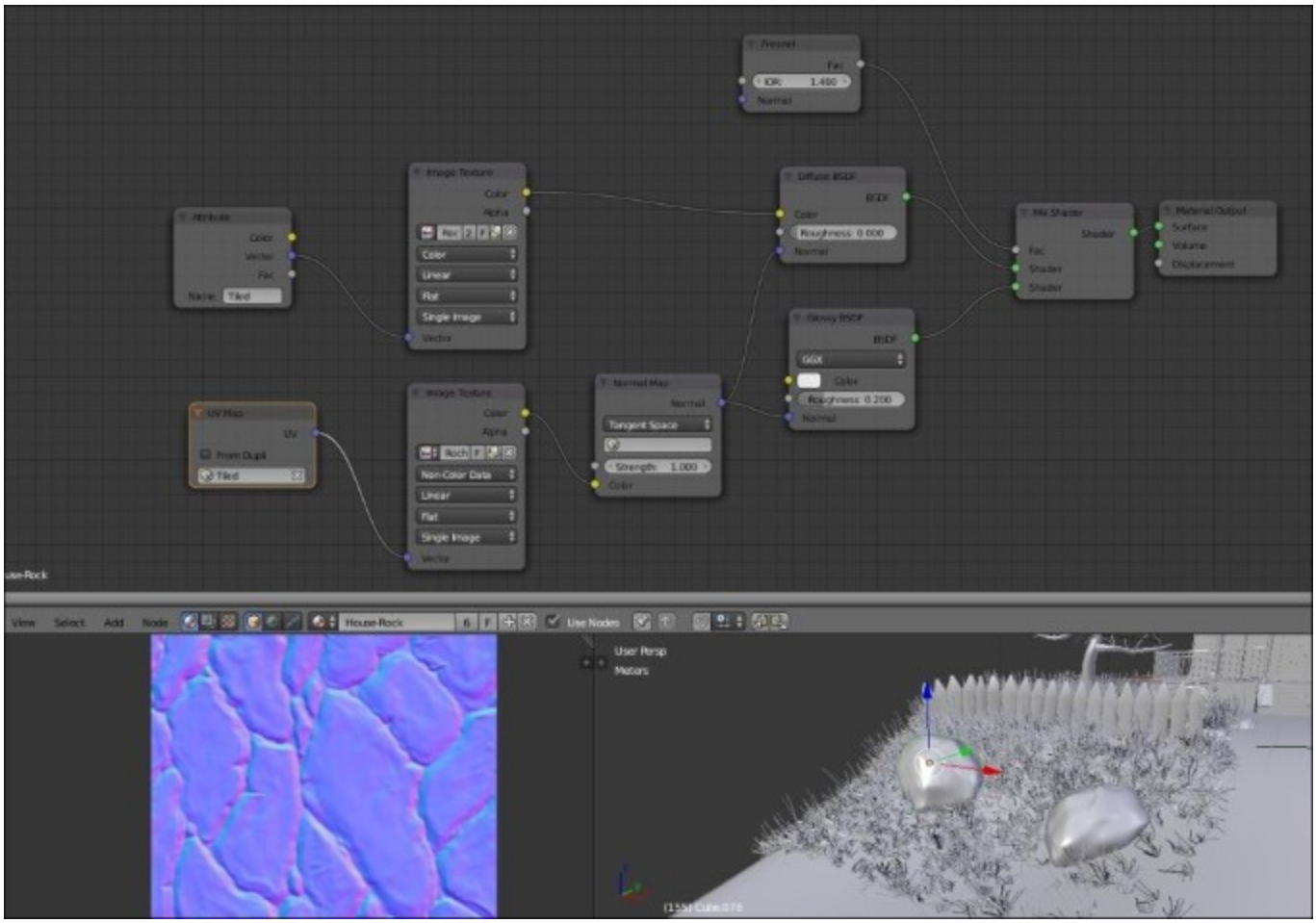
1. We will need to click on the **Smooth Shading** button, otherwise we will see the polygons on our bake.
2. Then, we will have to first select **Plane-1 (RMB)** and then **Plane-2** (press *Shift* and the RMB). This becomes the active object.
3. Now, in the **Properties** editor, under the **Render** section, we will expand the **Bake** subpanel.
4. The first option to choose is what type of map (or texture) we want to bake. So, in the **Bake Mode** drop-down menu, we will select **Normal Map.**
5. The next thing we'll have to check is the **Selected to Active** option that tells Blender to bake from the sculpture to the active object (our low poly plane).
6. Now you can click the **Bake** button. Don't forget to save your map (select **Image** | **Save As Image** or press *F3*), or it will be lost!

We will use the same process of Normal Map creation for every tiled texture. Once this is done, we can return to our shaders and apply our normal maps.

We will start with the House-Rock shader, which is very simple for the moment:

1. We will add a **Glossy** node (*Shift + A)* and navigate to **Shader** | **Glossy BSDF**, which we will mix with **diffuse** using a **Mix Shader** (*Shift + A* and select **Shader** | **Mix Shader**). To better visualize the normal maps, we will need glossiness.
2. We will add **Fresnel** (*Shift + A* and navigate to **Input** | **Fresnel**) that we will position in the **FAC** input socket of the **Mix Shader** node. We will put a **IOR** value of **1.4**.
3. We will add a **Normal Map** node (*Shift + A* and select **Vector** | **Normal Map**) with a **Strength** value of **1.0** and connect its output to the Normal input socket of the Diffuse and Glossy shaders.
4. We will duplicate an Image Texture node (*Shift + D*), and we will open the normal map texture named Roch-Tilling-NM.png file. We must switch the **Color Data** option of this second Image Texture to **Non-Color Data**. The data should not be interpreted as color data but as normal direction data.
5. We will add a UV Map node (*Shift + A)* and navigate to **Input** | **Glossy UV Map**. Then we will change the UVLayer to **Tiled**. You will recall that the rocks have two UV layers, so we must select one.

We can apply this process for almost every shader using hand-painted tiled textures, except the brick walls in our case. It is a special case that requires that we bake the normal map on a larger map with a little modification of painting in order to hide the bricks behind the windows.



The normal map of the rock (low left corner) and its material in the nodal editor (top)

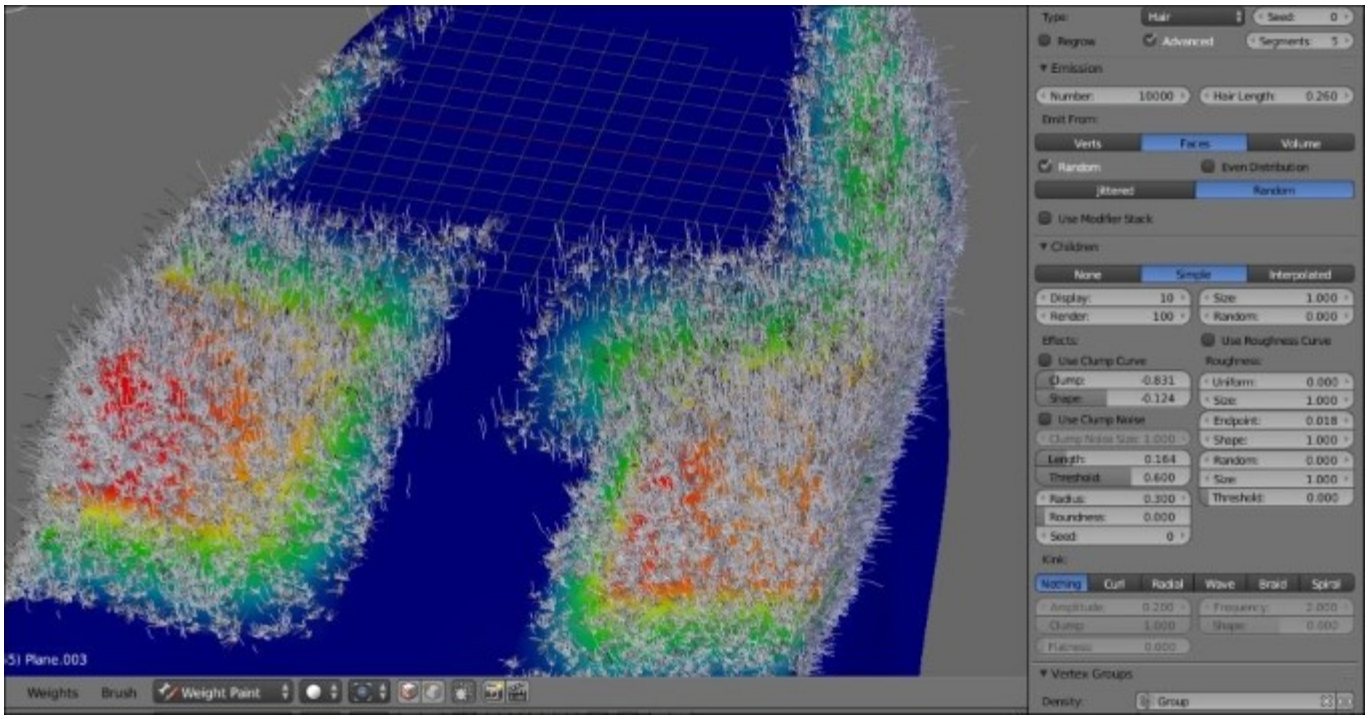
Creating realistic grass

In this section, you will see how we can create realistic grass in place of the actual grass planes. In order to create our realistic grass, we are going to use a hair particle system.

Generating the grass with particles

A particle system will be used here in order to generate the grass strand without modeling and placing them by hand:

1. We will first select the cliff and isolate it (**/ Numpad**). Then, we can go into the **Particle** tab of the **Properties** editor and add a new particle system. This will add a new modifier in the stack, but we can only control it here.
2. We will now have to change the type from **Emitter** to **Hair**. We can activate the **Advance** tab.
3. In the **Emission** subpanel, we can change the **Number** value to **10000**, which corresponds to the number of grass strands that we will have. We will also emit the particle from the faces in a **Random** manner. We can also change the **Hair Length** to **0.26**.
4. In the **Physics** subpanel, we can change the **Brownian** value to 0.120 in order to add more randomness to the grass.
5. In the **Children** subpanel, we can set the amount of strands we want to spawn around the main guides. In our case, we will activate the Simple mode and set **10** children for the **preview** (in the viewport) and **100** for the **render**. In the effect section, we can change the **Clump** value to **-0.831** so that the children start near the base of the guide strand and are sprayed out near the tip. We can also change the **Shape** value to **-0.124** to shrink the children in the middle a little. We will also change the **Endpoint** value to 0.018 to add more randomness.
6. We will now paint a vertex group with the **Weight paint** tool in order to choose where we want grass on the cliff. For instance, we don't want strands under the house. To do this, we will switch from **Object mode** to **Weight Paint** mode while the cliff is selected. As you can see, you have brushes as in the **Sculpt** mode. We can use the **Add** brush to add weight and the **Subtract** brush to remove weight. **Red** means that it will be full of grass, **Blue** means you won't have grass. Now, we can choose the vertex group that we have painted in the **Density** field of the **Vertex Groups** subpanel in the Particle System settings.
7. Now we can add a new particle system in order to add long grass. To do this, we will add a new slot in the particle settings. We will also copy the settings from the previous particle system but unlink them (the button with the number on the right-hand side of the name). We can lower the number of particles to **1000** and change the **Hair Length** to **1.120**. The number of children will be **5** in the **Simple** mode.

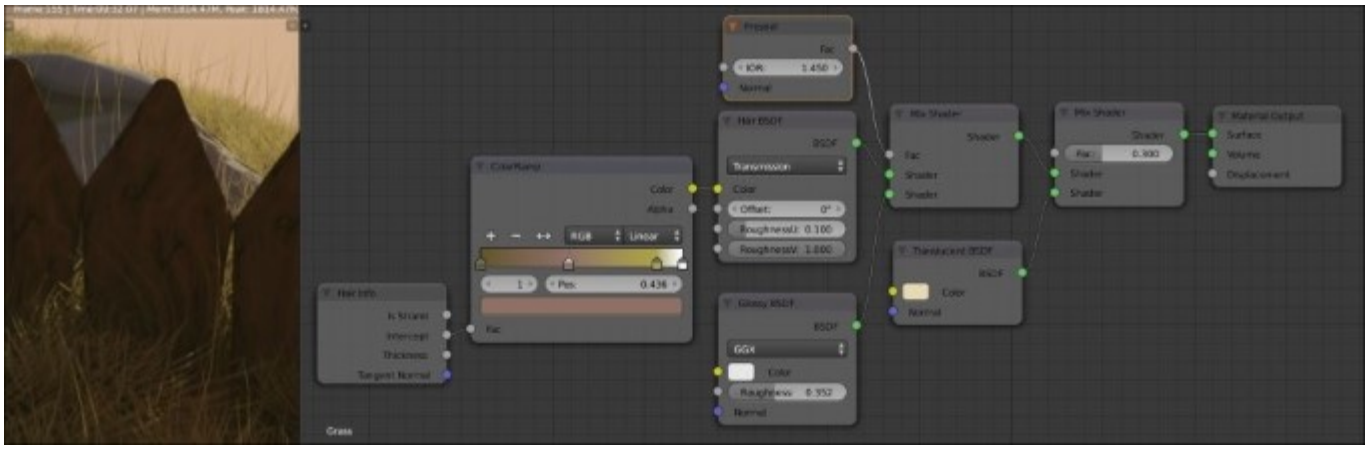


The settings of the grass

Creating the grass shader

Now the last thing that we will create is the grass material:

1. The first thing we will need to do is change the **Cycles Hair Settings** in the **Particle Systems** tab. We will change **Root** of the strand to **0.20** and set **Tip** to **0.0**.
2. We can now add a new material slot for the cliff and rename it as Grass. In the particle settings, we will change the material to **Grass** under the **Render** subpanel.
3. We will now select the grass shader and open the **Node** editor. The first node that we will add is the **Hair BSDF** shader (press *Shift* + *A* and select **Shader** | **Hair BSDF**) and change it from **Reflection** to **Transmission**. We will mix it using **Mix Shader** with a **Glossy BSDF** shader. We will change **Glossy Roughness** to **0.352** so that the glossiness is more diffuse.
4. Next, we will have to plug a **Fresnel** node to the **Fac** input of **Mix Shader**.
5. For the color of **Hair BSDF**, we will add a **Color Ramp** node (press *Shift* + *A* and select **Converter** | **Color Ramp**). For its **Fac** input, we will add a **Hair Info** node (press *Shift* + *A* and select **Input** | **Hair Info**) and choose the **Intercept** output. This will enable us to set the different colors along the strand. We will do a gradient that starts from a brownish color (to the left) to a desaturated green (to the right). Usually, the tip of the grass strand is white, so we will add a small amount of white on the far right of the color ramp.



The grass shader (to the right) and the result (to the left)

6. We will also mix the result of our Mix shader with a **Translucent BSDF** shader (press *Shift* + *A* and select **Shader Translucent**). Indeed, the grass is very translucent. We will change its color to a desaturated yellow and change the **Fac** value of the shader to **0.3**. We can finally plug our latest **Mix** shader to the surface input of the **Material** output node.

Baking textures in Cycles

Cycles allow us to bake textures as does Blender Internal, but there are some differences between the two render engines.

Cycles versus Blender Internal

As we have seen previously, texture baking in Blender Internal can be very efficient to produce normal maps, ambient occlusion, color textures, and many other kinds of maps that we won't cover here. All of this in a very short time. So you might wonder why it is interesting to bake in Cycles.

Cycles is a ray tracer render engine based on physical parameters with global illumination. It is then possible to get some very realistic renders in a much more efficient manner. Baking in Cycles allows us, for example, to calculate a few heavy special effects only once, such as caustics. When a render is baked on a texture, you can visualize the effect in real time. This can be very useful if you want to change the frame and make several renders. In this way, it is possible to create a realistic environment in real time.

However, in the context of a video game, if you have many dynamic assets you must pay attention. You could be limited by fixed lighting. Even though baking in Cycles can be very interesting, it has some faults. Doing a good baking without noise requires the same settings as a normal render, so you do need a high sampling value, which greatly increases rendering time compared to Blender Internal.

Note

For more information, you can have a look at the official Blender Reference manual at this address:

http://www.blender.org/manual/render/cycles/baking.html

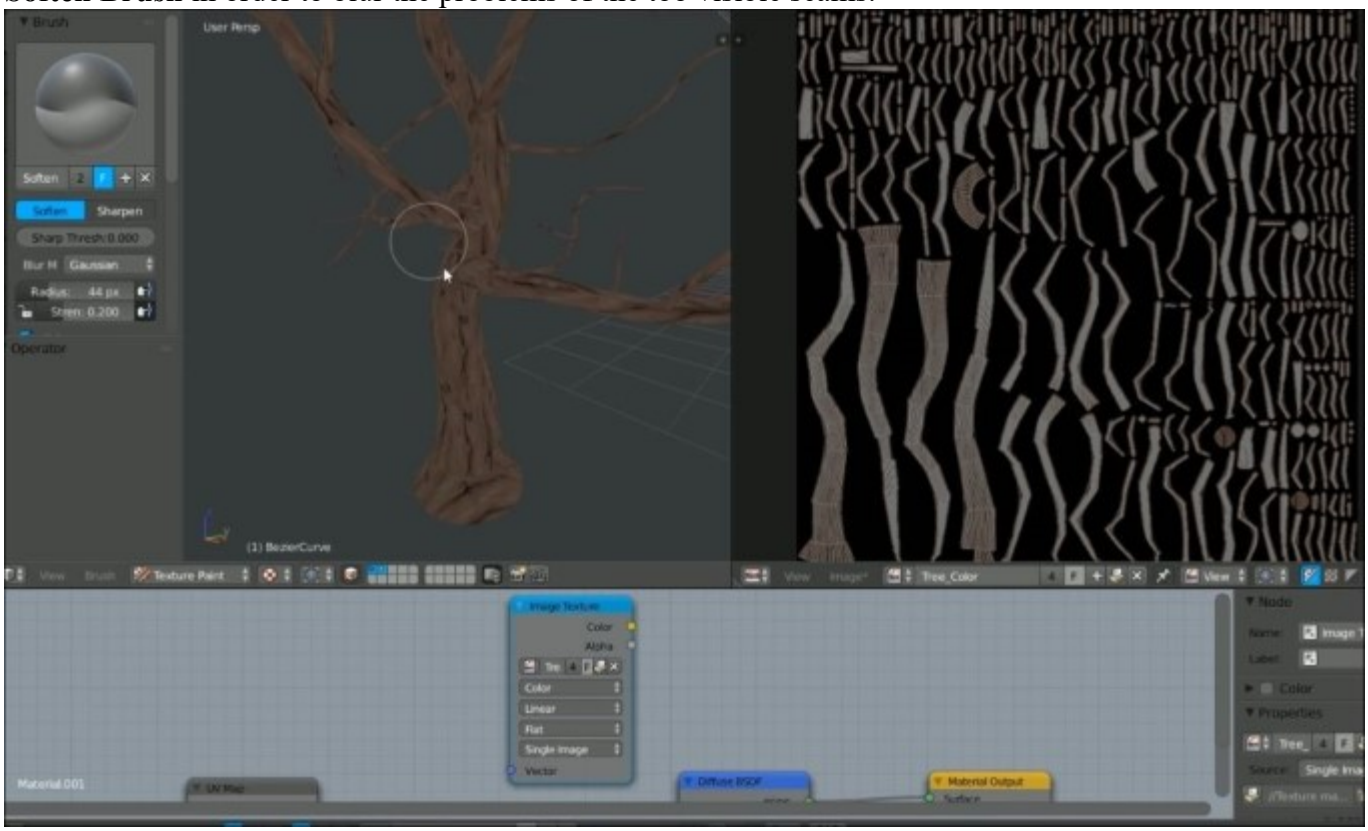
Baking the tree

We won't bake the maps of every objects in our scene with Cycles; however, we will see how to proceed with the 3D mesh of the tree as an example.

In order to optimize the render time, we will import our 3D mesh to another Blender window:

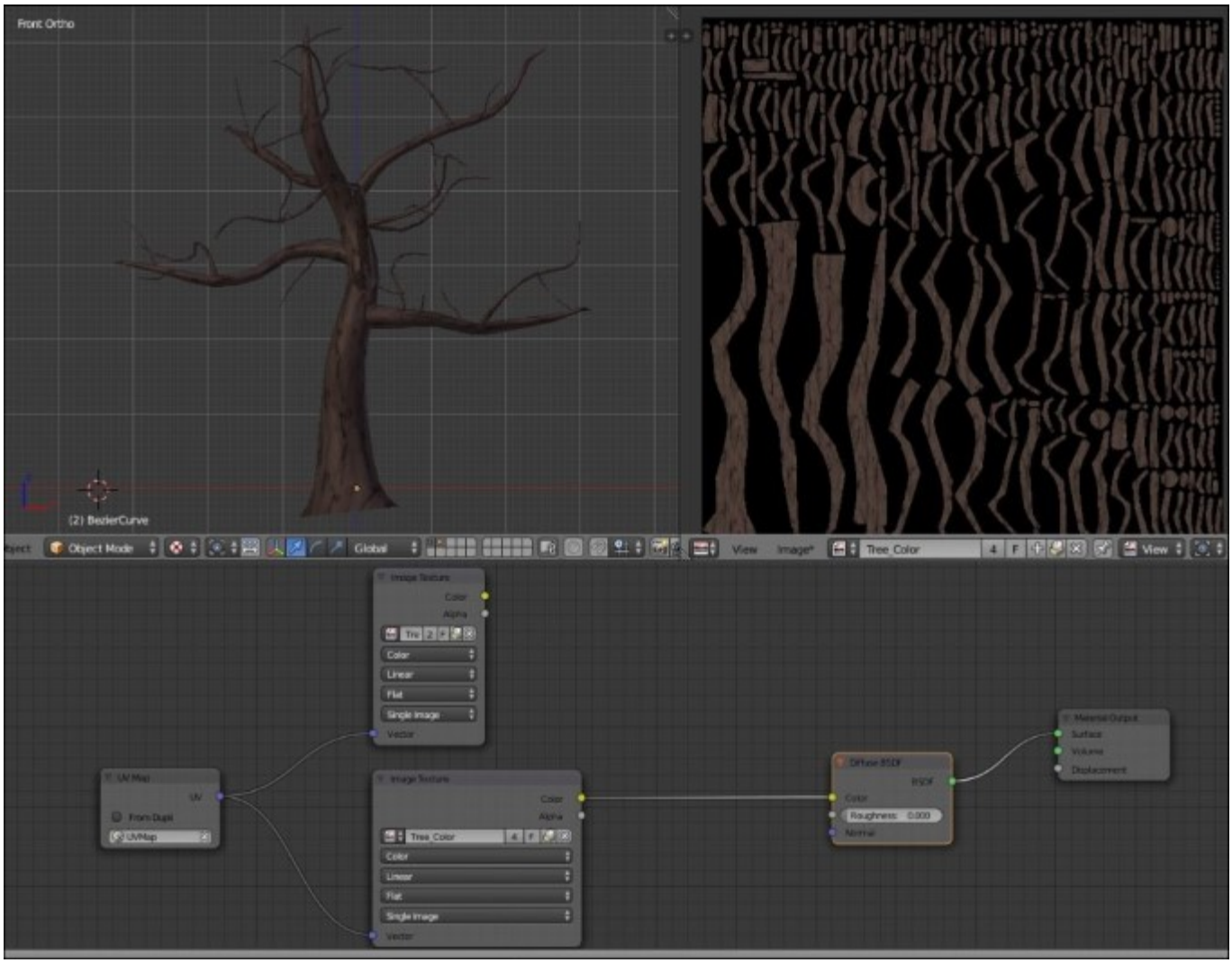
1. We will launch Blender a second time.
2. We will select the tree in the scene of the haunted house, and we will press *Ctrl* + *C* to copy it. You will see the **Copied selected object to buffer** message in the header of the work space.
3. In the other Blender window, we will press *Ctrl* +*V* to paste the tree. We will see the **Objects pasted from buffer** message.
4. In the same way, we will also import all the lights, and we must recreate the shader of the Image Base Lighting (we can append it from the main file). We don't modify the location of the tree and the lights in order to keep the same light configuration. But this wont be exactly the same lighting effect as we don't have the house here.
5. We will need a second UVs layer with UVs that are restrained in the UV square this time. We will use the *Ctrl* + *P* shortcut to automatically replace the UVs. Then we will adjust the margin in the options of the left panel of the 3D viewport.

6. We will start with color baking using the new UVs. For this, we will add an **Image Texture** node (*Shift* + *A* and select **Texture** | **Image Texture**) to the shader of the tree.
7. We will select all the polygons of the tree in the **Edit Mode** (*Tab* and *A*), and then we will create a new image named **Tree_Color**. A size of 2048 x 2048 is enough.
8. In the **Image texture** node that we have just created, we will select the **Tree_Color** texture. This node must stay u.nconnected.
9. Now we will need to go into the **Bake** tab in the **Render** options. Here, we will change the type to **Diffuse Color**. We will set the **Margin** value to **5**, and then we will press **Bake**.
10. When we have our color map, we must adjust the seams with Texture Paint. We will use the **Soften Brush** in order to blur the problems of the too visible seams.



Hiding the seams on the color bake

11. When the color texture is fine, we will again select the polygons of the tree in the **Edit Mode**, and we will create a new image (the same size) and rename it as **Tree_Combined**.
12. Now we can make another combined baking following the same process, which this time allows us to get all the lighting information on the texture. Make sure you open the good image in the **Image Texture** node with a high enough sample value. In our case, we have 500 samples to obviate the noise.
13. We can now go back to our haunted house scene and replace the tree with the one with new UVs (*Ctrl* + *C* and *Ctrl* + *V*); then we can replace the old texture with the **Tree Combined** texture in our shader.



The combined bake of the tree

Compositing a mist pass

As a bonus, we are going to learn how we can create and composite a mist pass with Cycles. But to do this, we will need to do a render. So let's do a render with 500 samples:

1. We will first have to activate the mist pass in the **Render Layer** tab of the Properties editor. Now we can access the mist settings in the World setting panel. We will set **Start** to **0 m** and Depth to **37 m**.
2. In order to see the mist, we will need to composite it over the render. We will learn more about compositing in further chapters, so don't worry if we don't go deep into the subject right now. In the Node Editor, we will have to switch to the Compositing Node mode (the second button after the material in the header). We will need to check **Use Nodes** and **Backdrop** in order to see our changes in real time. As you can see, we already have a RenderLayers node plug in the **Composite** node (the final output of the image).
3. In between, we can add a **Mix** node (press *Shift* + *A* and select **Color** | **Mix**) and feed the first color input with the Image output of the **RenderLayers** node. The **Fac** input of the **Mix** node will receive a **Map Value** node (press *Shift* + *A* and select **Vector** | **Map Value**) with **Offset** of **0.105** and **Size** of **0.06**. For the input of Map Value, we will simply plug our **Mist** pass (the fourth output of the **RenderLayers** node). The **Map Value** will control the amount of mist we see.
4. In order to view the result in the **Node Editor** in real time, you will have to add a **Viewer** Node. To do this, we will simply press *Ctrl* + *Shift* and right-click on any node.

The final Cycles render of the Haunted House project

5. We now have a nice mist! In order to change the aspect of the final render, we can tweak the **Color Management** options as we did in the previous chapter. Congratulations, you've completed the Haunted House project.

Summary

This chapter was really robust, but you now understand how to create nice materials with the Cycles Render engine and how to light a scene properly. You also learned how to produce normal maps and how to bake your objects. This last technique would be very interesting for video games. We also covered Blender's Compositing tool to a slight extent by mixing a mist pass. Now let's create a new project!

Chapter 8. Rat Cowboy – Learning To Rig a Character for Animation

This chapter will cover the rigging and the skinning of a character. This character will be a Rat Cowboy that has been already modeled for you. Here, you will understand what the rigging process involves. We will start by placing deforming bones. After this, we will learn how to rig these bones with controllers and constraints such as IK or Copy. Then, we will skin our character so that the mesh follows the deforming bones. As a bonus, you will learn how to use shape keys in order to add some basic facial controls that will be controlled by drivers. The rig, which is covered here, will be basic, but you will have all the necessary knowledge to go further. We are going to use this rig to animate our character in the next chapter. Enjoy!

In this chapter, we will cover the following topics:

- Making a symmetric skeleton
- Using the basic bones constraints
- Rigging the eyes
- Correcting the deformation of the meshes with weight painting
- Improving the accessibility of the rig with custom shapes
- Using shape keys

An introduction to the rigging process

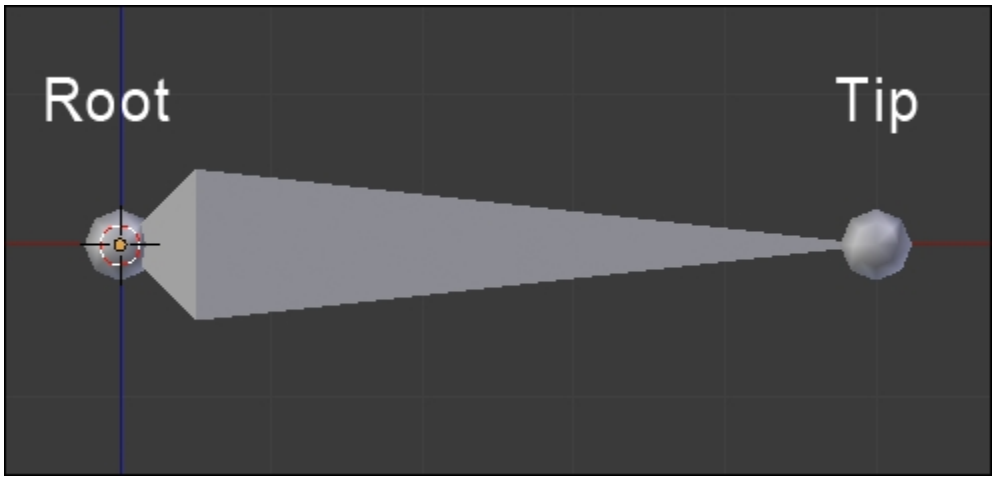
We are now going to discover the process of character rigging. The point of this is to prepare objects or characters for animation in order to pose them in a simple way. For instance, when rigging a biped character, we will place virtual bones that mimic the character's real skeleton. Those bones are going to have relationships between them. In the case of a finger, for instance, we will usually add three bones that follow the phalanges. The tip bone will be the child of the mid bone, which in turn will be the child of the top bone. So when we rotate the top bone, it will automatically rotate its children. On the top of the network of the bones, we will need to add some constraints that define automation so that it is easier for the animator to pose the character. The next step is to specify to the geometry to follow the bones in some way. For instance, in the case of a character, we will tell Blender to deform the mesh according to the deformable bones. This stage is called **Weight Painting** in Blender and **Skinning** is a common term, too. However, we will not always face a case where skinning is necessary. For instance, if you have to rig a car, you will not want to deform the wheels, so you will create a bone hierarchy or constraints in order to follow the rig. The entire process could be tricky at some point, but mastering the rigging process allows you to better understand the animation process and is the reason why having a good topology is so important.

Note

Anatomy of a bone in Blender

A bone has a root and a tip. The root corresponds to the pivot point of the bone, and the tip defines the length of the bone. Bones can have a parent-child relationship in two ways. The first method is by connecting them, so the root of the child is merged with the tip of its parent. The other method is by

telling Blender that they are visually disconnected while still having a parent/child relationship. Each bone has a roll that corresponds to its orientation on itself. When manipulating an **Armature** object, you can be in the **Edit Mode** to create the network of the bones and set their relationships, or you can be in the **Pose Mode** where you can pose the rig as if you were posing a marionette.



Rigging the Rat Cowboy

Let's do the rig of the Rat Cowboy. We are not going to show the modeling process here as you already know how to model proper characters from the Alien project.

Placing the deforming bones

The first thing that we will need to do for our rig is place the bones that will directly deform our mesh. These are the main bones. In Blender, a rig is contained in an **Armature** object, so let's go!

Let's begin with the process:

1. We will first be sure that our character is placed at the center of the scene with his feet on the *x* axis.
2. Now we can add a new bone that will be placed in an **Armature** object (press *Shift* + *A* and select **Armature** | **Bone**).
3. Next, we will enter the **Edit Mode** of our new **Armature** object, and we will place the bone in the hip location and rename it as `hips`. You can rename a bone in the right panel of the 3D view in the **Item** subpanel. Be careful to rename just the bone and not the **Armature** object.
4. We can now start to extrude the bones of the spine. To do this, we will select the tip of the hips and extrude it (*E*) twice as far as the base of the neck. It's very important that you don't move these bones on the X axis. We will rename the bones as **Spine01** and **Spine02** respectively**.** From the side view, be sure that these bones are slightly bent.
5. We will now extrude the left clavicle according to the Rat Cowboy's structure and rename it as **Clavicle.L**. The **.L** part is really important here because Blender will understand that this is on the left-hand side and will manage the right-hand side automatically later when mirroring the rig.
6. Now, from the tip of the clavicle, we will extrude the bones of the arm. Rename the two bones as **TopArm.L** and **Forearm.L**.
7. Now it's time to extrude the bone of the hand, starting from the tip of the forearm. Name this as **Hand.L**.

   ### Note

   Please note that if you want to follow the process step by step, you can download the starting file for this chapter on the Packt Publishing website.

8. In order to create the finger bones, we will start from a new chain and parent it back to the hand. This will allow us to have bones that are visually disconnected from their parents. To do this, we will place the 3D cursor near the base of the first finger and press *Shift* + *A*. Since you can only add bones when you are in the **Edit Mode** of the **Armature**, this will automatically create a new bone. We will orient it correctly and move its tip to the first phalange. We will extrude its tip to form the next two bones. It's important to place the bones right in the middle of the finger and on the phalanges so that the finger bends properly. Analyze the topology of the mesh to do this precisely. If you want, you can activate the Snap option (the magnet in the 3D view header) and change its mode from **Increment** to **Volume** to automatically place the bones according to the volume of the finger.

9. Then we will create the chains for the other fingers and the thumb and rename them as **Finger[Which finger]Top.L**, **Finger[Which finger]Mid.L**, and **Finger[Which finger]Tip.L**.
10. We will now need to re-parent them to the hand so, when the hand moves, the fingers follow. To do this, we will select the top bone of each finger (the root of each finger chain) and then select the hand bone (so that it is the active selection) while holding *Shift*, pressing *Ctrl* + *P*, and selecting **Keep Offset**. Keep Offset means that the bones are going to be parented but they will keeping their original positions (that is, they are not connected to their parent).
11. We will now change our cursor location to the left thigh of our character and press *Shift* + *A*. We can then place the tip of this bone to the knee location. Also, check the side view and put this tip a little bit forward. We can then extrude a new bone from the knee to the ankle. Rename these bones as **Thigh.L** and **Bottom Leg.L**.
12. We can then extrude the foot and the toes. Name them as **Foot.L** and **Toes.L**.
13. The leg chain needs to be parented to the hips. This can be done with *Ctrl* + *P* and selecting **Keep Offset**. Remember to first select the child and then the parent while doing your selection for parenting.
14. Now we can add the bones of the tail. We will create a chain of bones starting from the back of the Rat Cowboy to the tip of the tail where the tips and roots of each bone are placed according to the topology of the mesh. Remember to rename the bones properly from **Tail01** to **Tail07**.
15. Then we will parent **Tail01** to **Hips** by pressing *Ctrl* + *P* and selecting **Keep Offset** so that the whole tail is attached to the rest of the body.
16. The last bones that we will need to extrude are the neck and the head. The neck starts from the tip of the **Spine01** bone and goes straight up along the Z axis. Then we will extrude the head bone from the neck tip. We will rename them as **Neck** and **Head**.

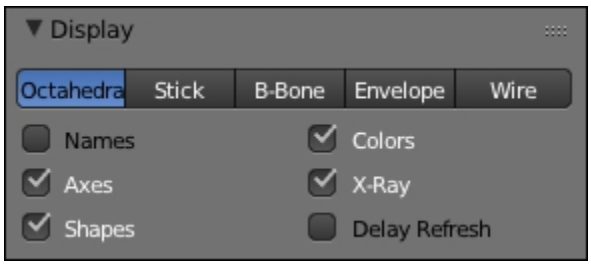
Now we will have to verify on which axis the bone will rotate. You can display the axes of the bones in the **Armature** tab of the **Properties** editor under the **Display** subpanel. We will need to adjust the roll of each bone (*Ctrl* + *R* in the **Edit Mode**) to align them along the *x* axis. You can test the rotations by going to the **Pose Mode** (*Ctrl* + *Tab*) and rotating the bones around their *x* local axis by pressing *R* and then pressing *X* twice. Beware, rolls are very important!

Placement of the deforming bones

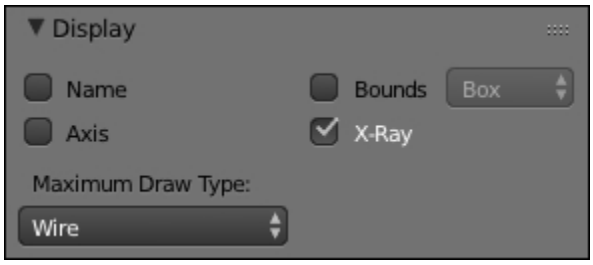
Note

The Display options

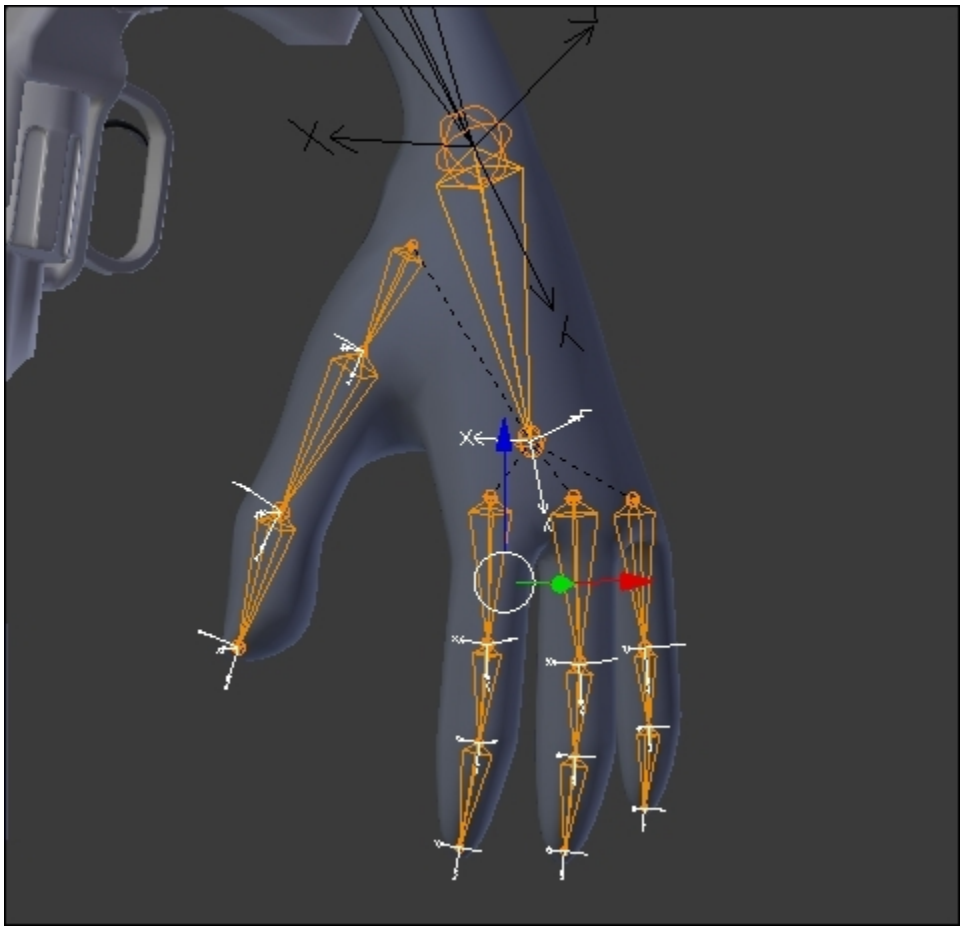
You can find different display options for your bones in the **Display** subpanel in the **Object Data** tab of the **Properties** editor. A nice way to display the bones is to activate the **X-Ray** display mode, which allows us to see the bones through the mesh even in **Solid** shading mode. We can also display the axes of orientation and the name of each bone and change its shape.



For instance, we can use the B-Bone mode that changes the bones to boxes that we can rescale with *Ctrl* + *Alt* + *S*. This is a nice way to display bones that are on top of each other. You can also use the **Maximum Draw Type** drop-down menu in order to change the shading of your selected object in the **Display** subpanel in the **Object** tab of the **Properties** editor.



The following image will show the placement of the deforming bones of the hand:



Placement of the deforming bones of the hand with a correct roll

The leg and the foot

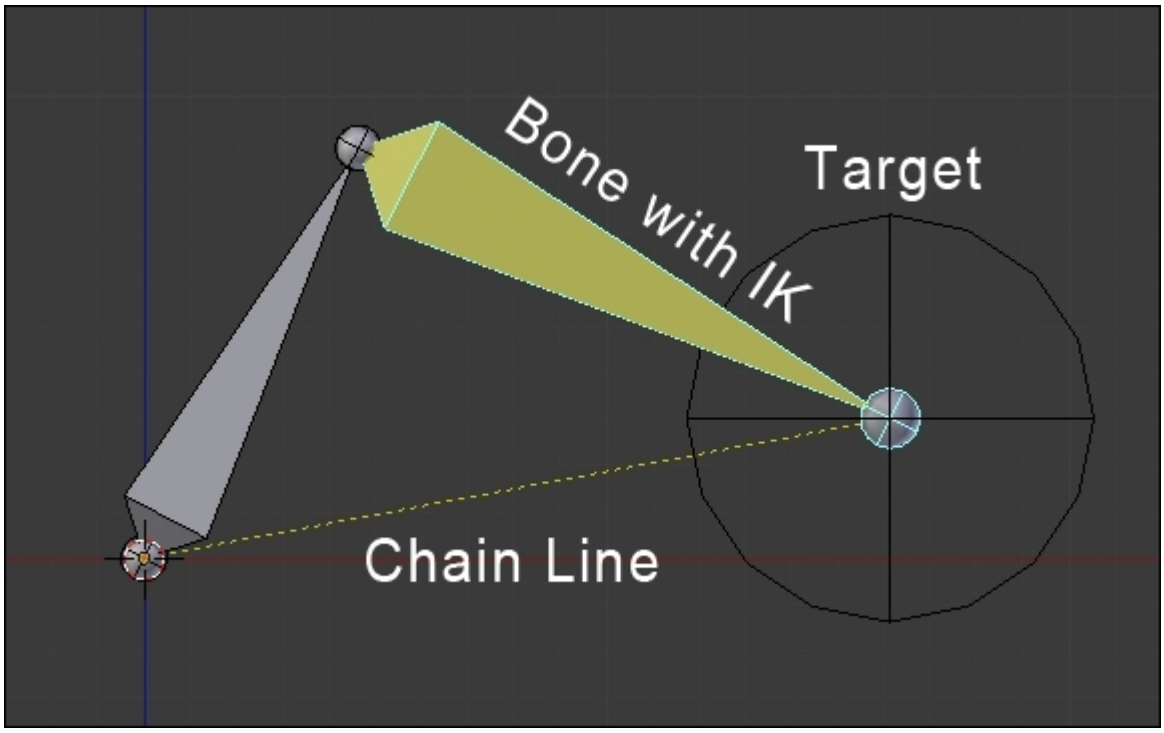
Now that we have all the deforming bones that are needed, we are going to add some bones that will help us to control the leg and the foot in a better way.

1. We will now add a bone that will be a controller for the **Inverse Kinematic** (**IK**) constraint of the leg. We add this to the ankle and align it with the floor. It's important that this bone is disconnected for now. We rename it as **LegIK.L**.
2. Under the **Bone** tab of the **Properties** editor, we will uncheck the **Deform** checkbox so that our bone does not deform our geometry later.

   ### Note

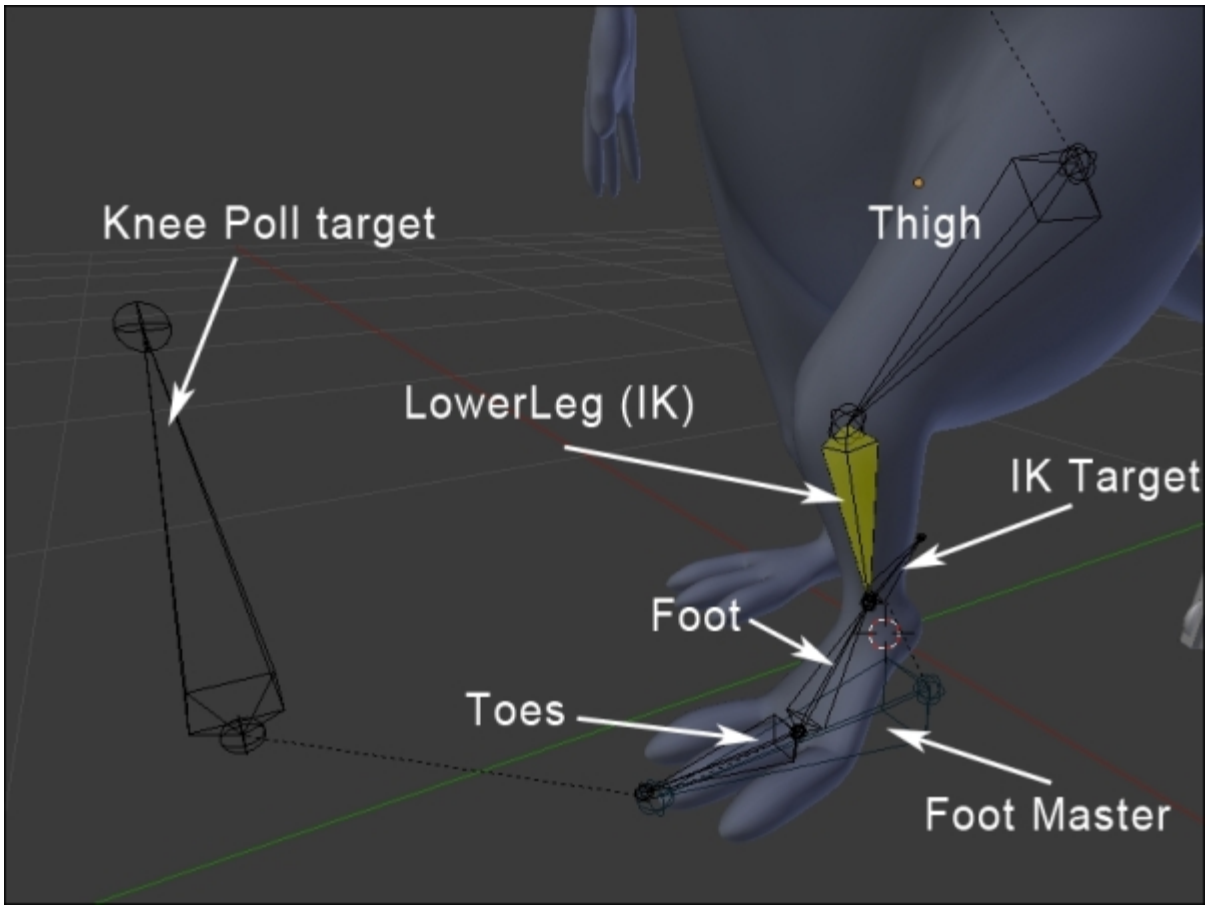
   What is an IK constraint?

   Usually, when you manipulate bones in the **Pose Mode**, you rotate each one in order to pose your object, and this method is called **FK** (**Forward Kinematic**). The role of an **IK constraint** is to let Blender calculate the angle between a minimum of two bones according to a target. To better understand what this does, imagine that your foot is a 3D object and you can move it where you want in space. As you can see, your thigh and lower leg will automatically bend with an appropriate angle and direction. That's the whole point of **IK**!



3. Now we are going to tell Blender that this new bone is the target that will control the IK constraint. To do this, we will first select it in the pose mode (*Ctrl* + *Tab*) and then select the lower leg bone to make it the active bone. Now we can use the *Shift* + *I* shortcut to create a new IK constraint. As you can see, the lower leg bone turns yellow.

4. Now we will change the settings of the constraint. The bone Constraints panel is located in the **properties** editor (a bone with a chain icon) in the **Pose Mode**. If we select the lower leg bone, we can see our IK constraint located here. As we've used the *Shift + I* shortcut, all the fields are already filled. The setting that we will change is **Chain Length**. We set this to two. This will tell Blender to calculate our IK constraint from the bone where the constraint is on the tip to the next bone in the leg chain.
5. We can now go back to the **Edit Mode** (*Tab*) and add a new floating bone in front of the knee location. This bone will be the target of the knee. Rename it as **PollTargetKnee.L**.
6. Back in the **Pose Mode** (*Ctrl + Tab*), we will set this new bone as the **Pole Target** bone for the IK constraint. We will first select the **Armature** object and then the bone. After this, we will adjust **Pole Angle** to reorient the IK constraint so that the leg points to the knee target. In our case, it's set to **90°**.
7. The next thing to do is to uncheck the stretch option so that the leg can't be longer than it already is.
8. Now that we've rigged the leg, it's time to rig the foot. We are going to make an easy foot rig here. But note that a foot rig can be much more complex with a foot roll. In our case, we will first remove the **Foot.L** parentation. To do this, we go into the **Edit Mode**, select the parent, press *Alt + P*, and select **Clear parent**.
9. Now we want to switch the direction of the bone so that its root is located at the toes. To do that, we will select the bone in the **Edit Mode**, press *W*, and select **Flip Direction**. Remember that a bone rotates around its root, so this will give us the ability to lift the foot up on the character's toes.
10. Now we can connect the IK target to the foot bone. To do this, we will simply select the child (**LegIK.L**), select the parent (**Foot.L**), press *Ctrl + P*, and select **Connected**. So now, when we rotate the foot in the **Pose Mode**, the IK target is going to lift up and the IK constraint will do its job.
11. The last thing that we will need to do is create a master bone that will move all our foot bones. In the **Edit Mode**, we will add a bone that starts from the heel to the toes and rename it as **FootMaster.L**. We will then parent the toes to this with the **Keep Offset** option. Then we will parent the foot to the toes with the **Keep Offset** option. As you can see, if you move the master bone in the **Pose Mode**, all the bones will follow this. We are done with the foot and the leg rig!



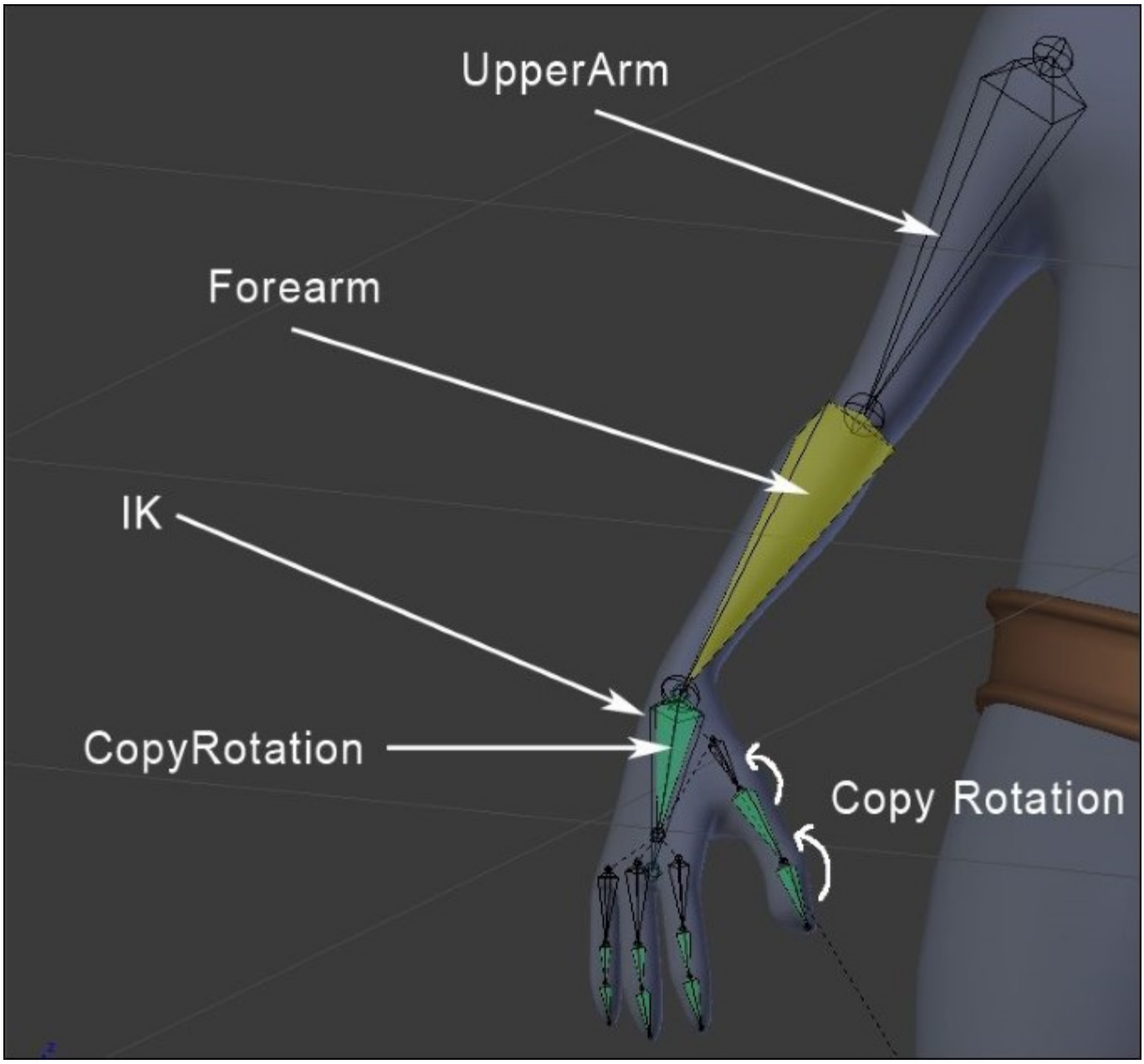
The rigging of the foot and the leg

The arm and the hand

The next important part to rig is the arm. In many rigs, you will have a method to switch between FK and IK for the arms. In our case, we are only going to use IK because it will be quite long and boring to teach how to create a proper IK/FK switch with snaps. When animating the arm with IK, you will have to animate arcs by hand, but this will allow you much more control if you don't have an FK/IK switch.

1. We will create a new floating bone that will become our IK target for the arm by duplicating the bone of the hand and clearing its parent. Remember that the target of an IK switch needs to be freely movable! We will rename this bone as **HandIK.L**.
2. Then we will set up our IK constraint by first selecting the target, then the forearm, and then pressing *Shift* + *I*. Now, we can change the chain length to two as we did for the leg.
3. The next thing to do is add a poll target for the orientation of the elbow. To do this, we will create a floating bone behind the elbow of our left arm, we rename this as **ElbowPollTarget.L**, and we set this as the poll target of the IK constraint. Also, we will change the **Poll Angle** to match the correct orientation of the elbow.
4. Both the target and the IK target need to have the **Deform** option turned off.

5. The animator will only want to manage one bone for the hand. The bone that will deform the hand is not the target as it needs to be connected to the arm, so we will need to find a way to tell the hand-deforming bone to follow the IK target rotation. If this happens, the animator will only control the target for both arm placement and the hand rotation. To do this, the IK target is going to transfer its rotation to the deform bone. So, we will first select the **HandIK.L** bone, then the **Hand.L** bone, and then press *Ctrl* + *Shift* + *C* to open the constraint floating menu and select **CopyRotation**.
6. We are now going to rig the fingers again with a **Copy Rotation** constraint. The motion that we want to achieve is that, when the base of the finger is rotated around the X local axis, the finger curls. To do this, we will indicate to the mid bone of the finger to copy the rotation of its parent (the top bone) and the tips to copy the rotation of its parent too (the mid bone). We will show the process for the index finger and let you do the rest for the others.
7. We will select the **Finger1Top.L** bone, then the **Finger1Mid.L** bone, press *Ctrl* + *Shift* + *C*, and select **CopyRotation**. After this, we will select the **Finger1Mid.L** bone, then the **Finger1Tip.L** bone, and create a copy rotation constraint. If we rotate **Finger1Top.L** on the X local axis, we can see that the finger bends.
8. In order to finish the hand, we will disable the **Y** and **Z** local rotations of the mid and tip bones of each finger. To disable a rotation on a particular axis, we will open the right panel of the 3D view (*N*) and change the rotation type from **Quaternion** to **XYZ Euler**. Then we can use the lock icon on a particular axis.



The rigging of the arm and the hand

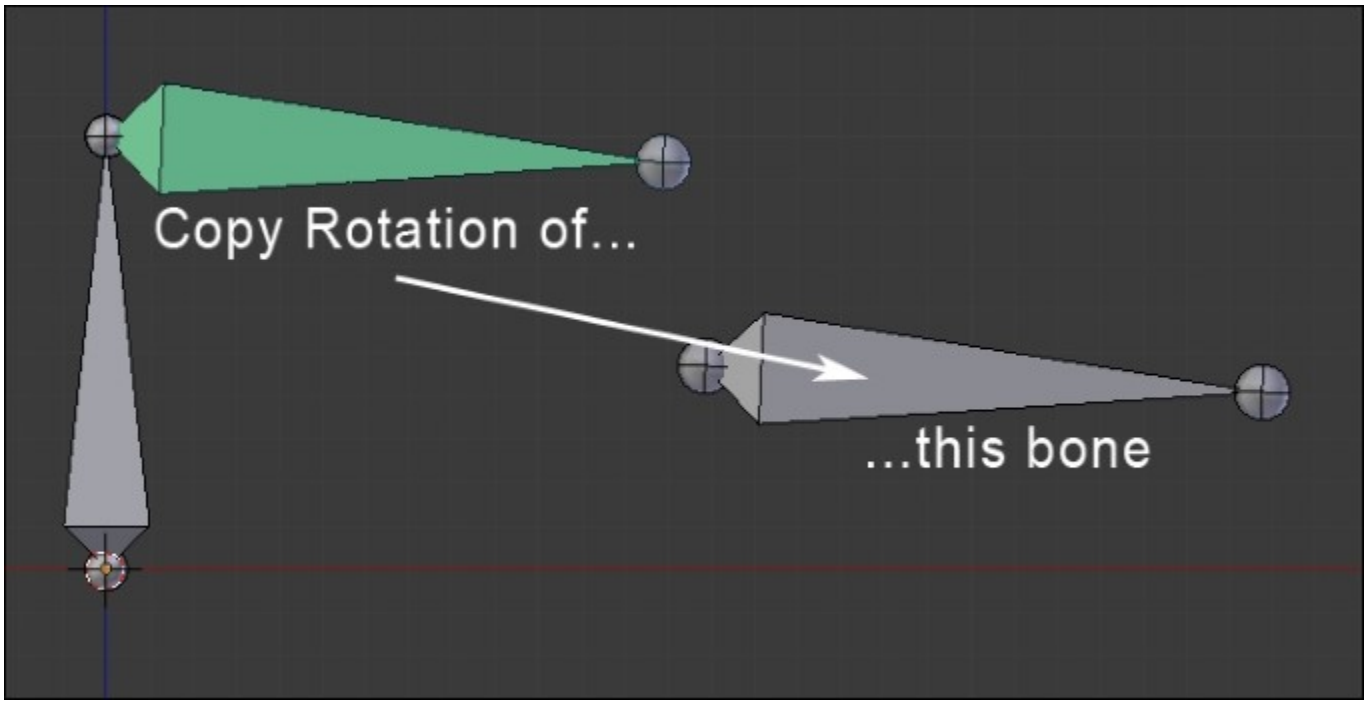
Note

What is a Copy Rotation constraint?

As its name implies, the copy rotation constraint will tell an entity to copy the rotation of another entity. The settings of the constraint allow you to choose in which space the rotation will be. The often used spaces are **World** or **Local**.

The **World** space has its axes aligned with the world (you can see them in the left corner of the 3D view).

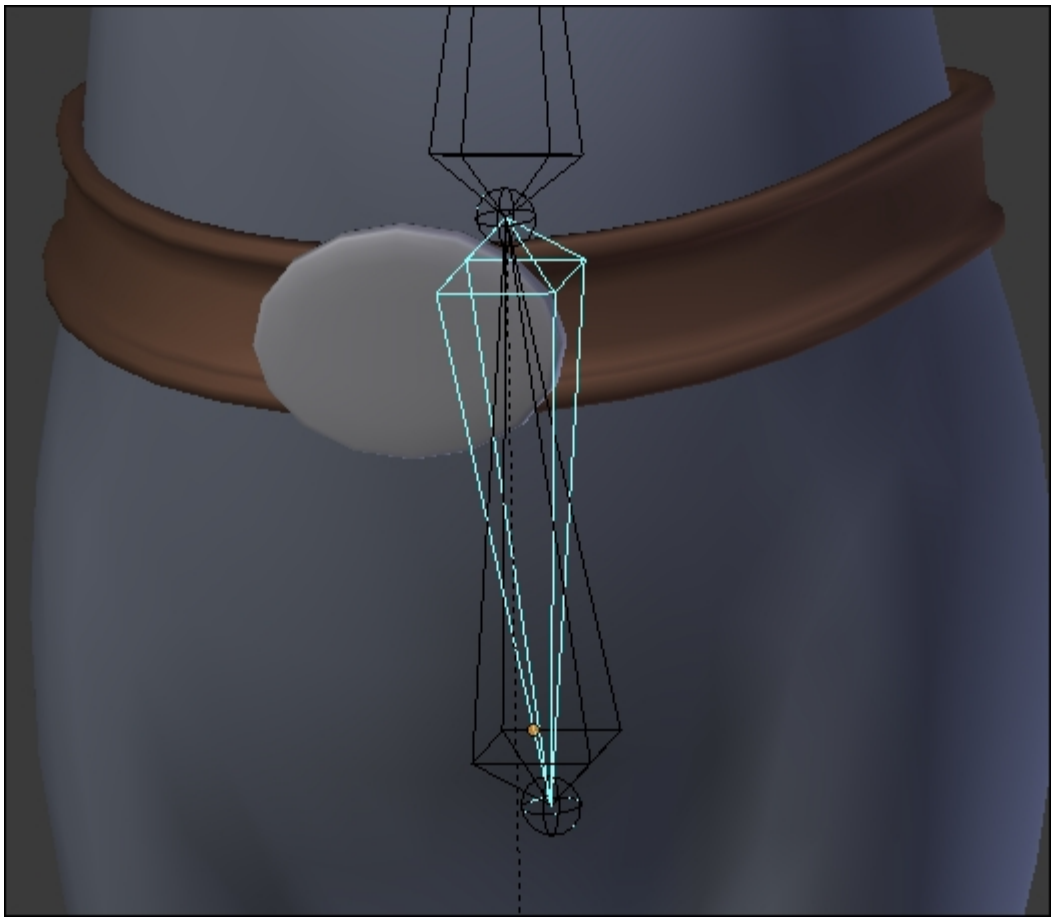
The **Local** space has to do with the orientation of an object. For instance, if an airplane has a certain direction, but when it rotates around its fuselage, this takes into account its orientation.



The hips

Now it's time to do the hips motion so that it is easier to control for animation. You have done a lot of work until here. Have yourself a cookie, you deserve it!

1. To create our hips motion, we will duplicate the hip bone. Also, remember to uncheck the **Deform** option. We will rename it as **HipsReverse**.
2. Now, we are going to flip the direction of this bone so that the hip deforming bone rotates around its tips (because the root of the hip's reverse bones will be here).
3. Now you can test in the **Pose Mode** that, when you rotate the **HipsReverse** bone, the hip's deform bone rotates with it.

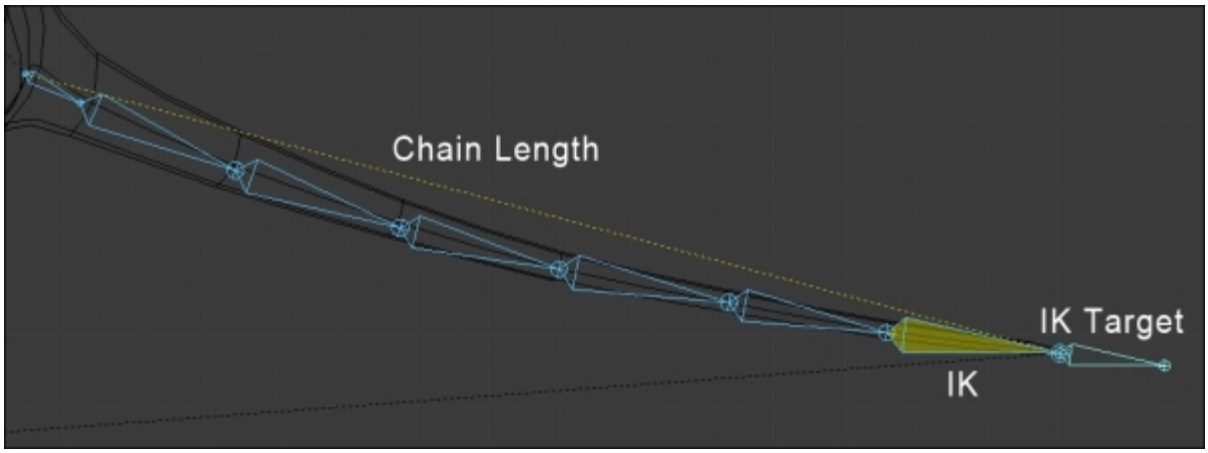


The rigging of the hips with the reversed bone

The tail

A rat without a tail is pretty strange, so we will take some time to rig his tail. The technique that we will show here is quite simple but very effective:

1. In order to rig the tail of the Rat Cowboy, we will need to add a new bone that controls this in the **Edit Mode**. To do this, we will extrude the last **Tail07** bone so that it is placed at the right location, and we will un-parent it with *Alt* + *P* and select **Clear Parent**. Rename this as **TailIK**.
2. We will now need to create an IK constraint. We will first select the target, then the **Tail07** bone (the last in the chain), and press *Shift* + *I*.
3. Now in the IK constraint settings, we will change the chain length to **7** (to let the IK constraint solve the angles from the tip to the last bone of the tail chain).
4. We will check the **Rotation** option, too. Now, as you can see, the tail is fully rigged and can be placed and rotated through the tail target:

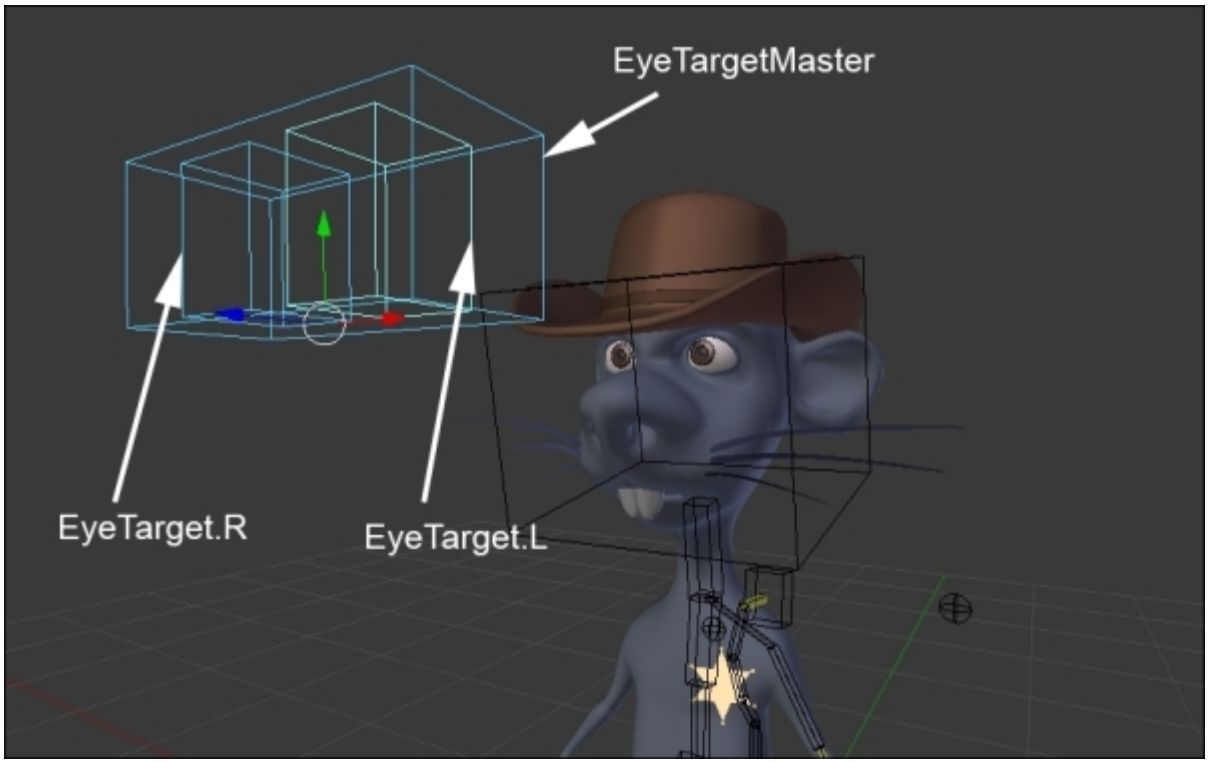


The rigging of the tail with a chain length of 7

The head and the eyes

In order to control the head, we will only manipulate the bone of the head, so we will directly begin the rigging of the eyes. We will ensure that the eyes are two separated objects and they have their pivot points at the center to make for good rotation. In our case, we have two half spheres, so we don't waste performance with hidden geometry:

1. We will select one of the eyes, and in the **Edit Mode** (*Tab*), we will select the outer edge loop. We will press *Shift* + *S* to open the *Snap* menu, then we will select the **Cursor to Selected** option, and then, in the **Object Mode** (*Tab*), we will press *Ctrl* + *Alt* + *Shift* + *C* and select **Origin to 3D cursor**. The pivot point must be at the right location. Do not hesitate to test this with a free rotation **(R x 2)** in the **Object Mode**. We must repeat the same process for the other eye.
2. In the **Object Mode**, we will select the eyes and the teeth, and then we will parent them to the head bone, but not with the usual method of parenting the bones that we saw earlier. To do this, we will first select the eyes and the teeth, and then the head bone (the armature must be in the **Pose Mode**). We will press *Ctrl* + *P*, and we will select the **Bone** option.
3. We now want to create a controller bone for each eye. We will select the armature, and in the **Edit Mode** (*Tab*), with the 3D Cursor in the middle of the eye, we will create a bone (*Shift* + *A*). In the Orthographic (*5*) left (*3*) view, we will move the controller bone in the front of the character. We need a small distance between the head and the bone controller. We will repeat the same process for the other eye, and we will rename them as **EyeTarget.L** and **EyeTarget.R**.
4. We will set the eyes to look at their controllers with a **Damped Track** constraint (**Properties** | **Constraint**). We will start with the left eye. We must select the Armature as **Target** and **EyeTarget.L** as the bone. Now, we must adjust the rotation by tweaking the **Z axis**.

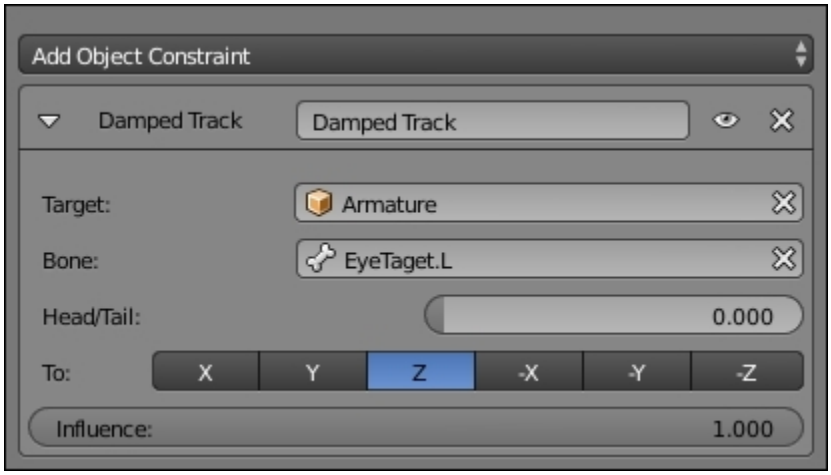


The eyes' controllers

Note

The Damped Track constraint

This allows us to constrain a 3D object to always point towards a target on an axis. The 3D object doesn't move, it just rotates on its pivot point depending on the location of the object.



Both the eyes must now follow the **EyeTargetMaster** movements. Do a test by pressing *G*.

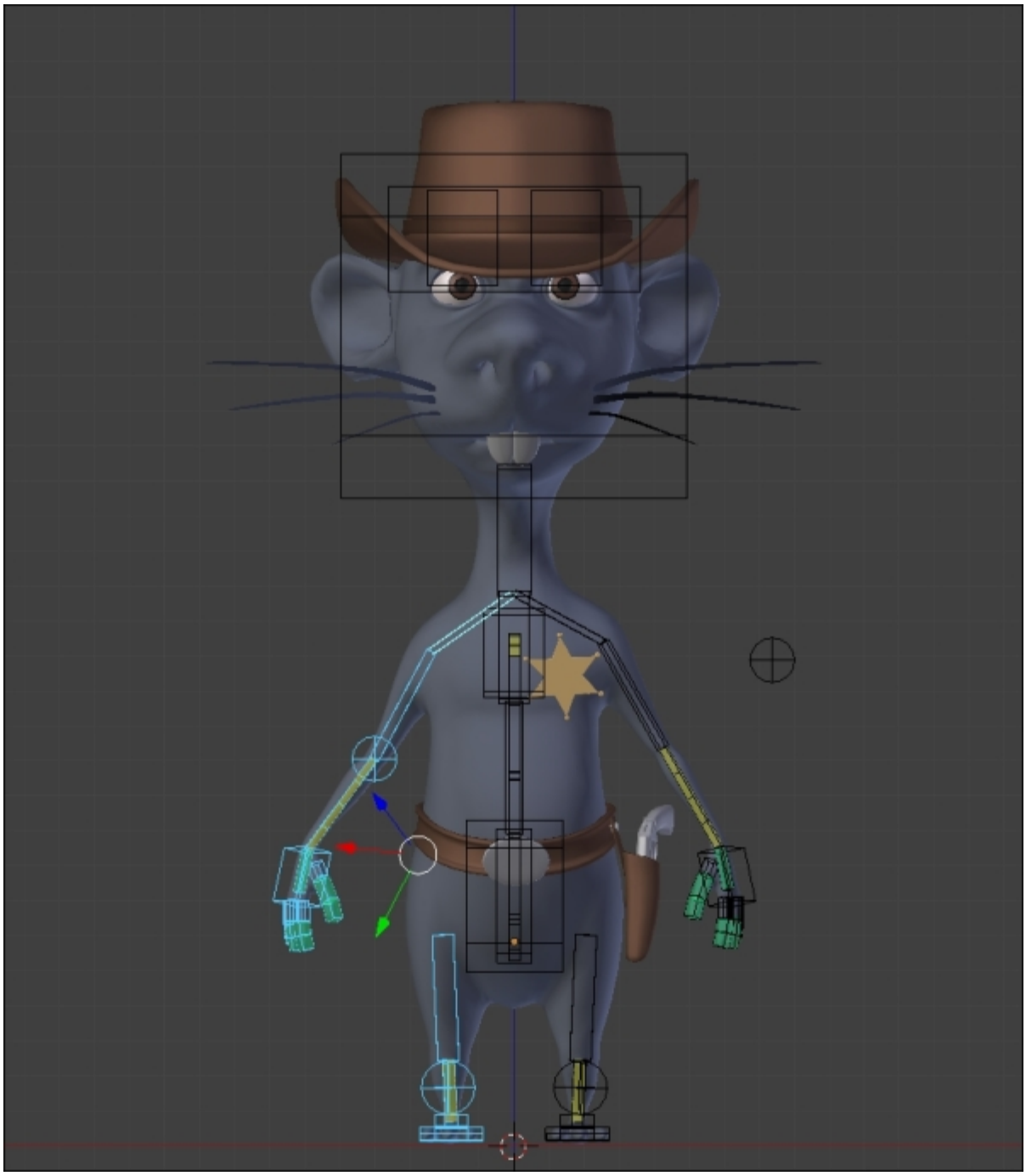
Mirroring the rig

In order to save time, as in the modeling or the sculpting process, it is often very useful to work with symmetry. There are three ways to do this in an armature.

The first method consists of checking the **X-Axis Mirror** option in the **Armature Options** tab in the left panel of the 3D Viewport (*T*) that allows us to directly create the bones in a symmetry. We must extrude the bones by pressing *E* while also pressing *Shift*. This solution doesn't copy the constraints.

The second method is efficient even if it requires some manipulations to get a perfect mirror. Until then, we will place the bones of the arms and legs on the left-hand side with all the constrains that we need and the appropriate names:

1. We will select the armature in the **Edit Mode**, and we will align the 3D cursor at the center (*Shift* + *S* and select **Cursor to Center**).
2. In the **Header**, we will put the **Pivot Point** options on 3D Cursor.
3. Then, we will select every bone of the arm and the leg on the left-hand side.
4. We will duplicate them (*Shift* + *D*), and we will mirror them by pressing *S* + *X* + *1* on the numeric keyboard. Then we will press *Enter*.
5. In the **Edit Mode**, we will select the bones of the arm and the leg on the right-hand side and flip the names (**Armature** | **Flip Names**). This renames all the bones of the left-hand side with the **.R** termination.



Mirroring the bones with their constraints from the left to the right side

The third method is very interesting. It has been available since Blender 2.75.

Once we have placed and named all the bones with the **.L** termination and have all the constraints, we will select them in the **Edit Mode**, and then we will press *W* and select Symmetrize. This will automatically rename the bones of the right-hand side and the constraints will be copied.

Let's now focus on the gun for a rig.

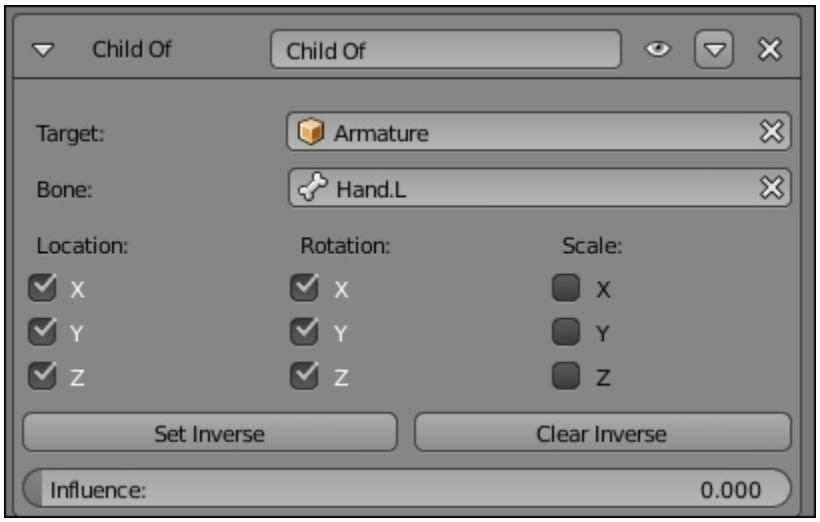
Rigging the gun

In order to easily animate the gun, we will need it to be able to follow the hand of our character when the Rat Cowboy uses it, and the gun must be able to follow the holster all the time. To do this, we will use a bone and a **Child Of** constraint.

Note

The Child Of constraint

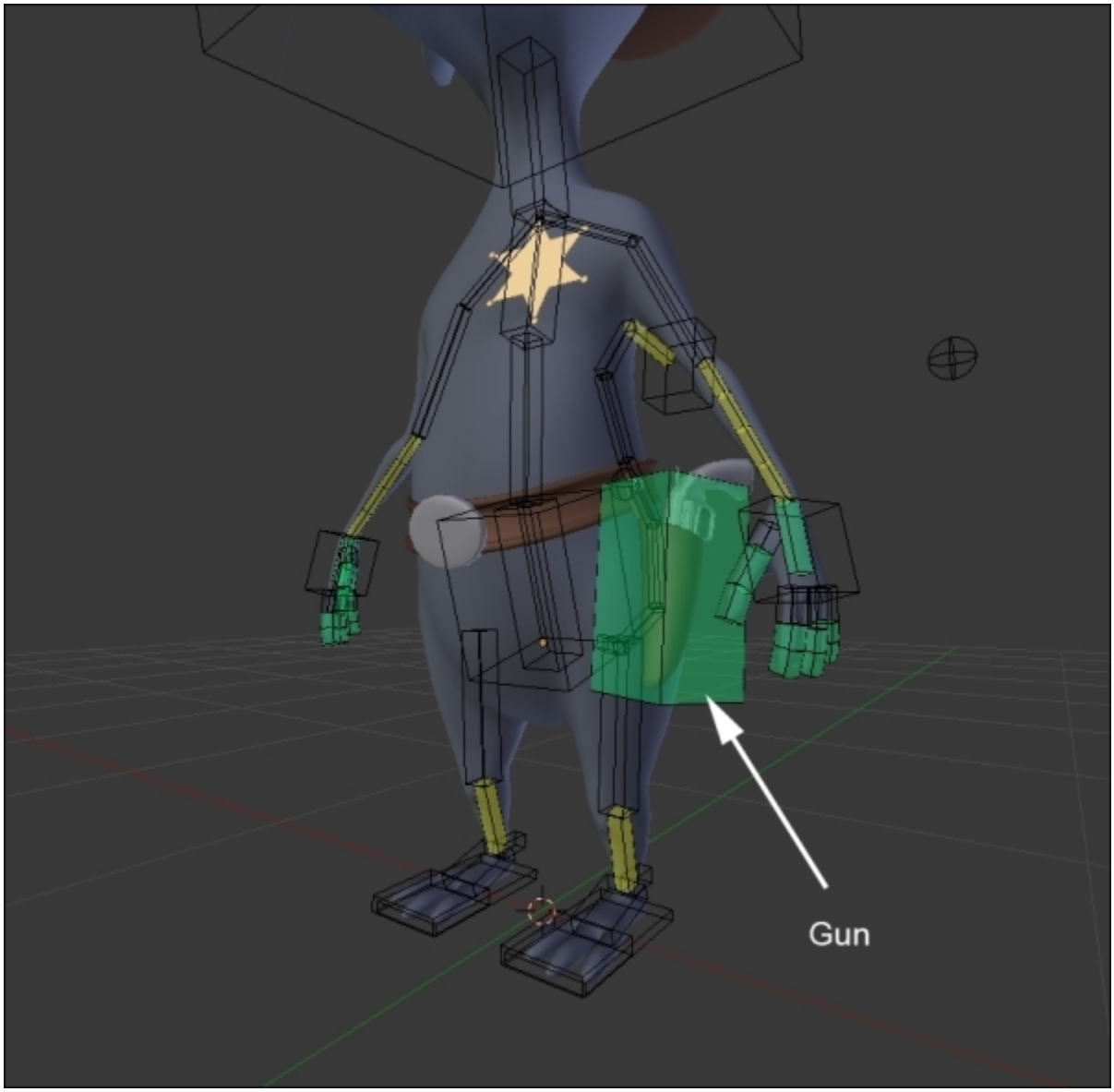
This constraint allows us to make an object a parent of another object by weighting their influences. This allows the animator to animate its influence in order to change its parent. This is much better than a classical parentation. This is very useful to make an object follow different objects one after another like a character driving with one hand on the steering wheel and the other hand on the speed box. You can also combine multiple children of the constraints.



1. We will begin by taking care to apply the rotation of the gun (press *Ctrl* + *A* and select **Scale and Rotate**).
2. We will select our Armature, and in the **Edit Mode**, we will place the 3D Cursor at the location of the hammer; then we will create a bone (*Shift* + *A*) that follows the length of the gun. We will rename this new bone as **Gun**.
3. We will make the gun the parent of the bone by first selecting the gun in the **Object Mode**, then the **Gun** bone in the **Pose Mode**, and then we will press *Ctrl* + *P* and select **Bone**.
4. We will then modify the appearance of the bone in B-Bone (**Properties** | **Object Data** | **Display** | **B-Bone**). We will scale it to make it easier to rig (*Ctrl* + *Alt* + *S*).
5. We will check the **X-Ray** option.
6. We will move the gun in the holster by moving and rotating the Gun bone. You can also directly rotate the gun a little bit in order to get the best position.

7. We will add two **Child Of** constraints to the Gun bone. We will uncheck the scale option of both the constraints.
8. We will decrease the influence to **0** on both the **Child Of** constraints**.**
9. With the first **Child Of** constraint, we will put our Armature as Target with the **Hand.L** bone in the bone option. We will press the **Set Inverse** button.
10. For the second **Child Of** constraint, we will put the **Holster** object as Target, and we will press the **Set Inverse** button again.

Now, when we place the left hand with **HandIK.L** near the gun and put the influence of the first **Child Of** constraint at **1.000** (the influence of the second Child Of constraint must still be at **0**), the gun joins the **HandIK.L** bone and follows it. In order to reposition the gun in the holster, it must be very near to the holster. The influence of the first Child Of constraint must be at **0,** and the influence of the second one must be at **1.0** (reversed influences).

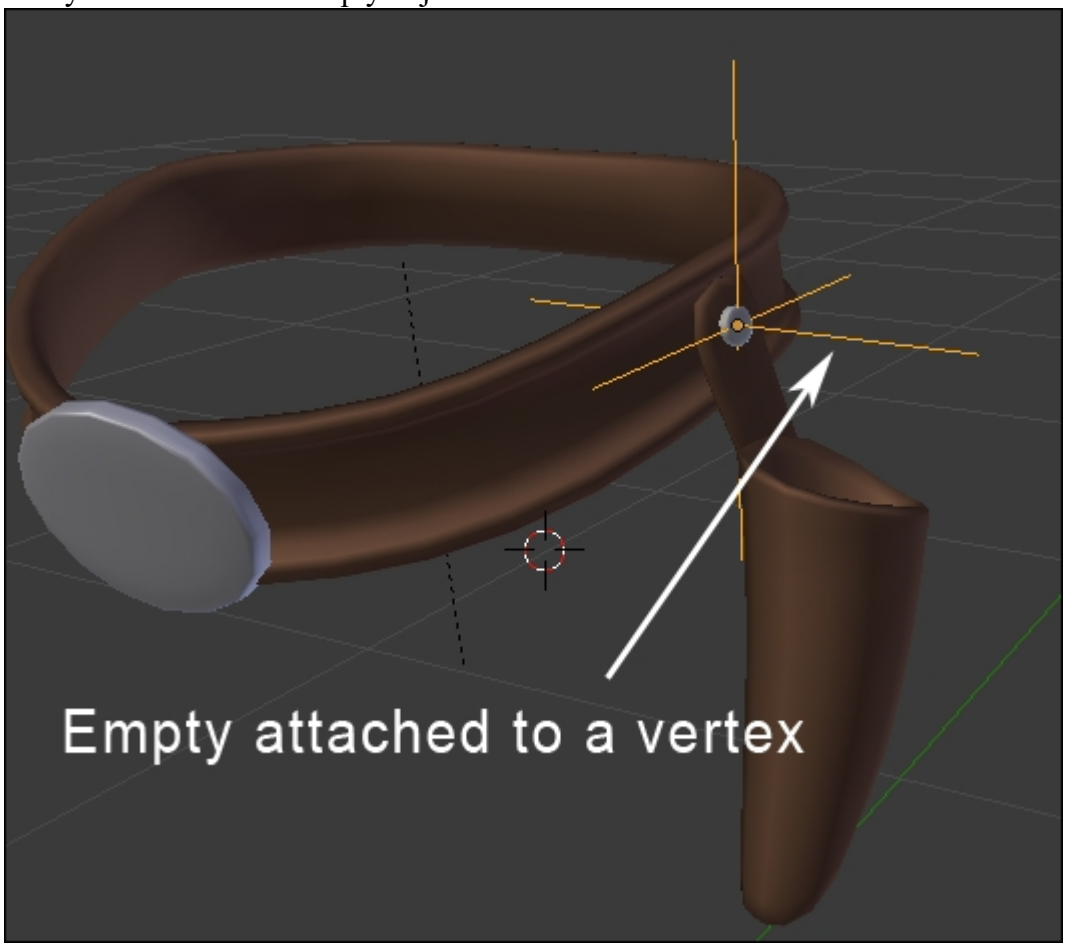


The gun bone

Rigging the holster

Now we are going to rig the holster. As you can see, it is a separate object. We will need to pin it to the belt. In this section, we won't use bones, but they are still a part of the rigging process.

1. First, what we are going to do here is create an empty object that will be the parent of the holster. To create an empty object, press *Shift* + *A* and select **Empty** | **Plain Axis**.
2. Now we will select the empty object, and while holding *Shift,* we will select the belt. We can now enter the **Edit Mode** of the belt and choose a vertex near the pin of the holster. We can press *Ctrl* + *P* and choose **Make Vertex Parent**. Now when we move the vertex, it moves the empty object!
3. The last thing that we need to do is make the holster the parent of the empty object, and the trick is done! Of course, we could have made the holster object a parent of the vertex directly, but it's always nice to have an empty object in between in this kind of situation.

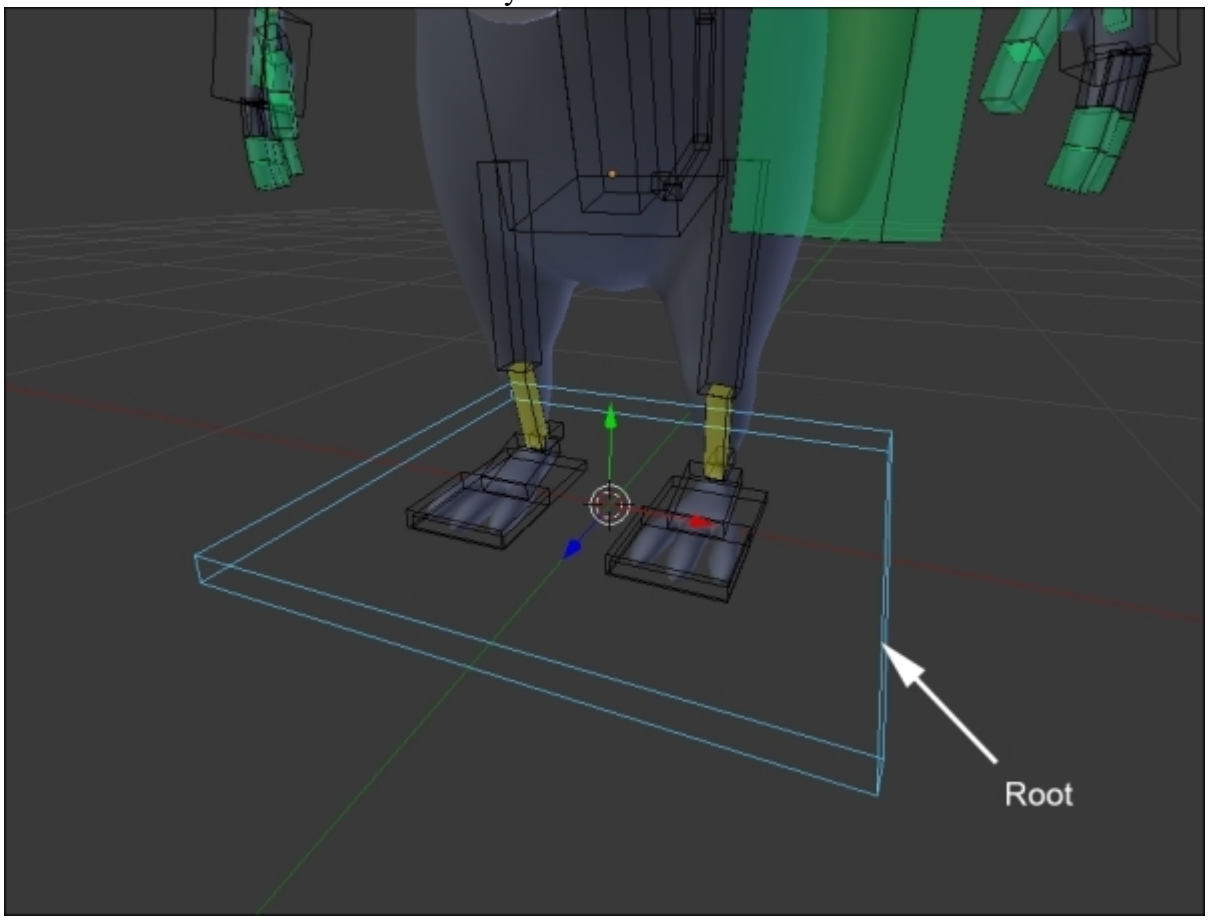


The rigging of the holster

Adding a root bone

The root bone is also called the master bone; it is the bone that will control the entire skeleton and is the top parent. With this, it is very convenient to place our character and animate it anywhere.

1. We will select the **Armature,** switch in the **Edit Mode** (*Tab*), place the Pivot Point at the center of the world (press *Shift* + *S* and select **Cursor to Center**), and then we will add a bone (*Shift* + *A*).
2. We will make this bone bigger (*Ctrl* + *Alt* + *S* in the **B-Bone** mode) because it represents the central control element of the **Armature**. We will flatten it by selecting and moving the tip of the bone. We will rename it as **Root**.
3. In the **Edit Mode**, we will select the IK bone controllers of the **hands**, **feet**, **tail**, **EyeTargetMaster** that controls the **Eyes**, **Hips**, and **Pole Targets** at the location of the knees and the elbows; finally, we will select the master bone (so that it is the active selection).
4. We will make them parents (press *Ctrl* + *P* and select **Keep Offset**).
5. Remember to uncheck the **Deform** option (Properties | **Bone** | **Deform**).
6. You can see all the bones follow when you move the **Root Bone**.



The root bone at the center of the world

Skinning

Skinning is a very important step in the setup of a character for animation that will allow us to deform a mesh parented to a rig. It should be noted that the term skinning is not directly used in Blender. In Blender, you will generally find the term "Weight" designating the influence that a bone has on the geometry. It is often a long and delicate step, but fortunately, Blender allows us to perform an automatic skinning that is already very clean and quick. This is one of the most efficient skinning algorithms.

Before skinning our character, just as a reminder, we must determine which bones will deform the mesh and which not. This will be done as follows:

1. We will verify whether the **Deform** option is unchecked for all deforming bones (**Properties** | **Bone** | **Deform**).
2. In the **Object Mode**, we will select the mesh of the **Rat Cowboy**, then the Armature, and we will make them child and parent (press *Ctrl* + *P* and select **With Automatic Weight**).

A nice skinning has been done for us. You can make a few rotations on the rig in the **Pose Mode** to visualize the result.

However, we must do a few adjustments to improve this. Let's dive into the tools that we have at our disposal.

The Weight Paint tools

If we select our mesh and observe the **Data** menu of the **Properties** panel, we can see **Vertex Groups** that matches the bones of the Armature. This menu, in **Weight Paint** mode, allows us to select and view the influence of each bone. The **Weight Paint** mode could be activated directly on the object. If the armature is in the **Pose Mode**, it is possible to select the bones by a RMB click as well while we are in the **Weight Paint** mode of the object.

In the **Weight Paint** mode, we can view the influence that each bone has on the geometry with a color play. Blue means a 0% influence, a greenish-blue color means 25%, green means 50%, yellow means 75%, orange means 85%, and red means 100%.

In order to modify the influence, we have several brushes that can be used exactly in the same manner as **Texture Paint** and **Sculpt Mode** in the left panel of the 3D viewport. These brushes allow us to paint bones influence directly on the mesh. In our case, we will use only three brushes. The brushes that will serve us for sure will be the **Add**, **Subtract**, and **Blur** brushes.

- The Add brush allows us to increase the weights
- The Subtract brush allows us to reduce the weights
- The Blur brush allows us to soften and mix the weights

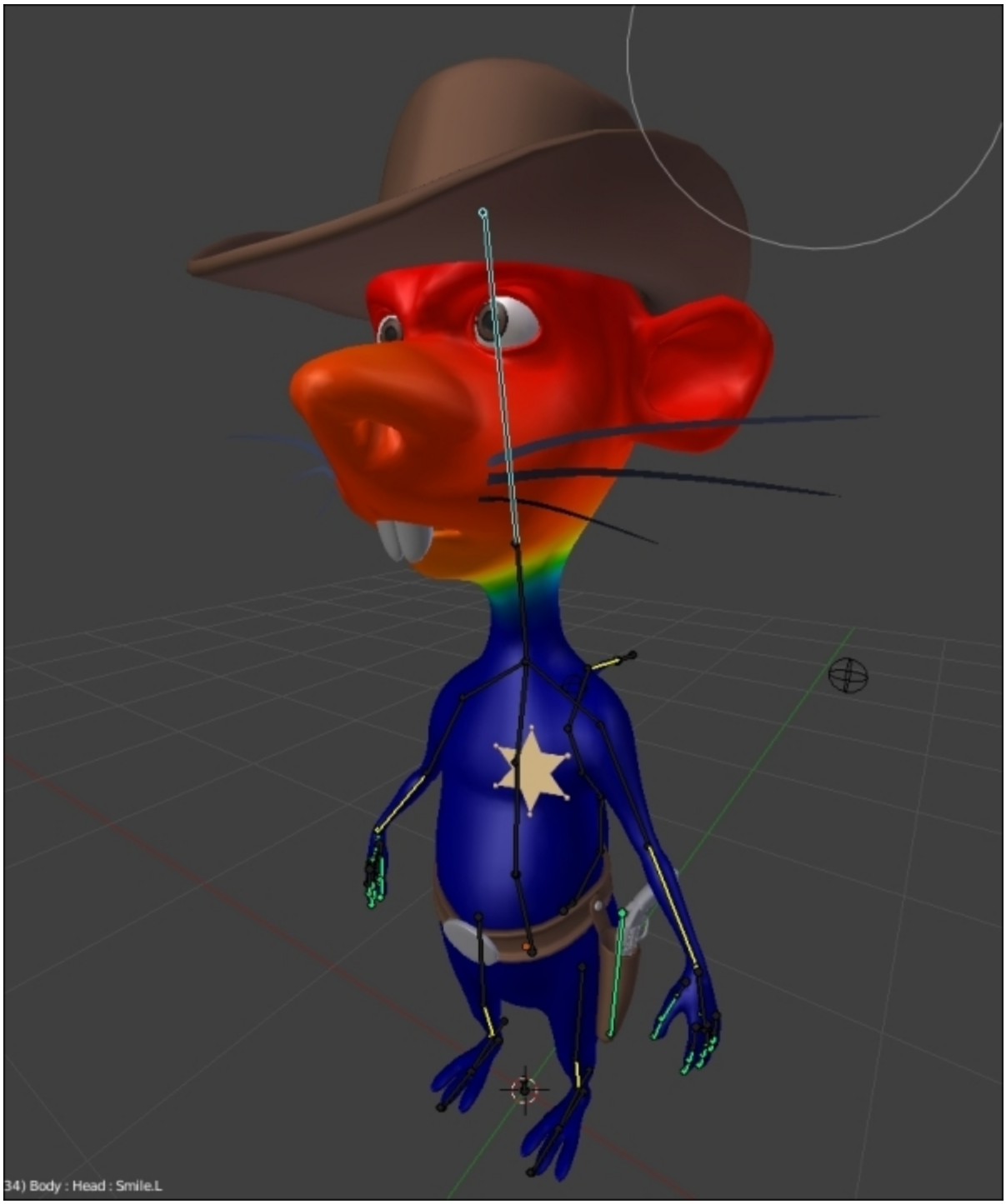
The options of the brushes are exactly the same as with **Texture Pain**t and **Sculpt Mode**. We can change the radius of the brush by pressing *F* and moving the mouse. Also, we can change the strength of the brush that will completely modify the impact of the brush. You can change the curve, too.

We can paint the weight in symmetry. To do this, the mesh must be perfectly symmetrical. We will check the **X-Mirror** option in the **Option** tab of **Left Panel** (*T*).

There are also a few useful options in the **Weight Tools** menu. For example, **Mirror**, to make a symmetric skinning and **Invert**, to invert the influence of our bones.

Note

For more information, you can have a look at the official Blender manual at this address: https://www.blender.org/manual/modeling/meshes/vertex_groups/weight_paint_tools.html.



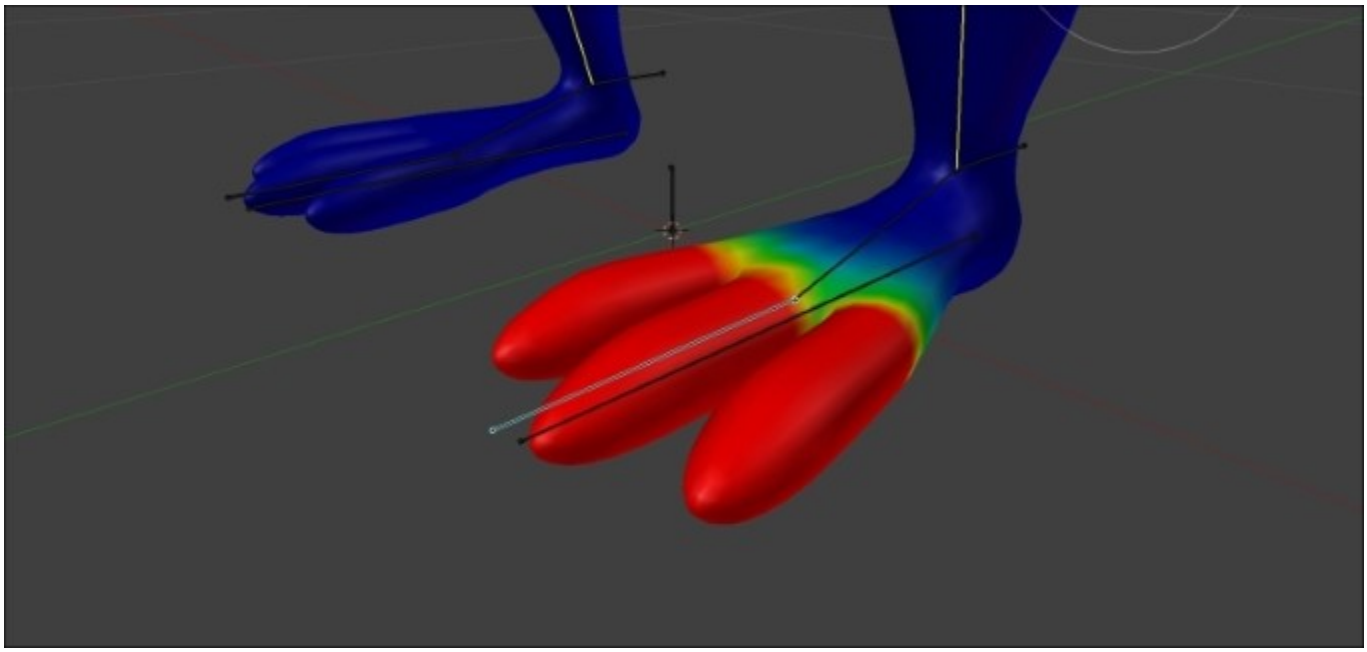
The weight paint of the Rat Cowboy (here the head influence is shown)

Manually assigning weight to vertices

Weight Paint is a very useful technique, but sometimes it is not very accurate. It can be useful to assign weight precisely on some geometry components.

To do this, we must be in the **Edit Mode** and select the vertices to which we want to assign a weight. Then we can go in the **Vertex Group** menu (**Properties** | **Data** | **Vertex Groups**). There will be a **Weight bar** from which we must select the desired weight and click on **Assign**. If you don't have any group, you can click on the + icon.

Another nice feature is located in the **Right Panel** (*N*). There the **Vertex Weight** menu allows us to visualize the bones that influence the selected vertex and adjust the vertex directly. We will notice that the total weights assigned to a vertex group may exceed **1.000**. In fact, Blender will add the total of influences and make an average.



Correcting the weight paint of the toes.

Correcting the foot deformation

If you have used the **With Automatic Weight** option, you should have a very few things to change. The arms, legs, and head deformations should be quite good.

However, the toes aren't working well because there isn't a bone for each toe. So we will fix this as follows:

1. We will check the **X-Mirror** option, and we will select **Toe bone**.
2. Now, we can paint with **Add Brush** to add some weight to the toes until we get a red color (100%).
3. Then, we will need to remove the influence that the foot bone has on the toes. To do this, we will use the subtract brush. Now the foot shouldn't affect the toes.

Note

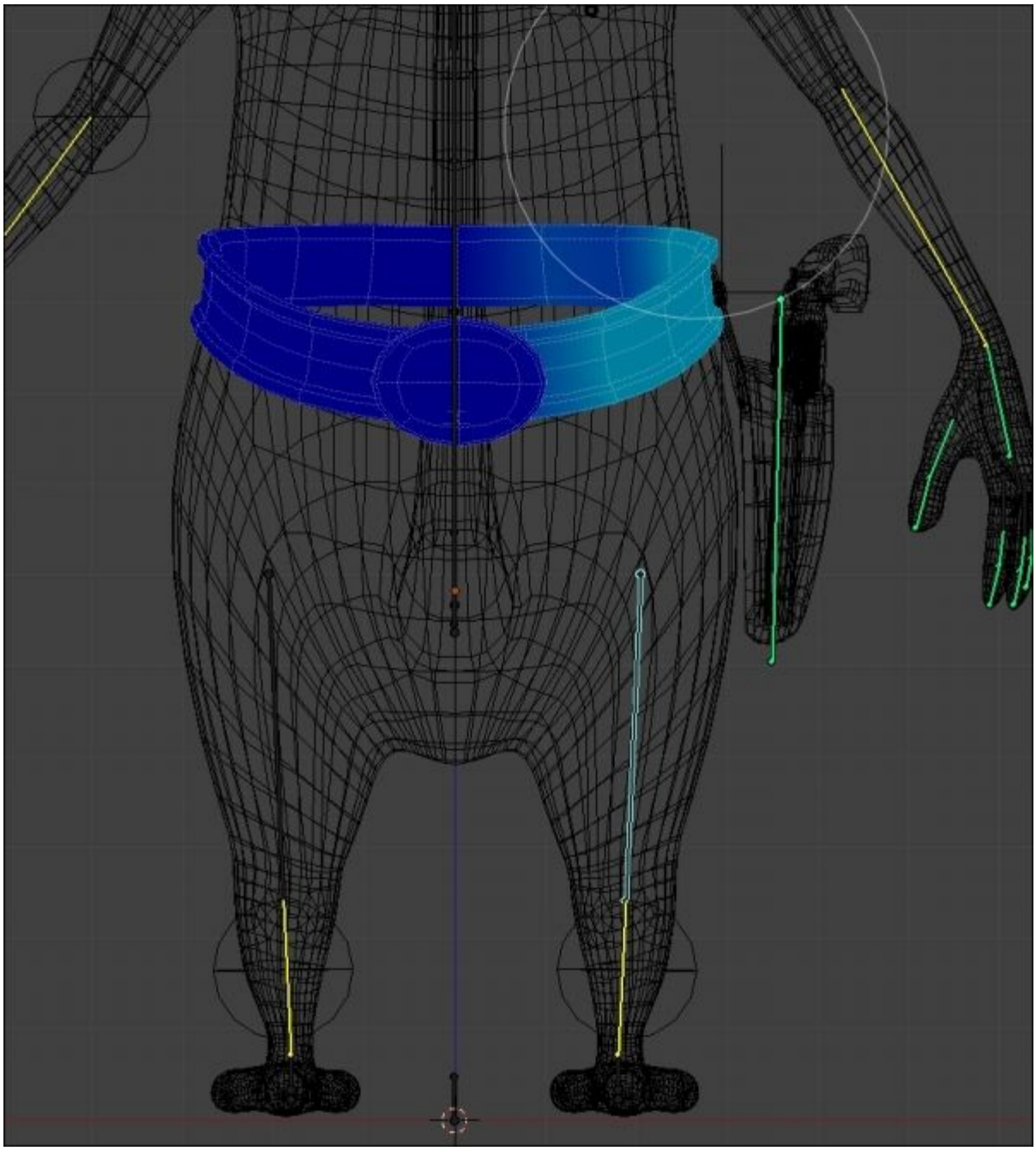
Stick Display for Weight Paint

To easily edit a skinning, it is advised to change the appearance of the bones to **Stick Mode** with the **X-Ray activated** option. You will get better visibility.

Correcting the belt deformation

Let's do the skinning of the belt. It is not a particularly easy element to skin because of its thickness and possible interpenetration with the body, but do not be discouraged. In this case, we will manually assign the weight to the vertices. This is shown in the following steps:

1. We will start by selecting the belt, then select the Armature, and we will make them child and parent (*Ctrl* + *P* and select **With Automatic Weight**).
2. In the **Vertex Groups** menu (**Properties** | **Data** | **Vertex Groups**), we will remove all the vertex groups except these: **Hips**, **Thigh.L**, and **Thigh.R** with the **-** button.
3. First, we will assign a weight of **1.000** to the **Hips** vertex group, so the hips bone deforms all the belt.
4. Then, we will select the vertices on the right half of the belt in the **Edit Mode**, and we will assign a weight of **0.2** to the **Thigh.R** group.
5. We will do the same thing for the vertices on the left half of the belt but with **Thigh.L**.
6. Then we will check the rotations of the **Belt** bone to verify some of the potential problems of transition. We will adjust the weight of a few vertices. The goal here is to create a gradient of weight at the center according to each thigh bone so that the thigh "attracts" some part of the belt in a smooth manner.



The weight of the left-hand side of the belt

Custom shapes

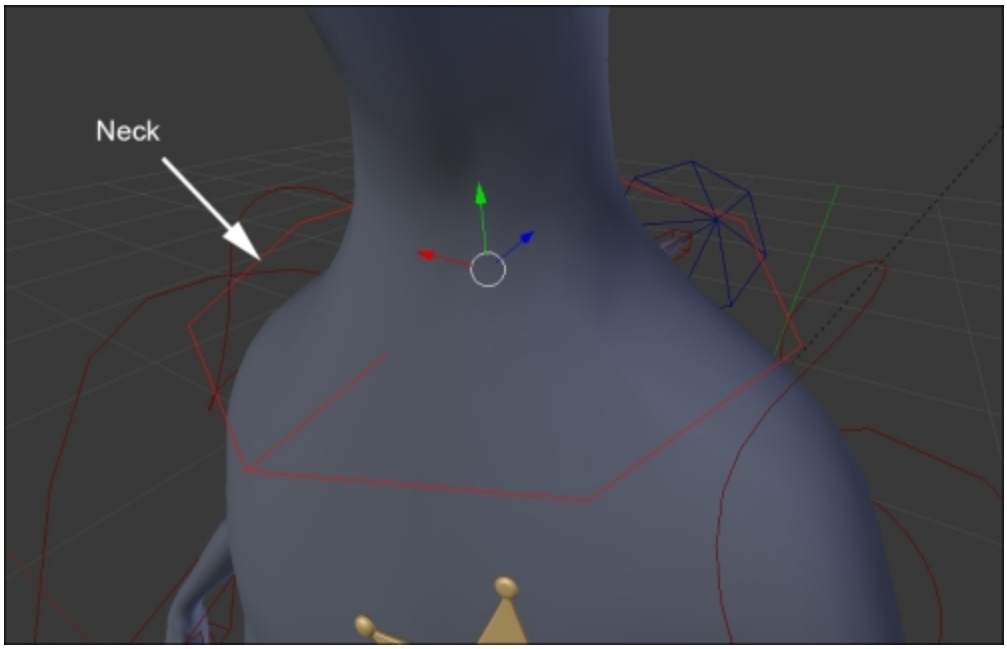
Now that our rigging is almost finished, we can opt for custom shapes. These are 3D objects that can replace the usual look of bones. The purpose is primarily aesthetic, but functional as well. We can make bone shapes that will go around the mesh and allow us to uncheck the X-ray option. Also, it will allow us to manipulate bones more easily.

We can hide the bones that do not require manipulation, such as the arms that are in fact only controlled by **HandIK.L** and **HandIK.R**.

To implement a custom shape, we must first to create a form, usually from a circle. We are going to make the custom shape of the **Neck** bone together and let you do the rest as this is very repetitive:

1. We will add a circle (eight sides is enough) to the scene (press *Shift* + *A* and select **Mesh** | **Circle**).
2. In the **Edit Mode**, we will select the opposite vertices on the Y axis, and we will connect them (*J*). This connection in the custom shape allows us to visualize the orientation of the bone better.
3. We will rename this object as **SHAPE_Circle_01**.
4. Now we will select the neck bone and **SHAPE_Circle_01** in the **Custom Shape** option (**Properties** | **Bone** | **Display** | **Custom Shape**).
5. There is often a problem with the axis of rotation and the scale. These must be adjusted in the **Edit Mode**. We can work on **SHAPE_Circle_01** again in the **Object Mode** without modifying the custom shape applied to the neck bone.
6. Once the custom shape is adjusted, we don't have to see the **SHAPE_Circle_01** object, so we move it to another layer (*M*). We will choose the last layer to the right in our case. This layer is usually used as a garbage layer where we put unwanted objects.

We redo the same process for almost every controller bone. Some shapes are very often used, such as glasses for the eyes, but try to find custom shapes that fit your needs. They must be explicit and made of very few vertices.



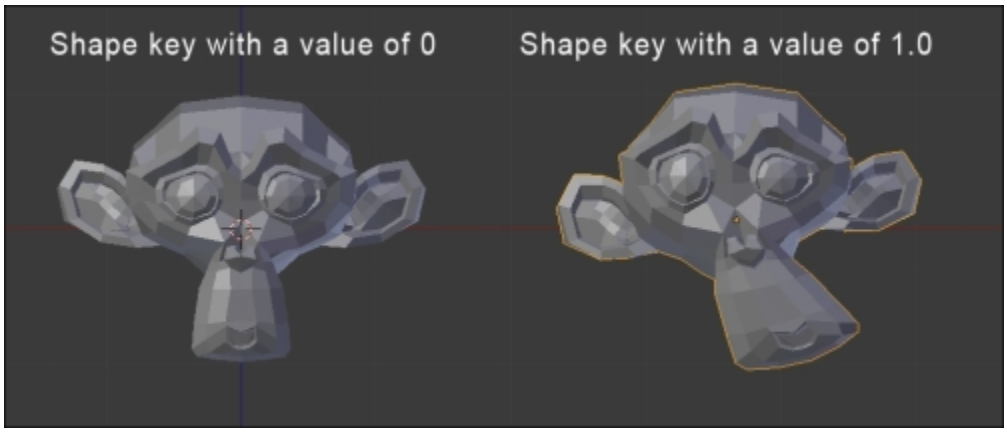
The custom shape of the neck bone

The shape keys

In the following sections, we are going to learn about shape keys and drivers. This will help us to create some very basic facial controls that we will use in the next chapter. Facial rigging is a long process, so we are not going to create a fully functioning facial rig here, but you will have all the tools needed to create your own if you want to.

What is a shape key?

A shape key is a method to store a change of geometry in a mesh. For instance, you can have a sphere object, add a shape key, move your vertices so that the sphere looks like a cube, and the shape key will store the changes for you. A shape key is controlled by a slider. The value of the slider corresponds to the distance each vertex has to move to get to the stored positions. As you can imagine, shape keys are very useful for facial rigging as they allow us to create different expressions and turn them on or off.



Example of a shape key with Suzanne

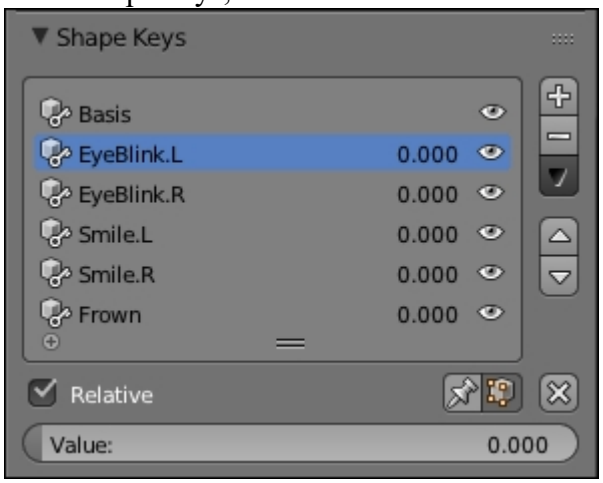
Creating basic shapes

In this section, we will create five basic shape keys. They are eye blink left and right, smile left and right, and frown. This will be done as follows:

1. First, we will need to select the object that will receive the shape keys. Then we will go to the **Object Data** tab of the **Properties** editor, and we will open the **Shape Keys** subpanel.
2. When creating the shape keys, we always need to have a reference key that will store the default position of each vertex. To do this, we will click on the + button. It is called **Basis** by default.
3. Now we can add a new shape key with the + icon and name it as **EyeBlink.L**. As you can see in the following, there is a **value** slider. At **0**, the shape key is turned off, so change the value to **1.0** in order to view your changes in the **Edit Mode**.
4. Now in the **Edit Mode**, we can change the left eye geometry to close the eye. You can use the **Connected Proportional Editing** tool (*Alt* + *O*) in order to smoothly move each eyelid to the

center of the eye. Beware, you only want to move the eyelid's geometry, otherwise any other changes done will be part of the shape key.

5. Next, we will mirror the shape key without tweaking our geometry on the right side again. We can do this because our mesh is perfectly symmetrical. We are going to create a new shape key based on the one that is currently active in the shape key stack. The value of **EyeBlink.L** is still at 1.0, so we can click on **black arrow** under the **–** icon and choose **New Shape From Mix**. Now, we have a new shape key with the same information as **EyeBlink.L**, but this not what we want. We will need to mirror this to the other side. To do this, we will again click on **black arrow,** and we will press **Mirror Shape Key**. We can also rename this as **EyeBlink.R**. Now if you change the value of **EyeBlink.R**, you can see that the right eye of the Rat Cowboy is closing perfectly. What a huge time-saver!
6. The next two keys, **Smile.L** and **Smile.R**, are created using the same method. What you need to do is create a nice smile on one side and mirror it using **New Shape From Mix** and **Mirror Shape Key**.
7. The last key to be created is a frown. To do this, we are just going to create a new key with the + icon and move the geometry between the eyebrows. There are tons of other things to know about shape keys, but for now this is all we are going to need.



Our facial shape keys

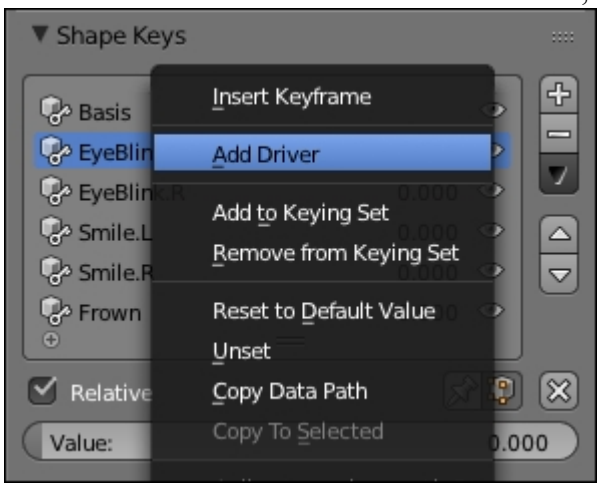
Driving a shape key

You might be thinking that animating directly shape key from the sliders is not very practical, and you are right! This is why we are going to use drivers. It is a method to indicate to an entity (object, bones, and so on) to control another value. In our case, we are going to tell Blender that the sliders of our shape keys are going to be controlled according to the transformations of the bones located on the face. This will give an impression that we are directly manipulating the head of our Rat Cowboy.

1. The first thing to do is add three new bones to our **Armature** object in the **Edit Mode**. The first one will be located between the two eyebrows, and this will control the frown. The others are going to be near the mouth corner and will be symmetrical. They are called **Frown**, **Smile.L,**

and **Smile.R** respectively. In order to control the eyes, we are going to use the bones we've used previously as targets.

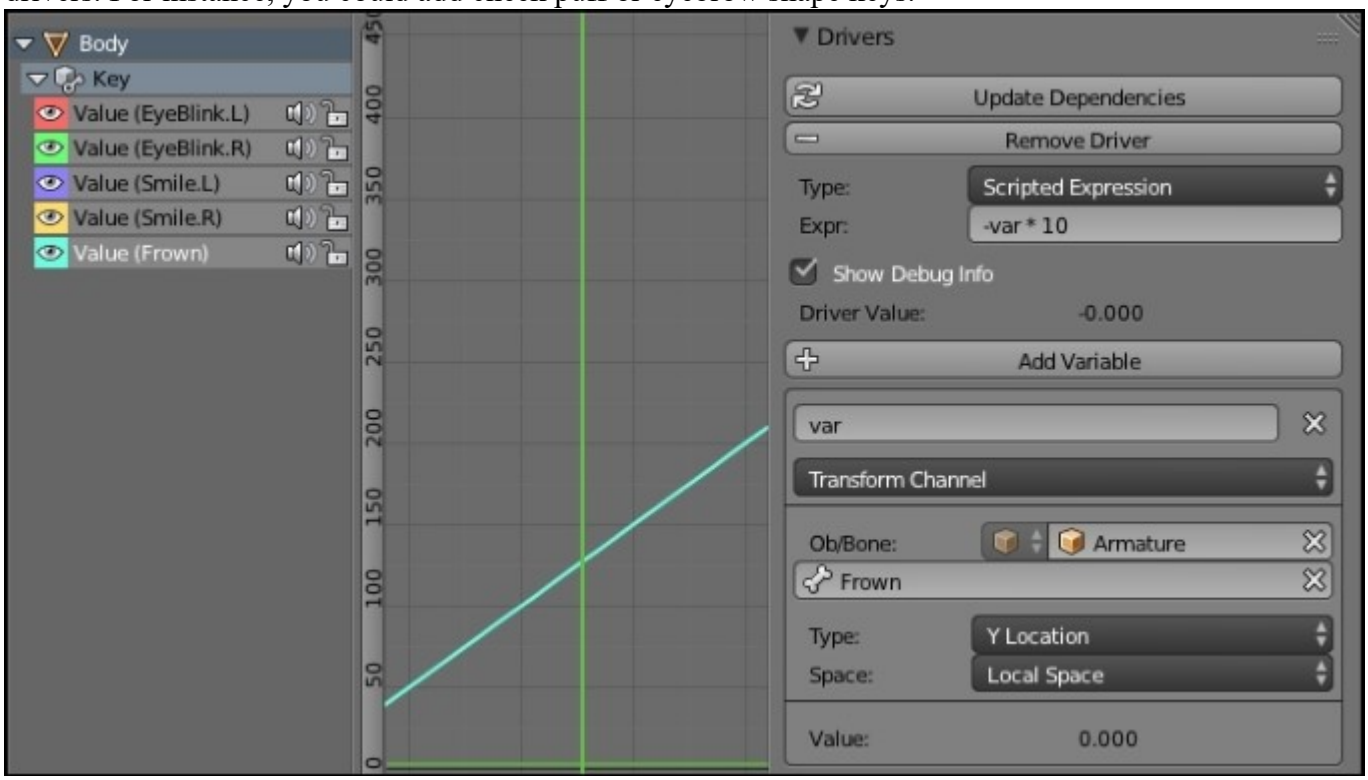
2. Now we will add a driver to the **EyeBlink.L** shape key by right-clicking on the value slider. Note that the value slider is now inaccessible, so it will be driven by another entity.



Adding a driver to a shape key

3. Now split your view in a new editor of the **Graph Editor** type. Switch the mode from **F-Curves** to **Drivers** in the Graph Editor's header. As you can see, if you still have the rat mesh selected, you will have a new key on the left-hand side of **Graph Editor**. We can click on the white arrow on the left-hand side to unfold the group of keys, and we can select the **EyeBlink.L** one.
4. Now we will open the right panel of **Graph Editor** (*N*), and under the **Drivers** subpanel, we can change the different settings. First, if you have an error, you can go to the user preferences under the **File** tab and check **Auto Run Python Scripts**. Drivers work with Python internally, so you must accept that it runs scripts. But don't be afraid, we are not going to code here!
5. The first thing that we will change in our settings is the **Ob/Bone** field. We are going to choose the **EyeTarget.L** bone. So first choose the **Armature** object and then the bone. This will tell Blender that the target will be the bone that affects the driver.
6. Now we are going to set **Type** to **Y Scale** and **Space** to **Local**. This means that, when we scale our target, the value of the variable called **var** in the upper field will take the value of **Y Local scale** of the bone.
7. Now you need to imagine that the field called **Expr** is directly linked to the value slider of the shape key. You can even test this. If you enter `0`, the eye will open, and if you enter `1`, the eye will close. So now what we can do is replace this field with the **var** variable that holds the value of the y local scale. You can now see that the Rat Cowboy's eye is closed. But if you scale it, it will open. We are close to the required result.
8. The last thing to do is change the expression so that it is open by default. We know that the **Y local scale** of our bone is **1** by default and that the "open state" of the eye corresponds to the **0** value of the shape key slider. So we can add the **1-var** expression to the **Expr** field. If the Y local scale of the bone is 1, we will have **1-1 = 0** and the slider value will be 0, so the eye will be open. If the **Y local scale** of the bone is 0, we will have **1-0 = 1 and** the slider value will be 1, so the eye will be closed. Done!

9. We can replicate the same process with the other eye. In order to save time when creating the driver, you can simply right-click on the **EyeBlink.L** value slider, press **Copy Driver,** right-click on the **EyeBlink.R,** and choose **Paste Driver**. At this point, you just need to change which bone is controlling the key in the driver settings.
10. For the smile driver, we will do a similar thing. But instead of feeding the variable with the Y local scale of the eye target bone, we are going to use the local Y location of the **Smile.L** bone. If you see a need to move the bone a lot in order to change the shape key, you can simply multiply **var** time a scalar in the **Expr** field. In our case, the expression will be **var * 5**.
11. We can do the same thing for the other **Smile.R** bone and for the frown. The expression of **Frown** is **-var * 10** in our case. As you can see, we have negated the **var** variable because we want to move the **Frown** bone down in order to control the shape key.
12. As a bonus, we are going to lift the hat with the frown. To do this, we will copy our Frown driver to the X rotation of the hat object in the right pane of the 3D view (*N*). We will adjust the expression. In our case, the expression is simply **-var**.
13. We will need to add a **Limit Location** constraint to the Frown bone as well. This will provide the animator with the ability to lift the hat high or low by locking the bone between a minimum and a maximum value on the Y axis in local space. To do this, we will change the settings of the constraint. We will check **Minimum Y** and set the value to **-0.115**, and then check **Maximum Y** and set its value to **0**. Also, we can provide moves on the X and Z location by checking the other check boxes and setting their value to **0**. Then we will need to select the **Local Space**.
14. Now, we have minimum control over the Rat Cowboy's face in order to start animating our character. We advise you to go further yourself in order to gain experience with shape keys and drivers. For instance, you could add cheek puff or eyebrow shape keys.



The setup of the Frown driver in Graph Editor

Summary

In this chapter, you've learned how to create a simple bipedal rig. All the knowledge that you've acquired could be used to push this further. For instance, you can add an FK/IK switch slider for the arms and the legs, add more facial controls, or create a more complex foot roll. If you want to see or use a more complex rig, you can check the rigify add-on (integrated in Blender by default). However, this rig will be quite interesting for animation. Talking about animation, let's start using our rig in the next chapter!

Chapter 9. Rat Cowboy – Animate a Full Sequence

This chapter will be devoted to the animation of a full sequence. We will begin our journey by discovering the 12 animation principles. Then, we will learn more about the preproduction stage that is all the things that we need in order to prepare the animation such as the writing of a script and the creation of a storyboard. After this, we will learn some important tools in order to animate in Blender such as the Timeline, Dope Sheet, Graph editor, and NLA. Next, we will create a layout that is a rough 3D visualization of the sequence without animation. After we've done all this, it will be time to start animating our shots. We will first learn how to animate a walk and use the NLA in order to mix actions together. We will then animate a close shot and a gunshot inspired by old western movies. The graph editor will be used extensively in order to animate a trap. Finally, we will learn how we can render a playblast of our shots. So let's dive into the wonderful world of animation where things start to move!

In this chapter, we will cover the following topics:

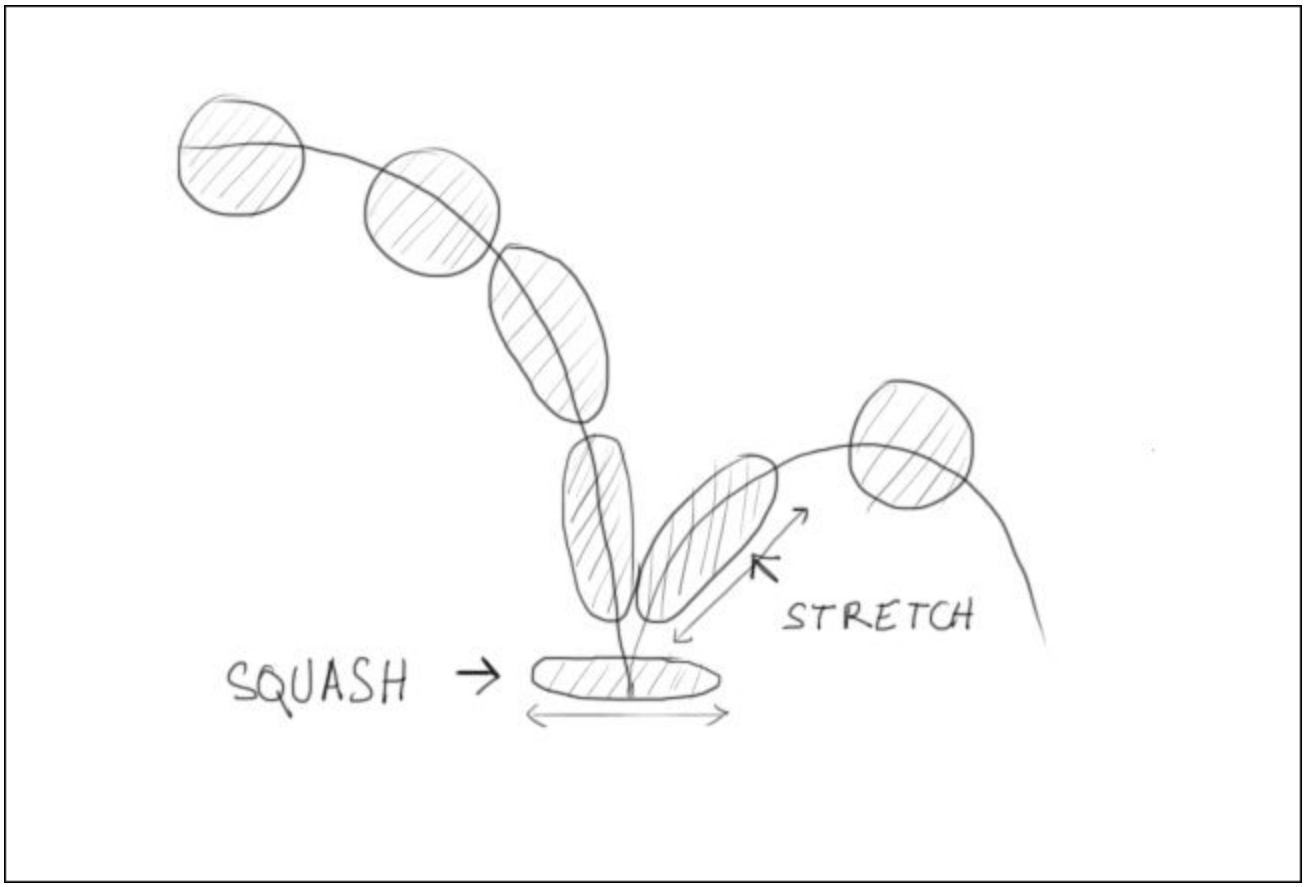
- Learning the principles of animation
- Preparing our animation with a script, storyboard, and layout
- Using the Blender animation tools
- Animate different shots
- Render a playblast

Principles of animation

In order to start to animate with Blender in the best way, it is important to understand some basic principles defined in the 80's by Ollie Johnston and Frank Thomas. These principles are inherited from the 2D animation art called "traditional animation". Animation involves recreating the illusion of motion by a sequence of images. Most of these principles also work for 3D animation. Here, they have been developed for a cartoon style, quite far from realistic movements. So, we don't have to apply them to any situation, but they still contain the secrets of animation.

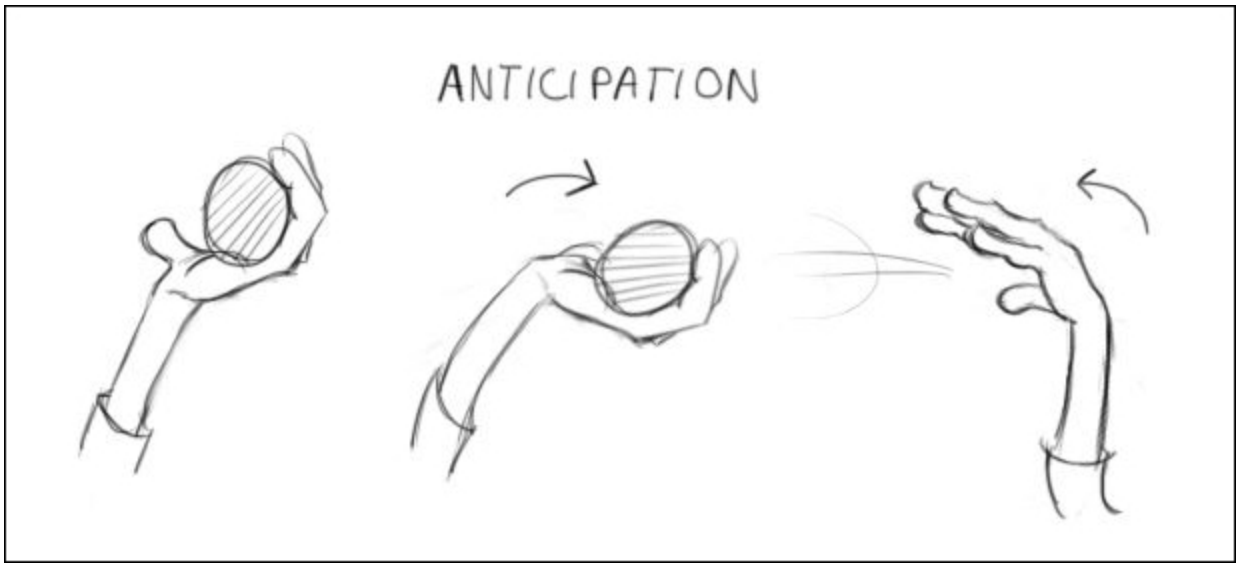
Squash and Stretch

This is one of the common principles that applies to cartoon-style animation. The goal is to over-exaggerate the effect of inertia and elasticity on a particular object. From a 2D perspective, it's quite hard to manage because the object doesn't need to lose its volume, so we need to judge the shape by eye, but in 3D this is just a matter of a good rig.



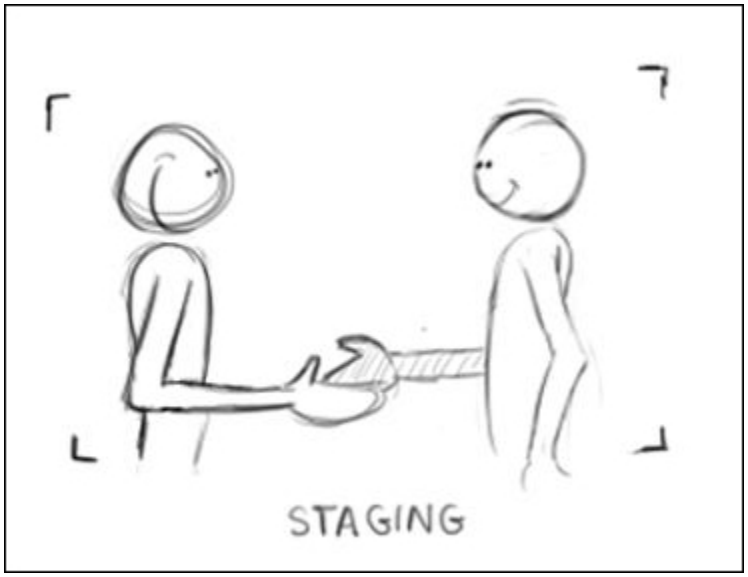
Anticipation

The principle of Anticipation describes the anticipation before an action. For example, a character ready to jump is going to bend the knees, the back, and the arms before the actual jump. It is important in order to add realism to the action you want to depict through your animation. To better understand this principle, try punching with your hand, you'll see your hand going back first.



Staging

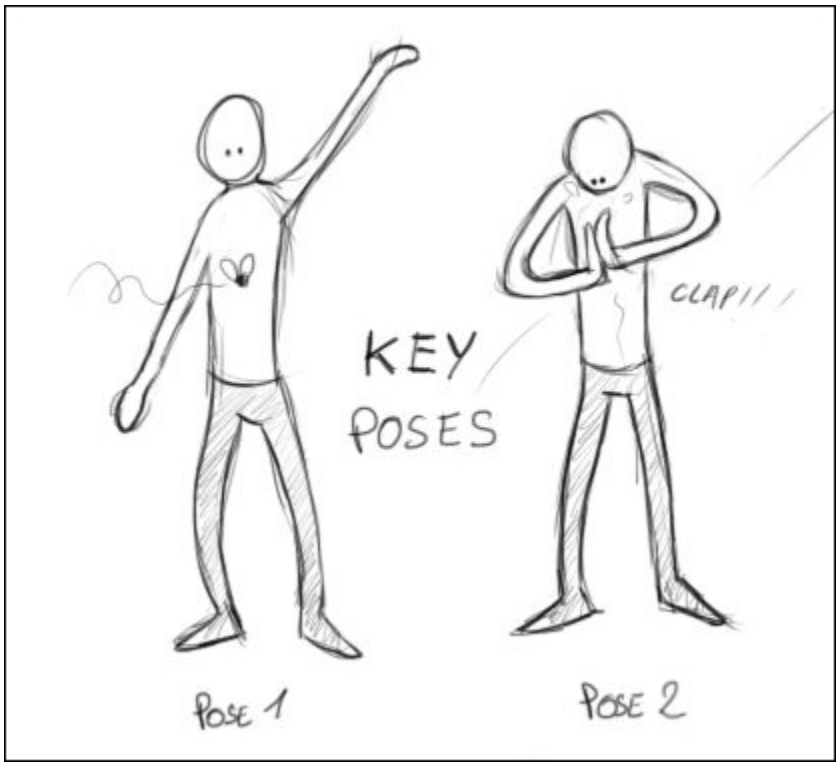
This principle is applied in cinema and theater to retain the attention of the spectator on specific elements and remove every useless detail from their view. This is useful to communicate a perfectly clear idea. This may include many areas of animation such as lighting, acting, or camera positions.



Straight Ahead Action and Pose to Pose

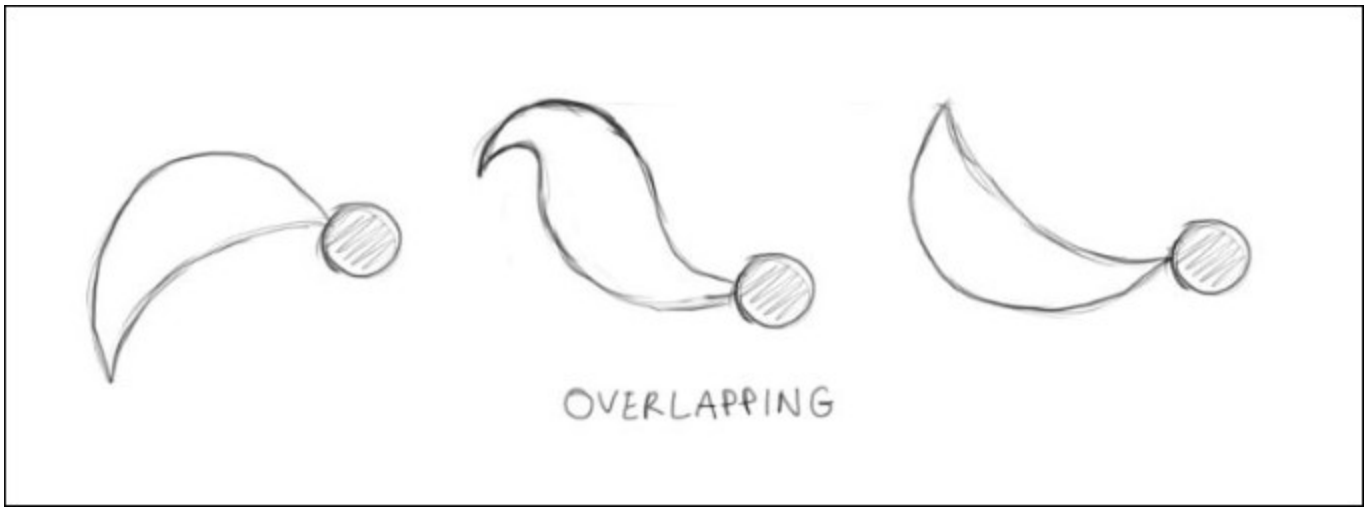
These are two different approaches while animating. The **Straight Ahead Action** method consists of making an animation gradually frame by frame from the beginning to the end. With 2D animation, this is especially useful to create special effects such as fire and water, and this allows improvisation.

The **Pose to Pose** method allows us a much better control of the timing and is easier to manage. This is often much more efficient when we animate characters. For this method, we start by adding the **Key Poses**. They are the main keys that indicate the action. Then comes the **Extremes** that we add to the extremities of motions that exaggerate the action between two key poses. Then, there are the **Breakdowns** that are the main intermediate keys of the action between the extremes. These add more fluidity to the action. A mix of the both methods is often used.



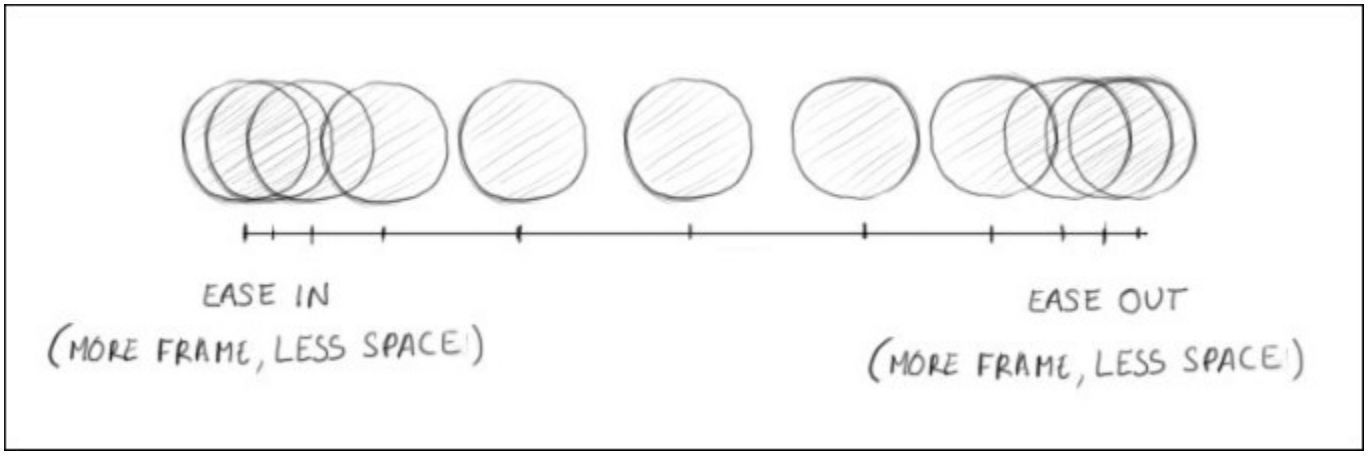
Follow Through and Overlapping Action

Follow Through and Overlapping Action techniques being very close consist of giving some inertia to an animated object and add a better sense of realism. **Follow Through** consists of continuing the movement of a part of an object after it has stopped moving. This can be applied to a tail, for instance. **Overlapping Action** consists of creating an offset between a movement of an object and a part of this object. For example, long hair moves at a different speed than the head.



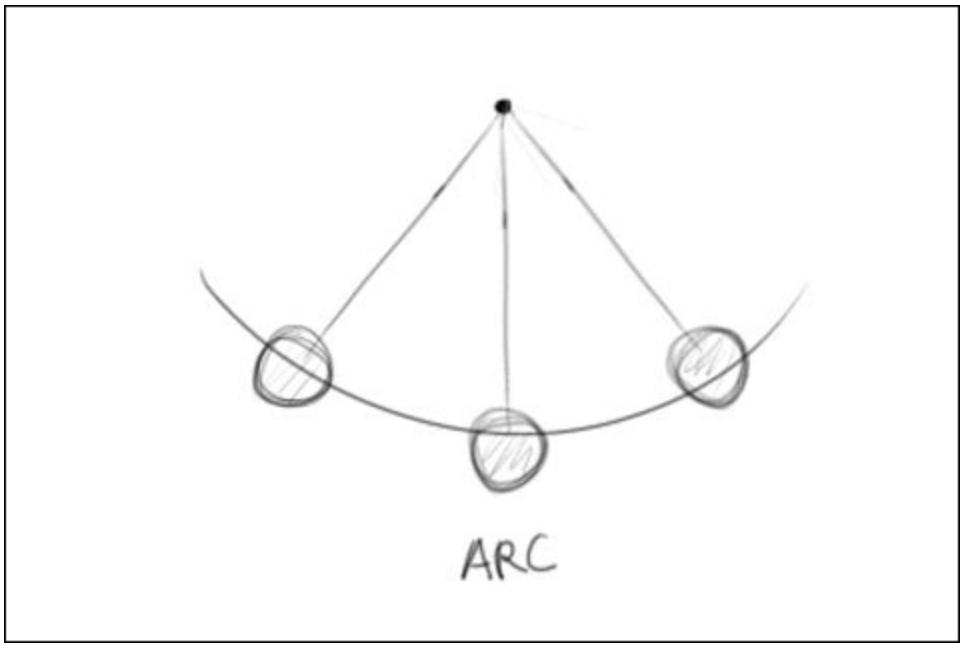
Slow In and Slow Out

These are two effects that consist of attenuating the speed of a moving object around the extreme keys. A Slow In effect is a deceleration at the beginning of an action, and a Slow Out effect is a deceleration at the end of an action. In Blender, these effects can be seen in the form of Bezier curves and are controlled by their handles in the Graph Editor.



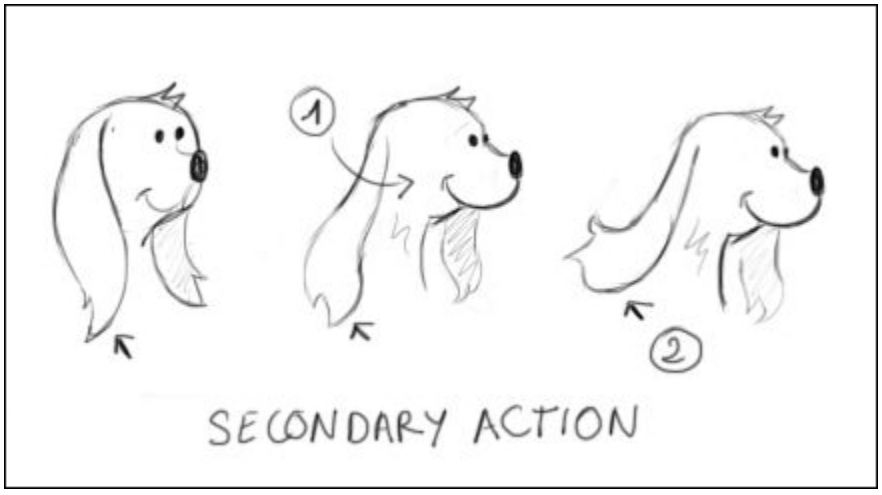
Arcs

This is a principle that consists of creating movements that follow an arc trajectory. Almost every motion follows this principle. For instance, when a character throws a ball, his or her hand follows an arc trajectory. If you take a pendulum and fix the ball, you will see that it follows an arc.



Secondary Action

The Secondary Action principle is about how every little action adds dimension to the main action. For example, in the case of a facial expression, a blink of the eyes can add more expression to the character. Another example could be the ears of a dog that hangs when he turns his head.



Timing

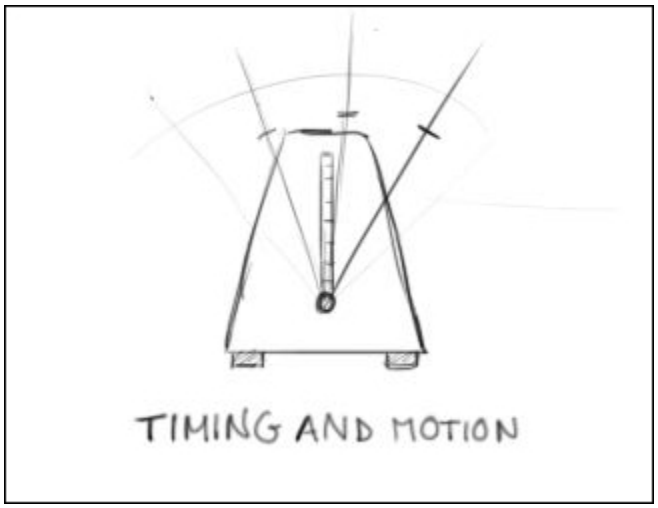
The number of frames between the beginning and end of an action directly affects the speed of the action. Timing is related to a physical consistency respecting certain laws of physics. In order to animate

a character, timing defines its personality and emotions. For example, if a character is sad, timing is going to be definitely slower.

Note

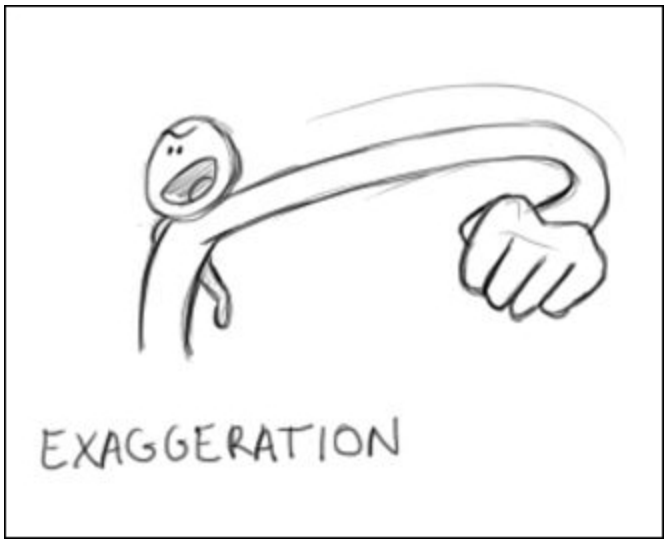
24 images per second

The frame rate in cinema is 24 images per second, and this is enough for the eye to have an impression of fluidity. This frame rate is used in the basic parameters of Blender.



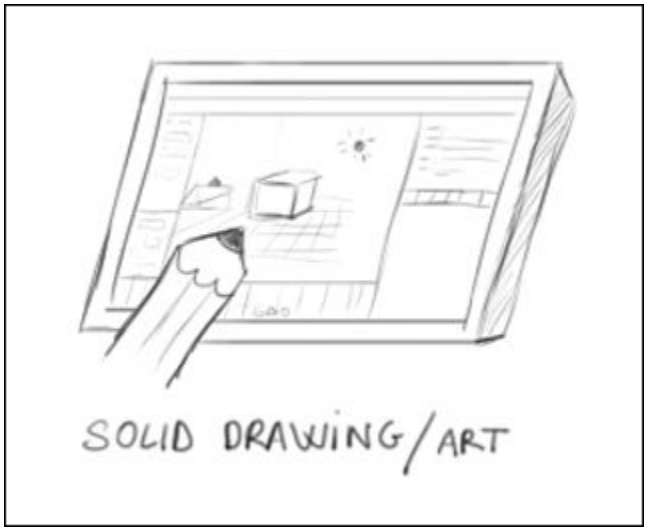
Exaggeration

The exaggeration principle is used when you want to accentuate the action that you are transmitting to the viewer. The action can be a pose, a gesture, or an expression. Sometimes it is better to slightly get away from realistic animation to get a better impact. For instance, if you want to accentuate a punch, you can exaggerate the proportions of the arm of a character.



Solid drawing

This principle mostly applies to 2D animation. It consists of ensuring certain realism in drawing regarding volume, weight, and balance. This advocates paying attention to volume and perspective in order to avoid a flat render. A character must be possibly seen from any angle of view. In 3D animation, it is a bit different. Even for a 3D animator, drawing can be seen as a strength, because it allows to quickly put the poses you have in mind on paper, but this is not going to be your main tool.



Appeal

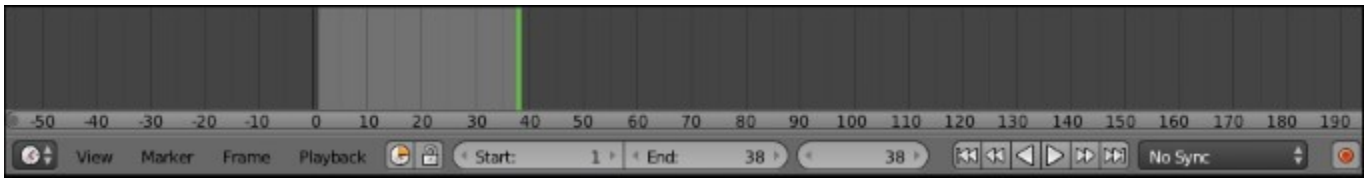
This principle is mainly about the interest and appearance of the characters you will animate. It means making a dynamic design through shapes, colors, proportions, gestures, and personality. It can be a hero or a villain, whatever.

Animation tools in Blender

Now that you understand a little bit more about what animation involves, we are going to dive into the different tools that we are going to use when animating our sequence. In order to test these tools, we are going to use the default cube of a new .blend file.

The timeline

The timeline editor gives us a lot of information about the animations of our file. The timeline represents each frame as you can see in its lower part. You can navigate to any frame by dragging the green bar. Using the Left and Right arrows, we will move one frame at a time. Using *Shift* and Up or Down arrows, we will move by an increment of ten frames. In the header we have two sliders, Start and End, which respectively represent at which frame the animation starts and ends. As you can see, we have dark grey parts in the timeline that visually shows this range. If you want, you can set the start and end frame by placing the timeline bar where they need to be and by pressing *S* and *E* respectively. You can quickly move the timeline bar to the start and end frame by pressing *Shift* and left or right mouse buttons. You can also zoom in to the frame with the Mouse Wheel. You can use the well-known play, rewind, and pause buttons in order to play the animation. The shortcut to play the animation is *Alt* + *A*.



The timeline

What is a keyframe?

Now it's time to learn how to set keyframes. But wait a minute, what is a keyframe? It is used to store the state of an object (or any other animatable thing) at a certain frame. Let's add a keyframe to our default cube:

1. We will open a fresh new blend file, and we will select the default cube.
2. We will then place the timeline bar at frame 0 and press *I* to open the **Insert Keyframe Menu**. We can now choose between many options, but we are going to choose the **LocRotScale** one because we are going to keyframe the location, rotation, and scale of our object.
3. Now we will go to frame 20 (remember the *Shift* + Up arrow shortcut of the timeline), and we will move, scale, and rotate our cube. Now we can add a new key to store the current state of the cube, press *I* and select **LocRotScale**.
4. If you move the timeline bar or play the animation, you can see that Blender automatically interpolates the motion for you! You can clearly see that you have two keyframes represented in yellow. Congratulations, this is your first animation in Blender.

Note

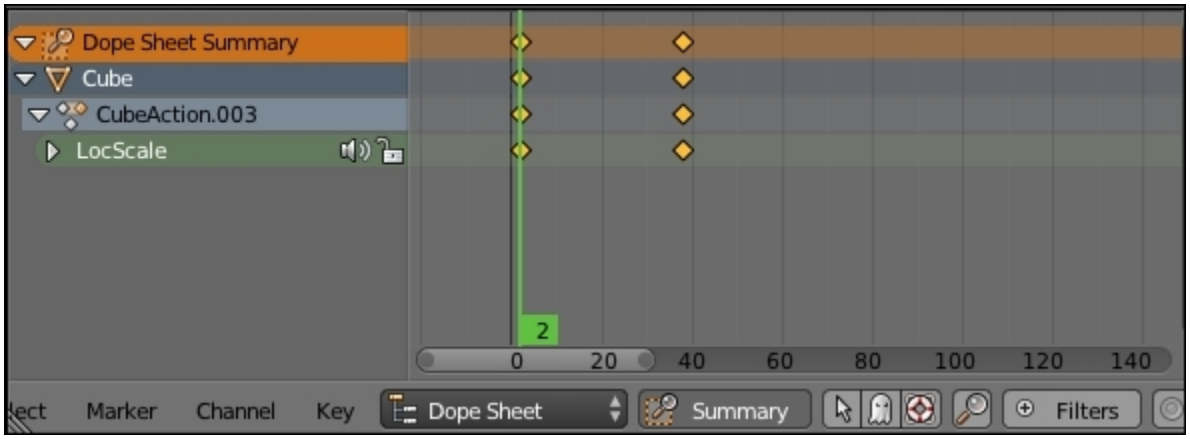
The AutoKey option

In the timeline header, we have a nice little red circle button that allows us to automatically create a keyframe as soon as we change the location, rotation, or scale of our selected object. Thus we don't have to call the Insert Keyframe menu every time.

The Dope Sheet

The timeline is great but is not very useful when we want to manipulate our keys. To do so, we can use a more robust editor called **Dope Sheet** (as always you can split your view and select the editor type that you want on the left-hand side of the header). On the left-hand side, we can see every object that has keys on it. In the header, we have many important options such as the **Show only selected** button (a mouse pointer icon) that tells Blender to only show the keyframe of the selected objects. The keys are represented by a diamond shape and can be selected with by right-clicking on them or by using the *B* key for the box select tool. Of course, you can select all the keys by pressing *A*. You can move the keys with *G*. You can also scale a group of keys with *S*. In this case, the timeline bar will be the pivot point. The **Dope Sheet Summary** row enables us to select every key that is below the corresponding key.

For instance, if we select the dope sheet summary key at frame 0 by right-clicking on it, we will automatically select every key on frame 0.



*The **Dope Sheet** editor*

The Graph editor

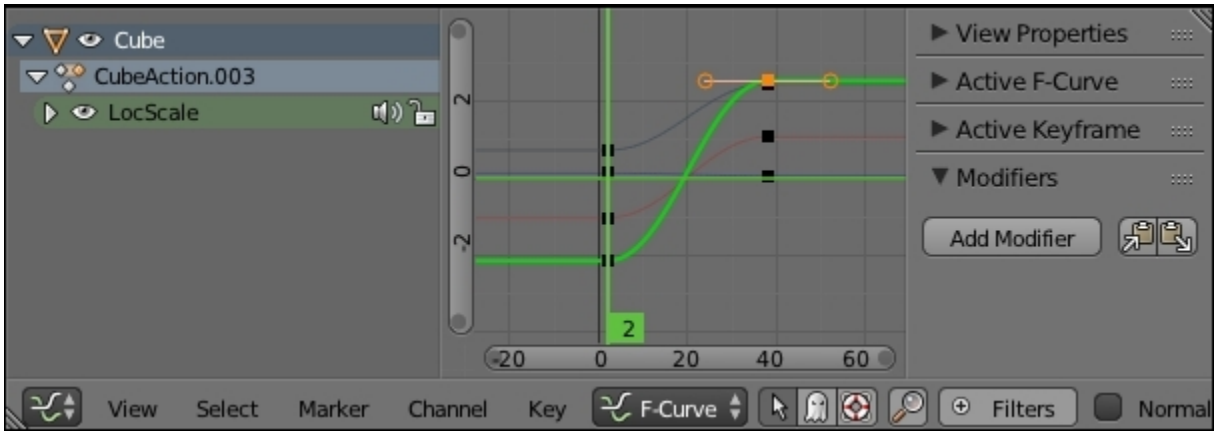
Now we will learn a little bit more about curves with the **Graph** editor. This is a really important editor to learn in order to become a good animator. It mainly allows us to control the interpolation between the keys. It looks like graph paper with the *y* axis corresponding to the value of the key and the *x* axis to the time. We can select the keys in the same way as the Dope Sheet. As you can see on the left-hand side, we still have the objects on which we added the keyframes. We can also see that we have the type of the

keyframe that we've placed; **LocRotScale** in the case of our cube. If we open this with the white arrow, we can see each transformation and its corresponding curve on the right-hand side. Each frame has a handle type that you can change with the *V* key. It looks a little bit like the curve handles that we saw in the Haunted House project. We can set them as **Vector** to have a linear interpolation, for instance. We can also change the interpolation type of each frame by pressing the *T* key. This is a very useful menu that we are going to use in order to quickly animate the bounce of our trap. If you press the *N* key, you will see a bunch of options. You can even see modifiers that allow us to add procedural effect to our curves, for instance, noise. Note that you can use the *Ctrl* + MMB shortcut to squash the graph.

Note

Default interpolation setting

In **User preferences**, in the **Editing** tab, we have the ability to choose the default type of interpolation that we want between two keys. Many animators like to set it as constant so they don't have any interpolation. When the animation plays, it looks like stop motion. After they have the right poses, they select all their keys in the graph editor, and they press *V* to set the Bezier interpolation type back in order to polish the animation.



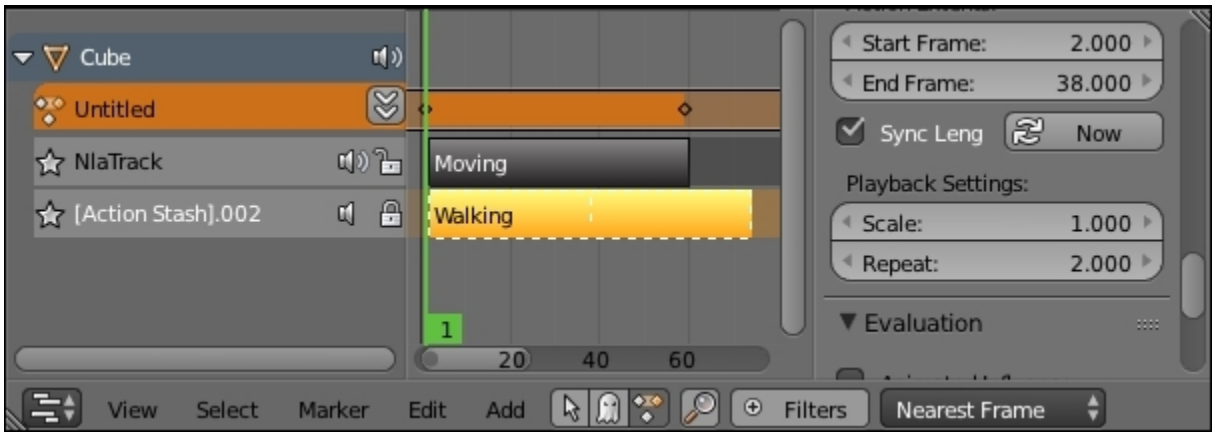
The Graph editor

The Non-Linear Action editor

The **NLA** or **Non-Linear Action** editor is a huge time-saver tool! It allows us to edit actions. Similar to doing video editing, you will edit tracks on which there are actions. You can see an action like a box of keyframes that represent a certain motion of an object. Actions are located in the **Dope Sheet** editor. In its header, we can see a drop-down menu. We can change it from **Dope Sheet** to **Action Editor**. Near this drop-down menu, we have a text field with the name of our action. We can create multiple actions by duplicating them with the + button. Each action has its own set of keyframes. So, for instance, you could animate the walk of a character in a specific action, its displacement in another action, and mix them back together in the NLA editor. Back to the NLA editor, we can press *Shift* + *A* in order to add a new action to a specific track. Tracks are represented on the left-hand side. We can add new tracks by

going to the **Add** menu in the header and by choosing the **Add Tracks** option. We can open a hidden panel by pressing *N*. In this panel, you will have many options concerning the selected action.

For instance, we can repeat the selected action by changing the **Repeat** value under the **Action Clip** subpanel. We can also move actions by pressing *G*. This also allows us to move an action from one track to another.



The NLA editor

Preparation of the animation

Before starting the creation of the sequence, it is important to plan what we are going to do.

Writing a short script

We start by organizing our ideas with some brief writing work. We must describe the scene to be animated shot by shot. We can be creative at this moment of the process and imagine any kind of place and situation.

In the first part, we will put some useful information such as the title, exposure (for instance, Out-Day in order to indicate that the action happens outdoors during the day), and the number of the sequence. In our case, there is only one sequence, so we call it **Sequence 1**. This kind of information is usual in a movie script.

The action of the sequence is in the desert. In a very warm and dangerous place, our rat sees a trap after a very long walk. This trap seems to be there for him.

We will follow a rather traditional script structure. There is an initial situation, a disruptive element that happens, and then the fall. In the case of a short animated film of one minute, there is no time to introduce and develop the characters and an enigma. We must go straight to the point.

For our short film, we will do a staging composed of different camera shots, which will require an editing step later. But we must conceive it now. Maybe you know that cinema has a visual grammar that is expressed by editing. It allows us to make sense of the different shots. It is something that you can learn, and there are certain rules to understand. Going deep in this area can only be a huge advantage for your 3D projects.

In order to write a script, we must describe our shots. There are different types of information. These are the field sizes of a shot:

- **The extreme long shot**: This is used for panoramas.
- **The long shot (or establishing shot)**: This allows us to introduce a situation.
- **The full shot**: This frames the characters entirely. It is great for large movements.
- **The medium shot**: This frames the chest and the head of the character.
- **The American shot (or ¾ shot)**: This frames a character from the thighs to the head. It is close enough to a medium shot.
- **The close-up shot**: This frames the face of the character, and it allows us to perceive emotions better.
- **The Italian shot** (or extreme close up): This frames the eyes of the character.

There are also the angles of a shot:

- **The low angle shot**: The camera is low and the frame upwards. This will enhance the character.
- **The high angle shot**: The camera is high and the frame downward.
- **The aerial shot**: This frames the scenery viewed from the sky.

The camera can also rotate with a pan or be mobile. We often speak of a **tracking shot**. In cinema, the camera is often mounted on a camera dolly or a steady cam for a perfect smooth shot. There are other types of shots and framing, but these are the main types you should know in order to express yourself.

Title : Rat Coybow

Outdoor Day

Sequence 1 :

Shot 01 : Long shot of the background. The character moves forward and the camera makes a lateral tracking shot. The character stops walking in front of the camera. He frowns and seems to observe something away.

Shot 02: Medium shot of the cheese placed on a trap.

Shot 03 : Full shot on the side. We see the position of the rat and the trap.

Shot 04 : Close up of the eyes of the rat.

Shot 05 : Close up of the cheese. slight zoom.

Shot 06 : Close up of the hand preparing to take the gun located above the holster.

Shot 07 : Close up of the cheese with the trap.

Shot 08 : Italian shot on the side. The rat waits a moment and shots with his gun.

Shot 09 : Full shot. The cheese is projected behind the trap that closes. Tracking shot focus on the rolling cheese.

Shot 11 : Close up of the eyes of the rat. He is smiling.

The storyboard

Making a storyboard

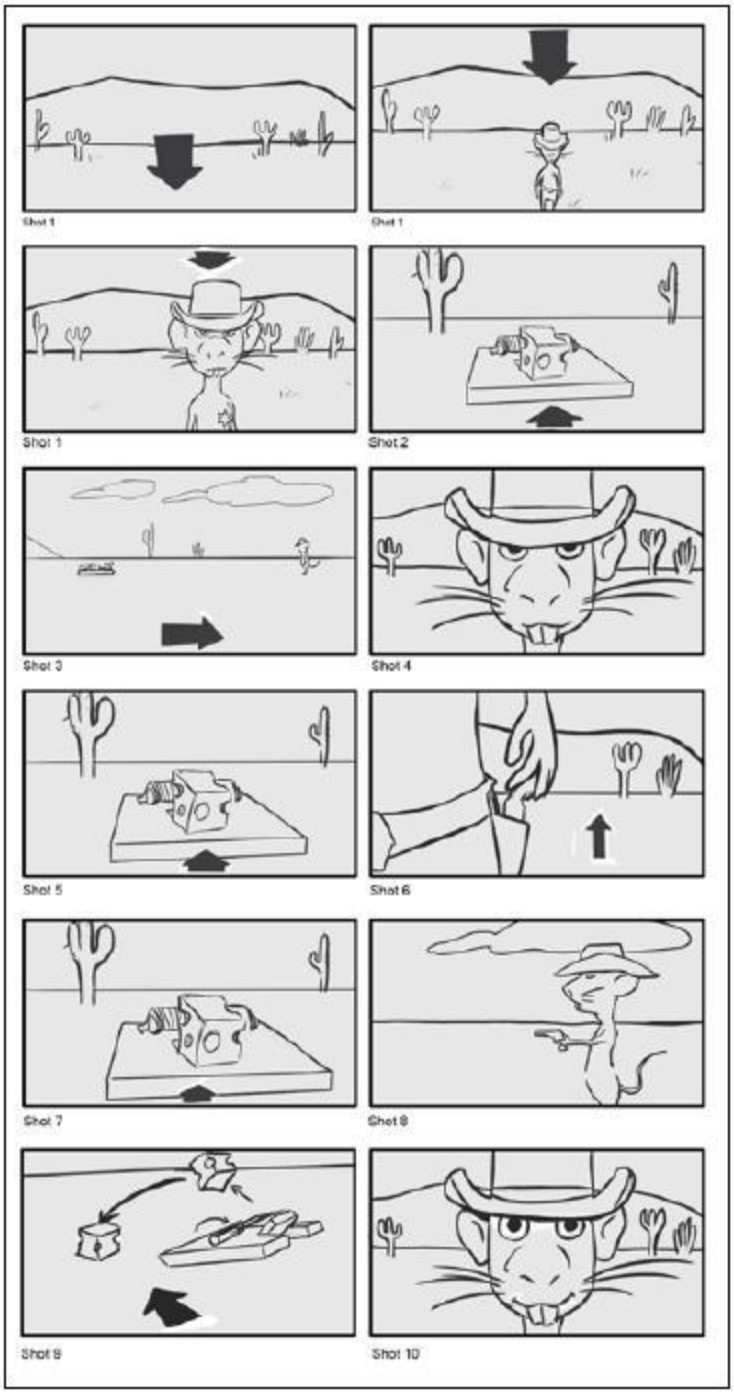
After this first reflection of writing the script, we can start making a storyboard. It is a technical document that the areas of animation films have been using since the 30's. A storyboard allows us to describe the action with drawings, but it also to goes further than the text in the design of the shots. In the case of teamwork, it is a very useful tool to communicate the work, and it gives a comprehensive view of a project.

Seeing that the storyboard allows us to save a huge amount of time and money, it is a practice that has gradually extended to the field of cinema (classical movies, but mostly special effects movies), theater,

clips, and commercials. Even if we are very far from making a blockbuster, and have no team to communicate our work to, a storyboard is a very important step to make a good animated short film.

Don't worry if you are not very gifted in drawing. Many storyboards are very simple and schematic. The most important thing is to clarify your ideas of staging. It must be easy to understand with the indications of stage direction such as camera and character motion.

For our storyboard, we draw the different shots referring to the script that we have done previously. Continuity is from left to right like a comic. To describe a shot, we can make several drawings. For shot 1, three drawings are used to describe the movement of the camera and the character. In order to avoid getting lost, we mark the number of the matching shot at the bottom left of each thumbnail.


Shot 1
Shot 1
Shot 1
Shot 2
Shot 3
Shot 4
Shot 5
Shot 6
Shot 7
Shot 8
Shot 9
Shot 10

Storyboard

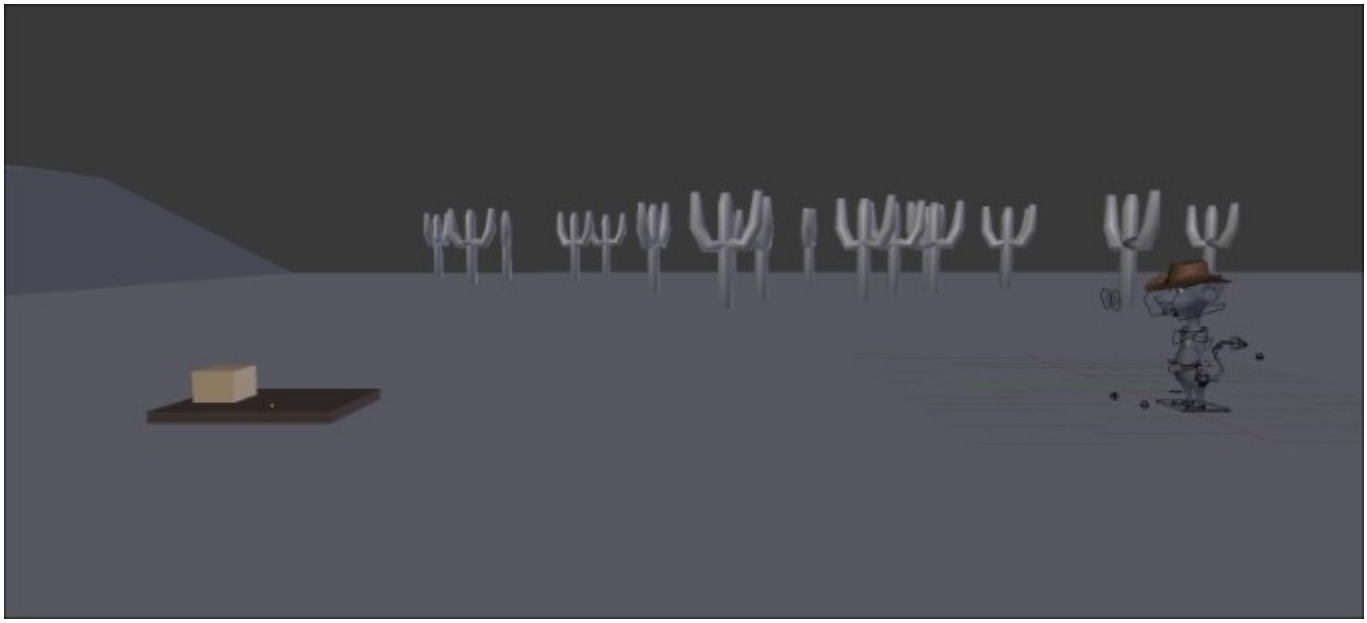
Finding the final camera placements and the timing through a layout

The layout is an animated version of the storyboard. It is sometimes called an "Animatic". We don't need to animate our character at this stage. We simply need to visualize the shots that we imagined previously with the script and the storyboard. We can then verify if this works and get a better idea of the time we need for each shot.

The process is as follows:

1. We will import the character in Blender by simply copying it from the `RayCharacter.blend` file to a new file.
2. We will save the scene as `RatLayout.blend`.
3. We will start outlining very simple scenery with a few low poly 3D models. We will use a plane for the floor and an extruded cube for the mountains beyond.
4. We will model a simple cactus and duplicate it pretty much everywhere on the floor.
5. We will also make a simple model of the trap and the cheese. We only need to get the basic shape.
6. To be more comfortable with this, we will organize the environment by dividing it in several parts. It is much better to display the **Dope Sheet**, **Graph** Editor, and camera view in a little window (0 of the numeric keyboard).
7. For each shot, we will create a new scene by clicking on the + button in the main menu bar, and we will select the **Full Copy** option.
8. The placement of our character, the trap with the cheese and making a few tests with the camera are still remaining.

This is the step where we can still make a few changes and test movements and timings.



Screenshot of the layout shot 03 with the rough modeling

Animation references

While animating, it is important to have as many references as you can so that you can have the perfect shot. Many computer animators have a folder with videos of themselves acting the shot. Recording yourself is one of the best ways to understand the gesture of a character. This way you will be able to catch many unconscious movements that you do when you look at the video. It is also a way to improvise different acts. Other reference materials such as character poses or animation cycle images are very interesting. Apart from this, paper and pencil are often useful in order to grasp some poses that you have in mind. You don't need to draw in detail, as a simple stick figure will suffice in order to put the poses ideas on paper.

Organization

Before starting to animate our shots, we will introduce to you how to organize yourself for the whole sequence. The different assets have been created in different `.blend` files. What's neat about this is that we are going to link them all in one final file for each shot. The benefit of this is that if we want to change the look of one of the assets, we can do it in the original file and it will replicate in the master file. For the rig of the rat cowboy, we are going to create a proxy.

In our case, we have ten shots, so we will create one `.blend` file for each of them. All these files will be placed in a `Scene` folder and will reference files that are placed one folder up in the folder hierarchy. Let's create our files:

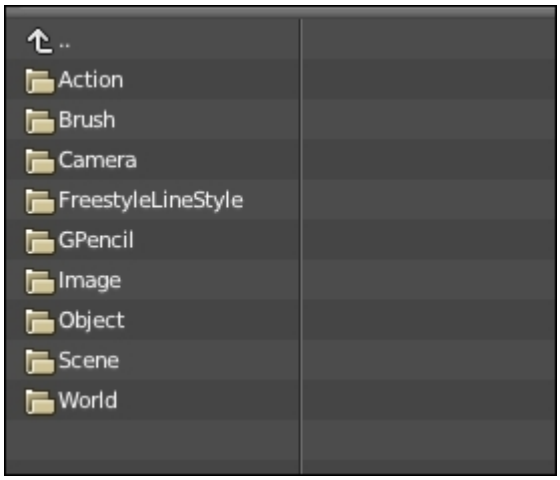
1. We will first create a new blank file in Blender and save it as `01.blend` in a new folder named `Scene`. Note that you can create a new folder in the file browser with the *I* key.
2. Now let's open the terrain file and select everything that needs to be linked in the shot file. We are only selecting the mesh type objects here. We are going to group our selection with *Ctrl* + *G* and rename the group as `Terrain` in the last tool option subpanel.
3. We will then repeat the same process with the other asset files. For the cactus file, you can create one group for each cactus.
4. Now the different groups are ready to be linked in the shot file. In the `01.blend` file, select the **Link** option in the **File** menu or press *Ctrl* + *Alt* + *O*. We can now click on the **Terrain** file and navigate to the **Group** folder in order to select the **Terrain** group that we've created within this file. We can validate by pressing **Link from Library**.

   ## Note

   The Link and Append file structures

   The structure of a `.blend` file is composed of different sections that represent the file. Each section is related to the entity it contains. For instance, in the **Group** section, you will find every group that has been created in the file, and in the **Nodes** section, you will every node that has been created in the file, and so on. This file format is quite nice because it's open and very well organized.

One very cool feature of Blender is the ability to mix files or parts of files together by linking or appending them. With the **Link** option, you keep a relation with the original file, so any modification will be replicated. The **Append** method creates a pure copy of what you want to mix.



The structure of a blend file

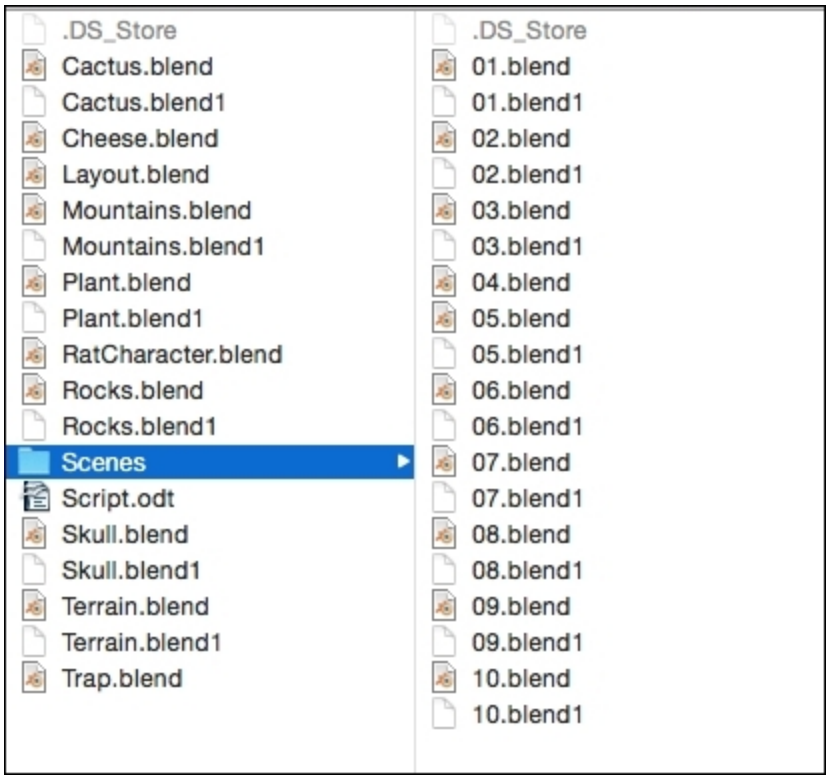
5. The link should be done. You can test whether it works by saving the **01** file then tweaking the **Terrain** file, and going back to the **01** file to see whether the changes appear. Now we can repeat the same process with the other assets that need to be linked in the shot file. You can easily nest files by linking groups to files that are linked themselves as a group in another file.
6. Cactus, bones, and bush are linked in the terrain file and they are part of the **Terrain** group (remember to add them to this group, as it will be linked in each shot file). The terrain is linked in each shot file. The **Cheese** group is linked to the **Trap** file, and the **Trap** group is linked to the **02**, **03**, **05**, **07,** and **08** files. We will not link the trap in the terrain file as it won't be needed for each shot. For the rat character, we will simply create a group with the **Armature** and **Mesh** object that we will link to the **01**, **03**, **04**, **06**, **08,** and **10** files.
7. In each file, we need to create a proxy for the rig of the rat. To do so, we will select the rig, and we will press *Ctrl* + *Alt* + *P* and click on the **Armature** object.

Note

Proxy

You may have already seen that when you use the link option, you can't do any modifications in the linked file. This is a security guard, so you only manage your art in one file. But in the case of a rigged character, this could be embarrassing. That's why we create a local access of the rig called a **Proxy** in the linked file with *Ctrl* + *Alt* + *P*.

You can have a look at the structure of our project as follows:



The architecture of our project

Animating the scene

Now that we have a story to tell, let's start to animate each shot using the tools that we saw previously.

The walk cycle

We are now going to learn how to create a walk cycle for the first shot. Why a cycle? This is because we are simply going to repeat the walk actions automatically later in order to save time. There are different types of walk that can express the actual feeling of the character. In our case, we are going to animate a cowboy walk, so this means our character will need a certain assurance. In order to be efficient, we are first going to "key" the three main poses of a walk as follows:

1. We will first open the 01.blend file and focus our view on the left-hand side view of the character. We will need to be sure that the **Auto Key** button is turned on, so any translation (grab, rotate, or scale) will create or override a key where the bar is located in the timeline. This button is located in the header of the **Timeline** editor and looks like a recording button.
2. Before creating our poses, we will have to select our armature in pose mode. We can then hide the **Master** bone by selecting it and pressing *H* because our walk will be animated onsite. Also, be sure that your timeline bar is at frame 0.
3. The first pose that we are going to create is usually called the "Contact pose" because both feet are touching the ground. From the side view, we will select the left foot bone, and we will move it in front of the character. We will then place the other foot bone in the opposite location. As you can see, because the legs are too far apart, the character isn't able to touch the ground (represented here by the Y world axis). So we have to select all the visible bones and move them down until the character is on the ground.
4. Now we will lift the front leg toes up a little bit on the local *x* axis (press *R* and then *X* twice).
5. We will then pose the back foot so that the rat stands on its toes.
6. From a front view, we will orient both legs outwards a little bit.
7. The arms need to follow the direction of their opposite leg. To move the arm, we will use the **HandIK** bone. We can also slightly bend the arms and rotate it, and then break the hand rotation a little bit.
8. From a front view, we can slightly put the right arm inward. The left arm will not rotate as much inward because of the holster.
9. In order to polish this pose, we will rotate the **Hips** bone down in the direction of the right leg and the **Spine01** and the **Neck** bone in the opposite direction of the hips. We will also rotate the head down so that the character looks at the ground. We will also pose the fingers so that the index finger is straighter than the others. We will also rotate the tail outward.
10. At this point it's a good idea to open a **Dope Sheet** editor. As you can see, we have small diamonds that represent the keys on each bone (described in the list on the left-hand side). For the main poses, it's always a good idea to have a key placed on every bone. To do so, we will select them all in the 3D view, and we will press *I* and choose the **LocRotScale** option.
11. In order to have a perfect cycle motion, the first and the last key of the walk needs to be the same. So in the dope sheet, we will select all the keys of our first pose by right-clicking on the **Dope Sheet Summary** corresponding to the key, and we will press *Shift* + *D* to duplicate this. We can move this to frame 24. You can clearly see that the keys are the same because of the dark lines that link them.

12. Now, we will copy this pose right between these two keys but in a mirror. To do so, we will select every bone in the 3D viewport, and we will store the pose with *Ctrl + C*. At frame 12, we can press *Ctrl + Shift + V* in order to copy the pose in the mirror, thanks to our bone naming convention with the **.L** and **.R** suffixes. If we scrub the timeline bar, we can see a preview of our walk. But there are still many things missing.

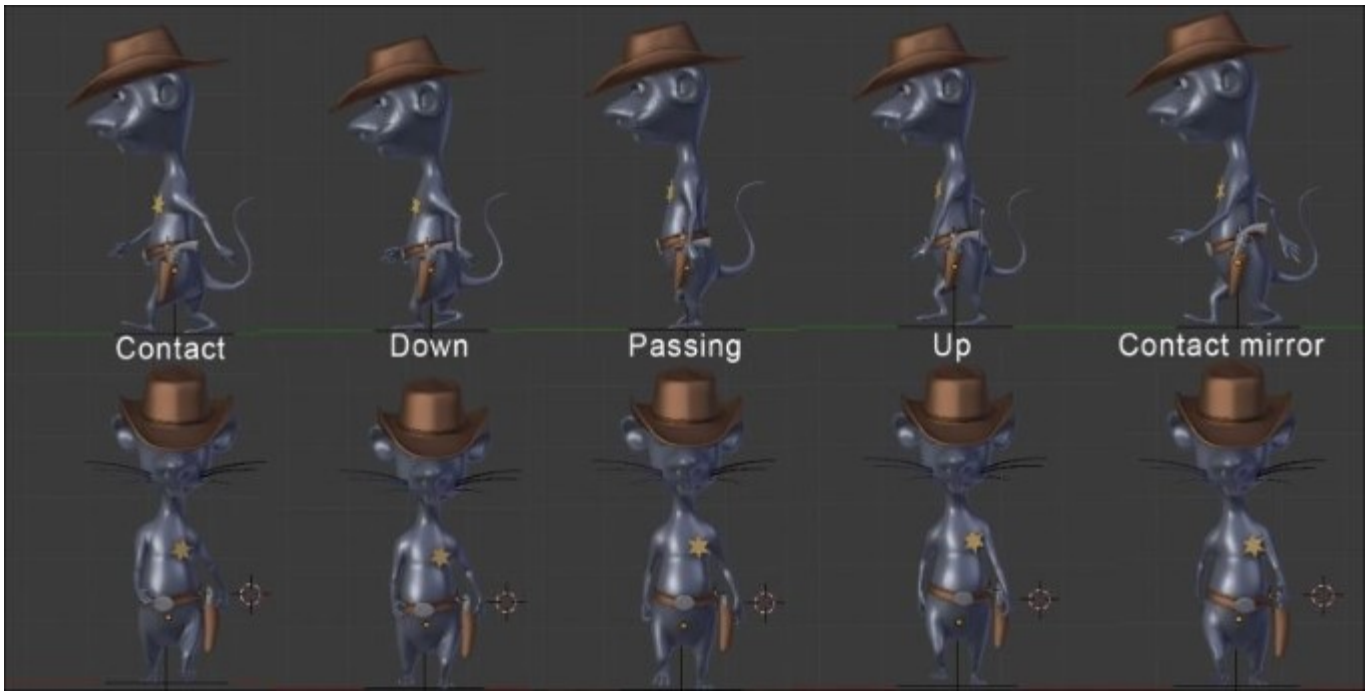
Congratulation, you have made the contact poses of the walk cycle! Now we will create the "Passing pose". This is done as follows:

1. To create our passing pose, we will use the same process as before. We will first create our pose between the 1 key and the 12 key, and we will mirror it between the 12 and 24 keys.
2. So we will place our timeline bar at frame 6. We will start creating our pose from the side view. The leg that was in the front of the character will be straight. So we will select the left foot controller, and we will align it horizontally with the rest of the body. As you can see, the leg can't be straight if the hips aren't moved up. We move the hips up. This is the key point of any walk: *the body always moves up and down*.
3. The other foot is bent. It is also placed slightly behind the straight leg, and the toes point downward.
4. At this point, the arms are almost straight and aligned with the rest of the body.
5. From the front view, **Hips**, **Spine01**, and **Neck** need to be aligned with the ground.
6. We can slightly rotate the head on the *z* axis to the left of the rat.
7. Now we ensure that we have keys on every bone. After this, we will copy our pose in the mirror on frame 18 with the same method seen previously.
8. A nice trick you can do is to change the visual look of your keys in the Dope Sheet by selecting them and by pressing *R*.

We now have the essential poses of a walk. The rest of our work will consist of exaggerating the motion by adding a "Down" and "Up" pose. This is done as follows:

1. The "Down" pose will be placed before the "Passing" pose. To create this, we will clear the toe rotation of the front foot.

   Then we will exaggerate the pose by slightly moving the pelvis down. The whole goal of this pose is to feel the weight of body on the ground.
2. The "Up" pose is placed after the "Passing" pose. This consists of lifting the character on his straight leg and toes.
3. After we have finished these poses, we can duplicate them in the mirror by following the same order according to the "Passing" pose.



Walk cycle poses

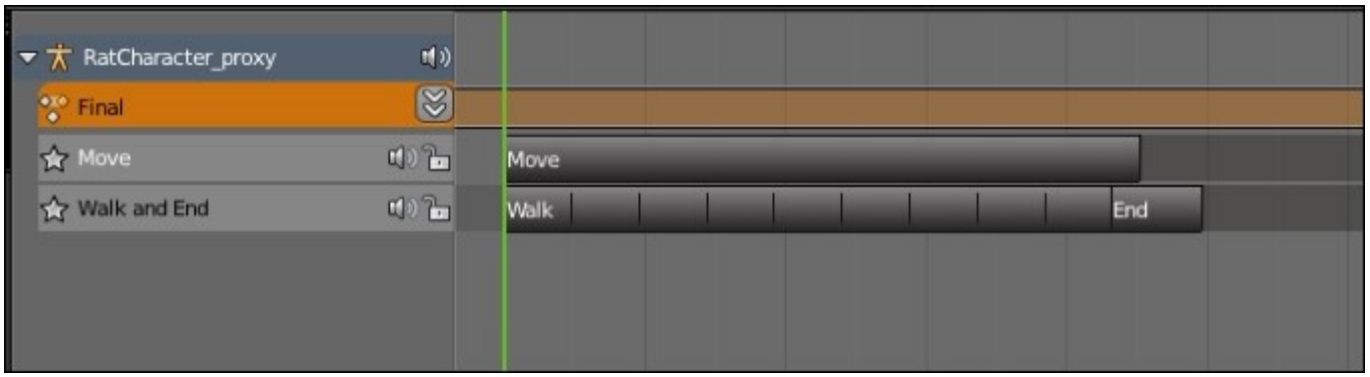
The walk cycles is completed now! We can view it by scrubbing on the timeline or simply by changing the end frame to the 23 frame in the timeline header and by playing the animation with *Alt* + *A*.

The Dope Sheet for our walk cycle

Mixing actions

We are now going to create three new actions that we will blend in with the NLA editor. One will be in charge of moving the character from its current location to the front of the camera. One will represent the character lifting his head up, and the last one will contain a mix of the others. This is done as follows:

1. The first thing to do is to copy the camera from the Layout blend file. To do so, we will simply copy the camera of the first scene in the corresponding file with *Ctrl + C*, and we will paste it with *Ctrl + V* in the **01.blend** file. If you have any other cameras in the scene, remove them and be sure that the copied one is the active one by selecting it and pressing *Ctrl + 0* numpad key.
2. In order to create our new actions, we will need to open the **Action Editor** from the Dope Sheet editor. As you can see, our walk cycle has been already placed on the default action. By the way, we can rename it `Walk`.
3. We can now click on the **+** button in order to create a new action. This new action will be a copy of the walk action, so we can delete all the present keys and rename the action `Move`.
4. Now we can start to animate the character displacement on this action. To do this, we will simply use the **Master** bone of our rig. In our case, we have added two keys at frame 0 and 225.
5. The problem now is that the rat seems to accelerate at the beginning and decelerate at the end. To solve this, we are going to use the graph editor and change the shape of the Y location curve. So in the graph editor, we select the keys for the Y location, and we press *V* and **Vector**. Now, as you can see, the curve is linear. Remember to save your file!
6. Now we are going to mix the walk and the displacement together. To do this, we will create a new action that contains all the others. We rename it as `Final` and remove all the present keys.
7. In the NLA editor, we will ensure that we are editing on this action. We will then create a new track (**Add** | **Add tracks**) and press *Shift + A* to add the `Move` action to it.
8. We can then press *Shift + A* in order to add the **Walk** action under the **Move** track. Now we just need to repeat our walk cycle with the **Repeat** option located in the right menu (*N*) under **Animation Clip** (be sure that the walk action is selected in your track). The important thing here is that the cycle stops just before the **Move** action because we are going to animate be hand the end of the walk and the head in a new action.
9. We now need to copy the pose of the character when the walk cycle stops, so we can start with this pose in a new action. We will use *Ctrl + C* with all the bones selected.
10. Now we are going to create the next action. To do this, we will click on the **+** button again, delete the keys, and rename the action `End`.
11. We can now paste our pose (*Ctrl + V*) on the same frame where we copied it in the NLA in the **End** action. This is because of the camera motion.
12. Now we need to complete a half walk and animate the character's head. In our case the animation starts at frame 216 and ends at frame 248. We ensure the character stands on his feet correctly, and we can animate the head pointing towards the camera. We can also add an eye blink in the middle. We will also rotate the head to the left of the camera.
13. After finishing the animation of the action, we will need to reposition our keys so that they start at frame 0. In the **Action Editor**, we will press *A* in order to select all the keys, and with *G*, we will drag them until the first frame of our animation is on frame 0.
14. Now we can go back to the **Final** action and open the NLA editor. We will then place our **End** when the walk cycle ends.



*The NLA with our three actions mixed together in the **Final** action.*

15. That's all! We have now blended our three actions with the NLA. Note that you can also rename your tracks on the left-hand side of the NLA by clicking on their default name.

*One frame of the **End** action*

Animation of a close shot

The close up of the face that we are going to do is directly inspired by Italian westerns of the 60's. A close up allows us to give a tension focusing on the eyes of our character. This is done as follows:

1. We will start by opening the **04.blend** scene.
2. We will need to place our character at the same place as in the **01.blend** scene, so we will also open the **01.blend** scene.

3. In the **Right Panel** (*N*) of scene **01**, we must copy the location information of the **Master** bone (Root) on the three axes, X, Y, and Z. In our case, **X: -1.19863**, **Y: 0,0**, and **Z: 12.29828,** and we will paste them on the location parameters of the **Master** bone in scene **04**. Our character is now in the right place.
4. We need to be sure that the **Auto Key** button is turned on, and we add the first key at frame 1 on the camera and all the bones of the character.

This animation will be in 50 frames.

1. We also need to place the camera in front of the face of the character.
2. We will move to frame 50 of the timeline, and we will slightly move the camera on the local *z* axis.
3. We can see a slow in and slow out effect of the camera, the keys are in Bezier interpolation mode. In order to change the interpolation mode, we must open the **Graph Editor**. We will select the keys of the camera and press V and select **Vector**. Now the camera is moving in a linear way.

Let's animate our character. He is watching the trap, so he doesn't move a lot. The animation is done as follows:

1. We begin by animating the head. We will move the cursor of the **Timeline** at frame 50, and we make a little rotation on the local *z* axis, a new key frame is added.
2. The Frown bone stays down to keep a serious look.
3. We can now animate the eyes. At frame 50, we will move down the **EyeTargetMaster** a little bit. He is still looking straight toward the camera.
4. We will now add a blink of the eyes to give more realism.
5. We will select the both **EyeTarget.L** and **EyeTarget.R**, and we will add a key at frames 16 and 23 (press *I* and select **Scale**). We will move then at frame 19, and we will scale the bone controllers to close his eyes.

This very short animation for shot 4 is finished. Shot 10 is almost the same but with a smile at the end. Let's start the animation of the gunshot!

Animation of the gunshot

The animation of shot 8 this time is a bit more complex. This will be done as follows:

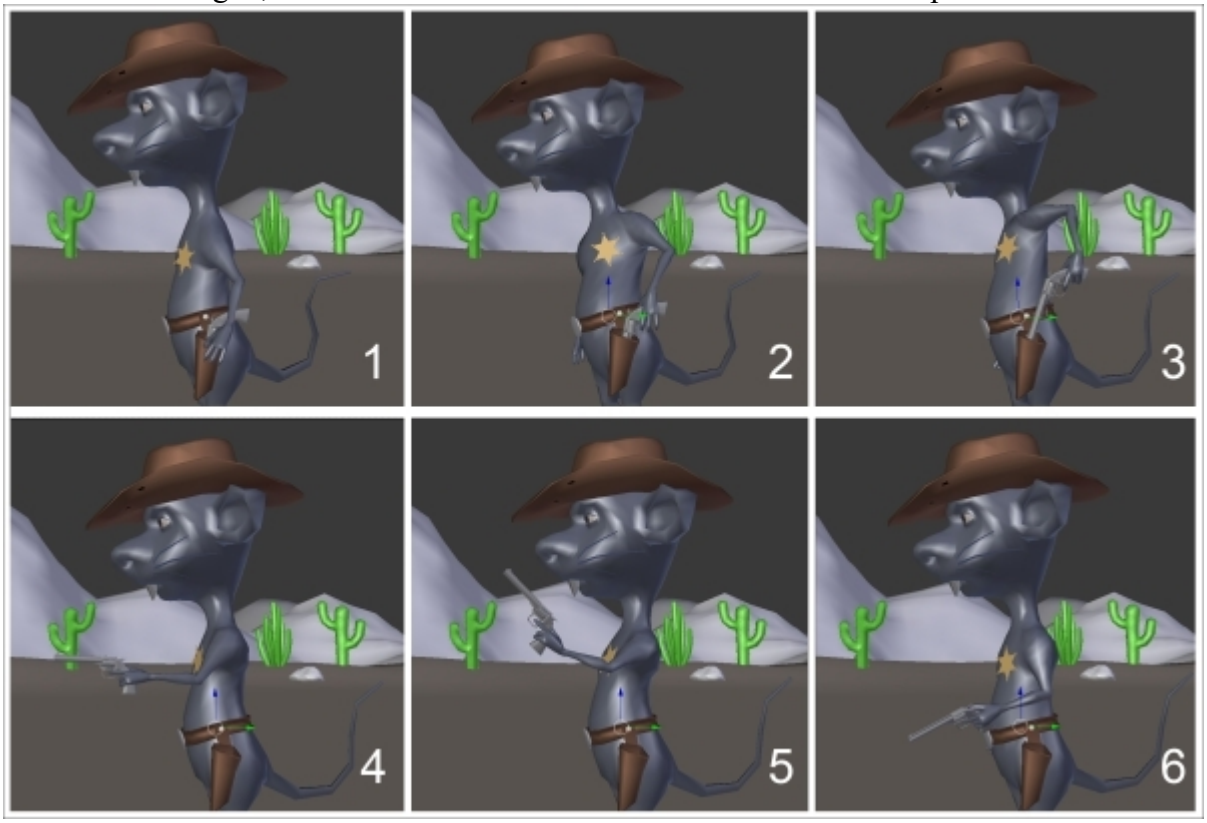
1. We will start by opening the **08.blend** scene.
2. We will place our character at the same location as the **01.blend** file.
3. With the **Auto Key** button turned on, we will start placing the camera on the left-hand side of the character.

It is an animation in 30 frames.

1. We will put the camera on the left-hand side of the character, and then we will move it slightly to the left along the *y* axis to frame 30. We will change the keys of the camera in the **Vector** mode (V) in the **Graph** Editor.
2. We can now animate our character. We will start animating the hand. The hand must go straight to the butt of the gun to hold it. The position and the inclination of the hand are very important. It doesn't matter if there is a little interpenetration of the thumb. The animation is fast, and the point of view of the camera can hide small mistakes. The forefinger must be close to the trigger when he holds the gun.

The gun shot

3. To move the fingers, we must select the top bones and rotate them on the local *x* axis (press *R* and then press *X* twice). The **Copy Rotation** constraints will make the rest.
4. When the gun is caught, we must change the influence of the **Child Of** constraint of the gun bone. The influence of child of controlled by the holster is now at **0.000**, and the influence of child of controlled by the left hand is now at **1.000**. The gun is now following the hand. We can keep animating the left hand.
5. We will position the hand so that the gun gives the impression that it will shoot ahead. There is the recoil of the gun, so the hand makes a sudden rotational movement upwards.



Animation of the gun shot

When the animation of the left hand and the gun is done, the largest part of the animation of the sequence will be done too, but we still need to polish this shot a little bit. The rest of the body must be animated and can't stay inert. This will be done as follows:

1. We need to make a rotation of the **Spine01** bone on the *z* and *y* local axis when the left hand catches the butt of the gun. Likewise, the left shoulder must rotate towards the top (refer to **2** and **3** in the preceding screenshot) at this moment.
2. As the rotation of the spine01 bone the head is inclined to the right, we will rotate it a little bit to adjust it. For the recoil of the gun, we will make a little rotation of the head.
3. The **HipsReverse** bone also makes a rotation on the local *z* and *y* axes to gives a more realistic feeling.
4. The right hand and the tail also make a little arc. In this case, they are quite important details.

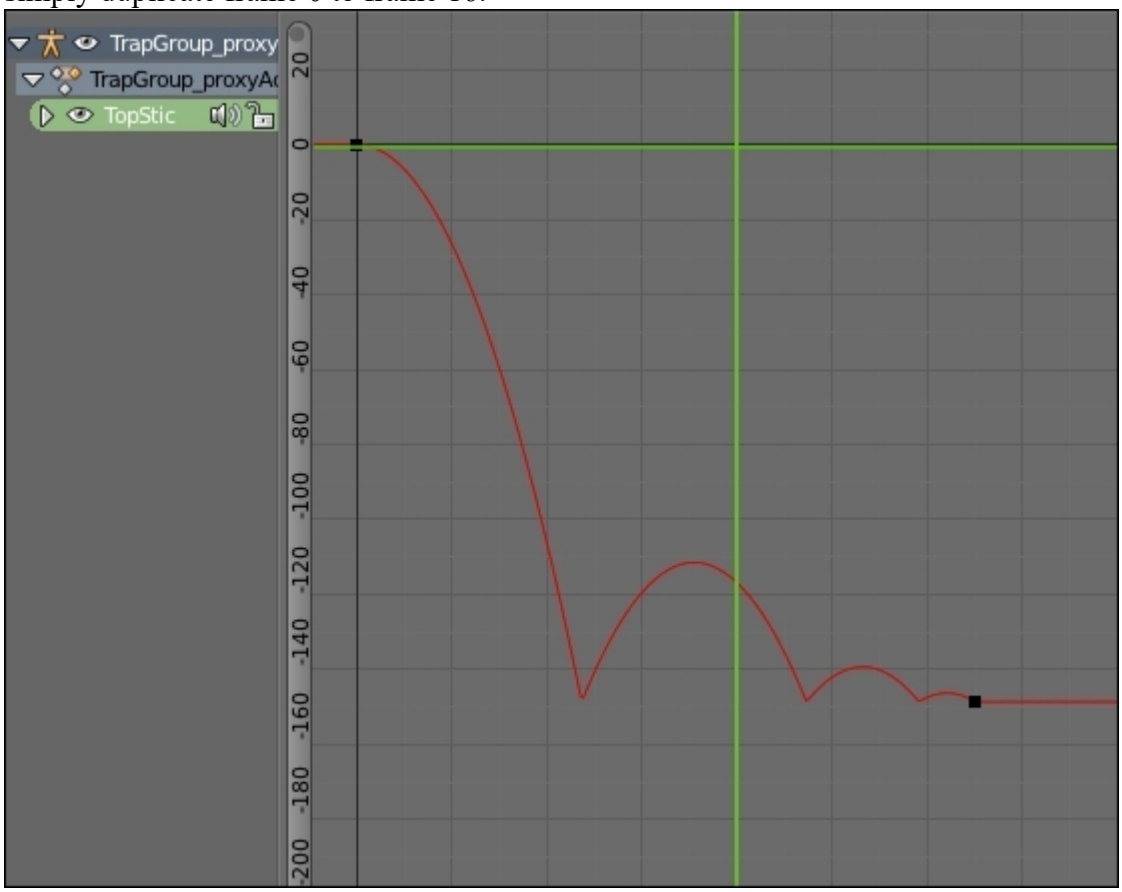
So, the animation of the gunshot is now complete. Let's talk about the animation of the trap.

Animation of the trap

It's now time to animate one of the most technical shots of our sequence where the cheese gets shot on the trap. This will be done as follows:

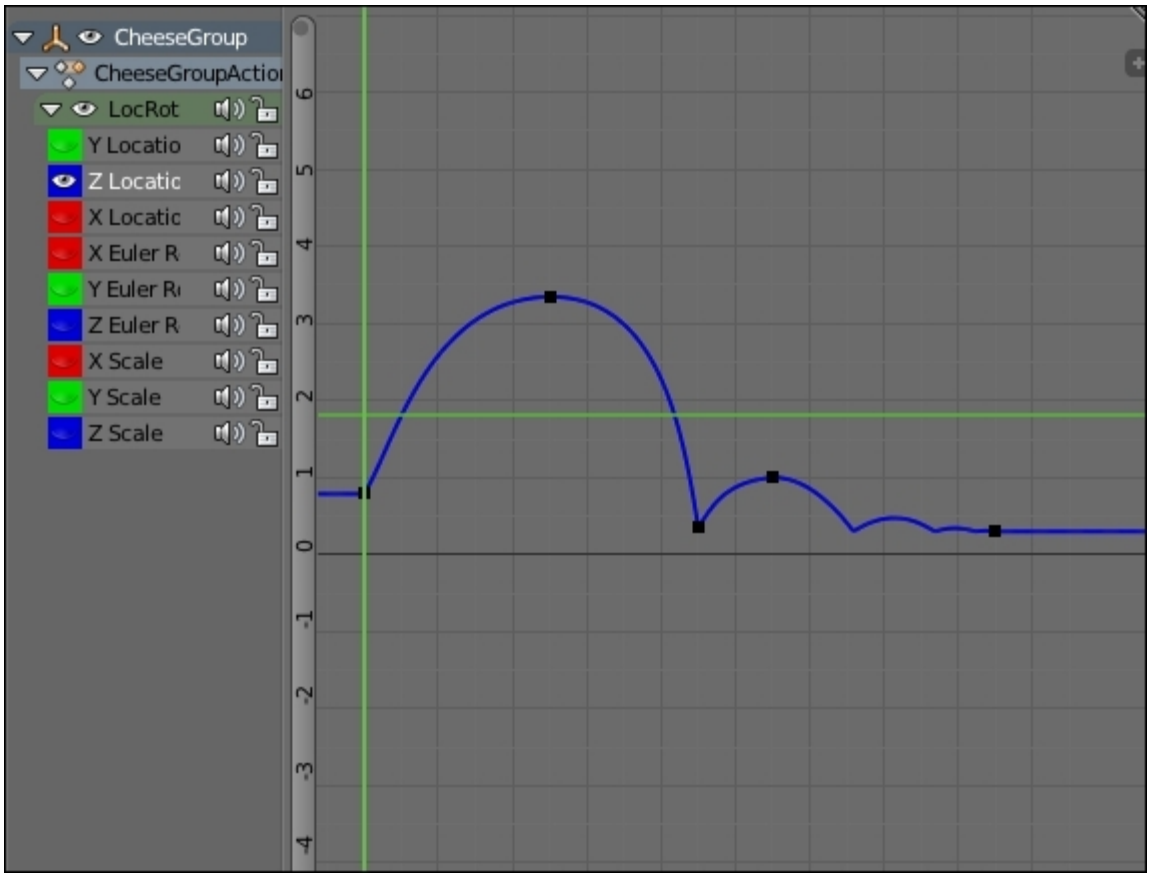
1. We will start by opening the **09.blend** file. Then we can frame the trap with camera as done in the layout.
2. The first thing to do is to add a key frame on each bone of the trap only for rotation, so we select them and press *I* and select **Rotation**. We won't use AutoKey for now. Remember that the trap is linked. If you can't access the rig, it's simply because you don't have a proxy in order to manipulate it.
3. Now we can go to frame 13 and rotate the **TopStick** and the **TrapPlank** bone on the their *x* local axes as far as their can logically go.
4. The **Spring** bone animation will be shorter, though. We will key its extreme rotation on frame 7.
5. We can now test our animation to see what's wrong with it, but first we will hide the cheese with the *H* key. As you can see, the animation doesn't seem very natural. This is why we are going to tweak the curves in the graph editor.
6. We will open the graph editor and click on the **Show only selected** button in the header.
7. Now we can select the **TopStick** bone and unfold its **Rotation** values on the left-hand side of the **Graph** editor. We can even hide the other *y* and *z* rotations as we don't need them. To do so, we will use the Eye icon. As you can see, the curve has an ease in and ease out effect. We don't want this. We want a bounce effect. To do so, we can, of course, add more keys and waste time to animate it by hand, or we can use the power of the Dynamic effects of the interpolation menu. So we select our keys, and we press *T* and select **Bounce**. As you can see, the curve changes.
8. We can do the same with the **TrapPlank** and the **Spring** bones. For this, a later bounce will be more subtle. Now if we play our animation, everything looks more natural.
9. Now let's animate the cheese. First unhide it with *Alt* + *H*.
10. We can now reactivate the **AutoKey** option, it will be easier. Let's start by adding a key on frame 0 with press *I* and select **LocRotScale**. Now, because of autokey, we can move to frame 16 and move the cheese to its final destination.
11. We can now go to frame 5 and put the cheese in the air. This will change its Z location. We can also rotate it and scale it locally on its *z* axis (press *S* and then press *Z* twice).
12. The animation at this point is not very convincing. In order to add more realism to it, we can open the graph editor and change the curve of the Z location of the cheese, so it is much more like a "dome" at the beginning. To do so, we can set frames 0 and 9 to a vector type (press *V* and select **Vector**). We will need to change the handles of frame 5 too.
13. Next, we will change the end of the motion by adding two new keyframes on the Z location at frames 11 and 17. These keys will have a **Bounce** Dynamic effect. We will use the handles in order to smooth the curve as much as possible where it is needed. Remember that the animation process involves a lot of trial and error. You also need to constantly play your animation in order to know what you'll have to correct.
14. We can now reset the scale of the cheese at frame 11 by pressing *Alt* + *S*.
15. What we've done so far is the base of the cheese animation. Now we can polish the animation by changing its rotation here and there and by polishing the Y location curve.

16. We can now animate the camera. Between keys 0 and 16, the camera doesn't move, so we simply duplicate frame 0 to frame 16.



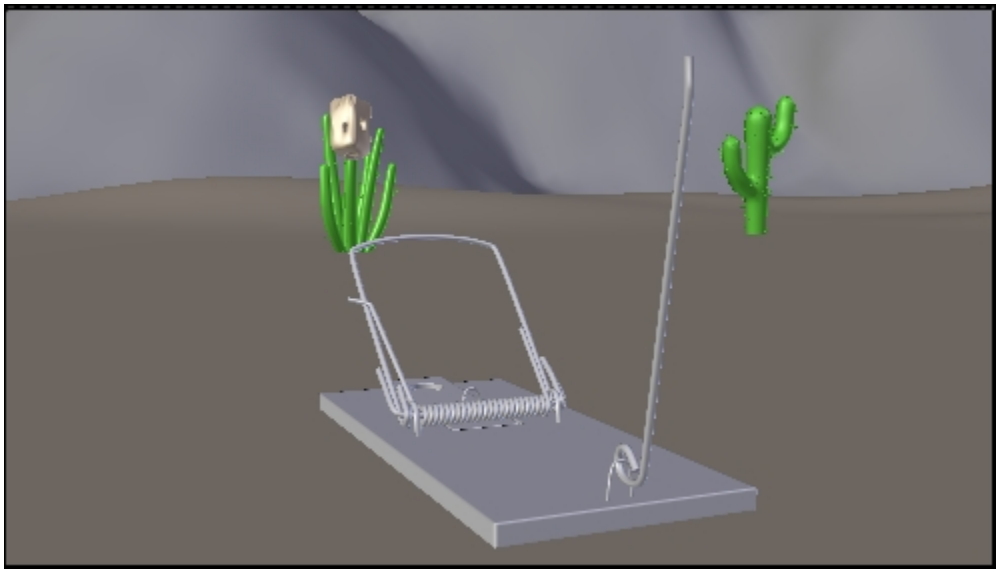
*The **TopStick** X rotation curve*

17. We can now place our timeline bar at frame 37 and focus on the cheese closely, so we can see the hole due to the gunshot. The key has been placed due to the AutoKey option.



The cheese Z location curve

The animation of the 09 shot is now completed, congratulations!

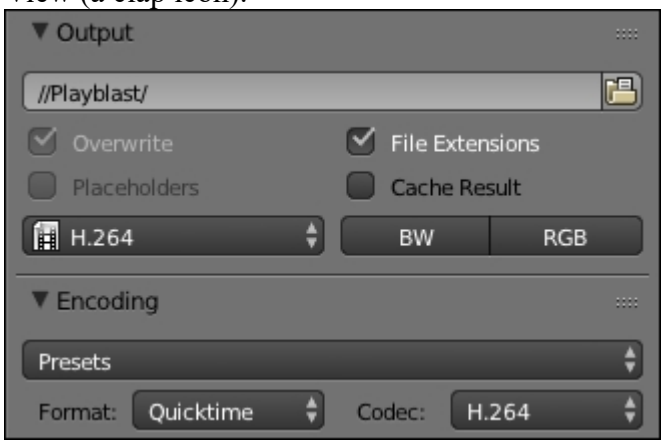


Frame 4 of the 09 shot.

Render a quick preview of a shot

The last thing we are going to do is render quick previews of each shot. This is often called a **PlayBlast**. This simply renders the animation of the viewport. In this section, we are only going to render the playblast of the first shot. So let's start:

1. We will open the **01.blend** file.
2. In the Properties editor, in the **Render** tab, we will go to the **Output** subpanel. Here we can choose the path where the render will be. To be well organized, let's create a **Playblast** folder in our **Scene** folder and set it as the output path.
3. Now we will need to choose the file format. By default, it will output a sequence of a PNG file. We change it to **H.264** in order to have a movie type file.
4. In the **Encoding** section, we can choose the extension of our movie, and we choose **Quicktime** to have a `.mov` file. Later, you'll see how to render the shots frame by frame in a non-video format, but for now in order to quickly see the result, we will use .mov.
5. Be sure that you are viewing from the camera by pressing the *0* numpad key. In order to render the playblast, we can press the **OpenGL Render active viewport** button in the header of the 3D View (a clap icon).



*The **Output** option for the **Playblast** rendering*

Summary

This was a very tough chapter, but we hope you have found it interesting. We covered a lot of things here, such as the principles of animation, the majority of the Blender animation tools, and how to prepare ourselves with a script, storyboard, and layout. We also learned how to animate a walk cycle and mix it with other actions using the NLA. We applied many of the 12 principles of animation such as pose-to-pose, squash, and stretch overlapping. Now, you should have all the knowledge to start telling your own stories with Blender. In the next chapter, we are going to finalize our sequence by rendering it with cycles and editing it with the VSE.

Chapter 10. Rat Cowboy – Rendering, Compositing, and Editing

Welcome to the last chapter of the module. In this chapter, you will learn advance material creation in Cycles, such as how to use a skin shader or how to create a realistic fur. Next, you will learn about passes and how to do a raw render with the different passes. Then, you will receive an introduction to the nodal compositing so that you can enhance your shots. Lastly, we will talk about the Video Sequence Editor in order to edit the final sequence. Let's start!

In this chapter, we will cover the following topics:

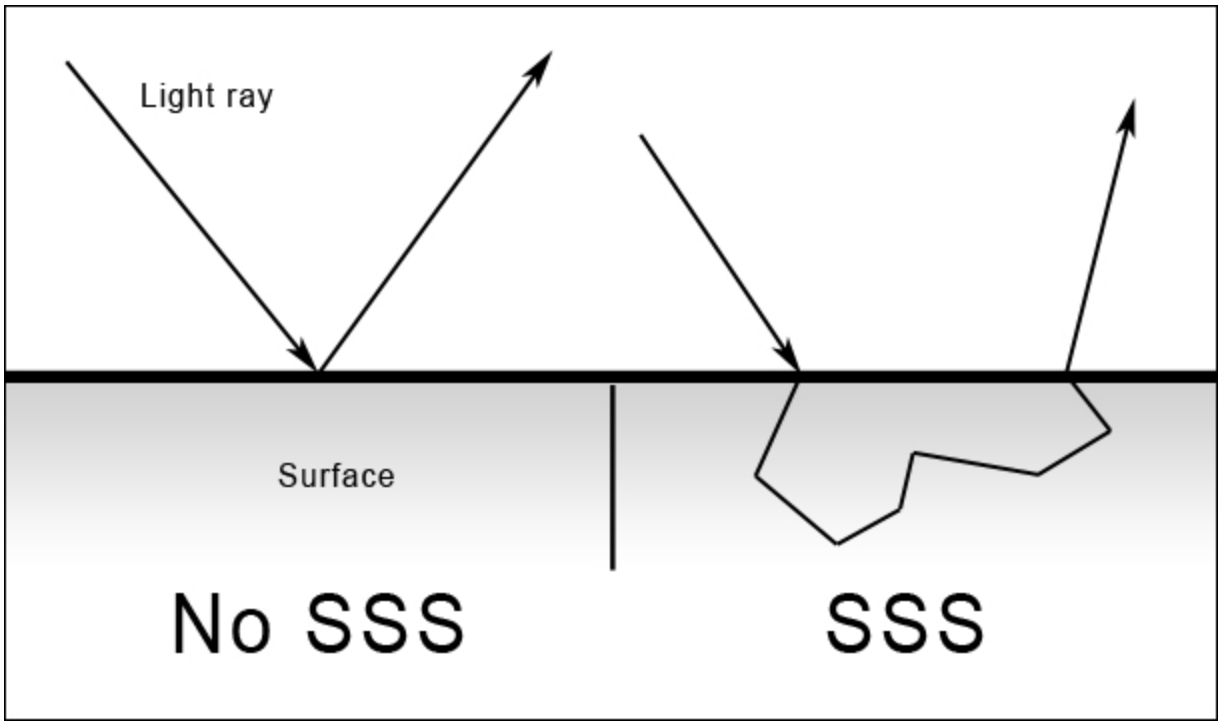
- Creating advance material
- Creating fur particle systems
- Setting up Cycles for an animated scene
- Using passes
- Introducing nodal compositing
- Editing a sequence

Creating advanced materials in Cycles

We already covered material creation with Cycles in the Haunted House project, but now we are going to go further by creating a skin material using subsurface scattering, a complete fur, and an eye material. Let's start!

Skin material with Subsurface Scattering

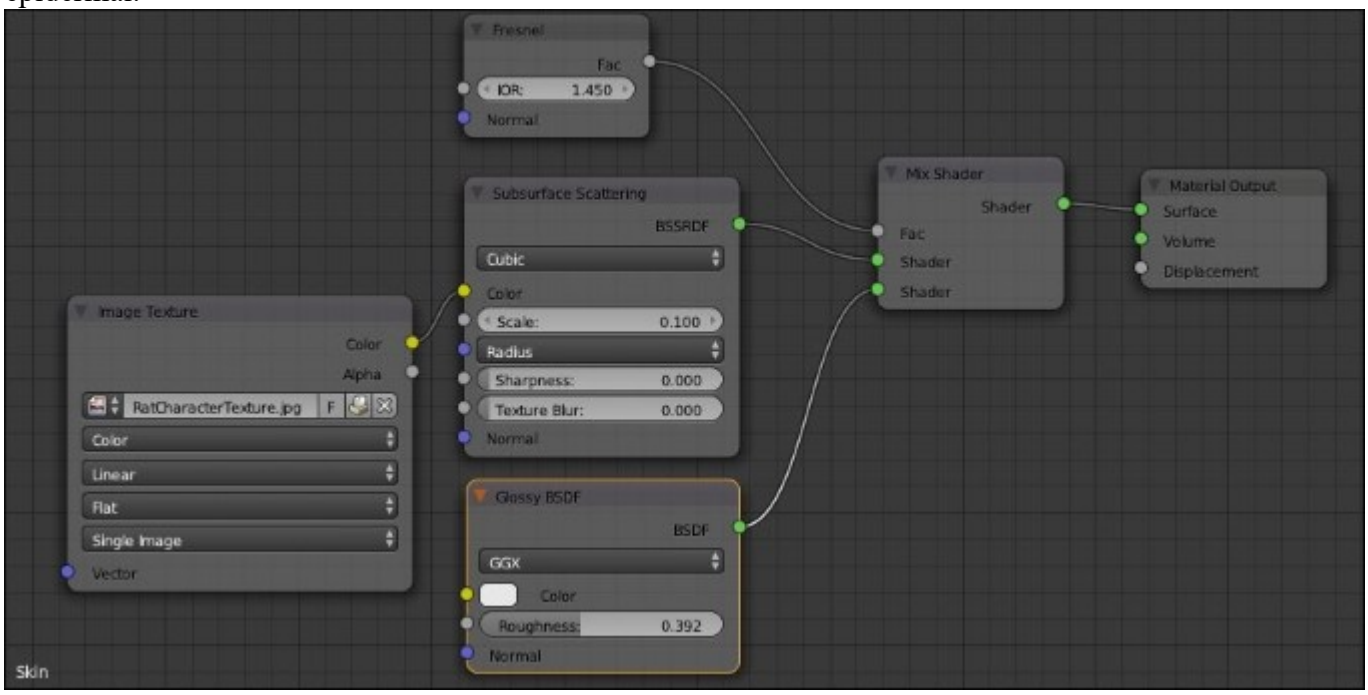
The skin has a very translucent aspect. We can truly see this effect when we pass our hand in front of a lamp or in the thin part of the ear (the helix). So, when creating a skin material, we get this phenomena with a Subsurface Scattering node (usually abbreviated SSS). It is called this because the light rays are scattered through the geometry when intersecting the mesh. This is not the case with a diffuse shader, for instance, as the light rays are simply blocked. SSS often gives a reddish tint to the thin parts where light rays scatter a lot. So let's create the skin material of the rat.



The way light rays react on SSS surfaces

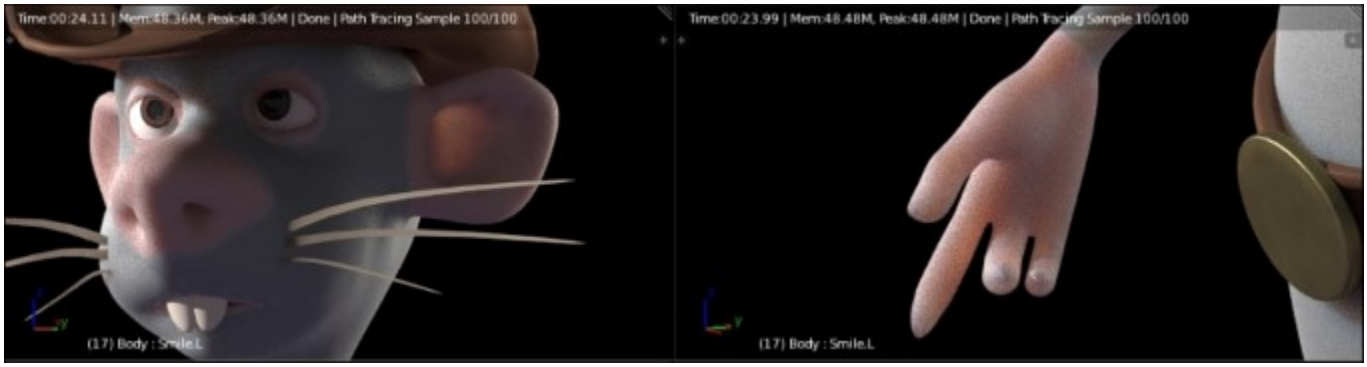
1. We will open the `RatCharacter.blend` file and split our interface so that we have a second 3D view for the real-time renderer and a node editor. Note that the real-time renderer for the SSS shader will only work in the CPU mode or with the GPU in the experimental mode.
2. We will add a new slot in the material tab of the Properties editor and a new material that we name as **Skin**. We will press the **Use node** button in order to work in the node editor.
3. In the node editor, we will remove the default **Diffuse** shader by selecting it and pressing **X**. Then, we can add a **Subsurface Scattering** shader by pressing *Shift* + *A*.
4. We have some options to tweak for this shader. The first one is the **Scale option,** which corresponds to the amount of SSS that we want. In the case of the rat, we set it to **0.1,** but in order to have the correct value, you will need to perform a test. It's just a matter of placing a light in the back of the character and looking at the thin parts, such as the ears.
5. The next very important setting to tweak is the radius that corresponds to the predominant color that will result to the SSS effect. In many cases, we will let more red than green and blue because of the color of the blood that's under the skin. This is a set of three values that corresponds to R, G, and B, in our case, we set them to **1.0**, **0.7,** and **0.5** respectively.
6. Now, we will plug our texture in the **Color** input. Finally, we can connect the shader to the output.
7. We will now add a reflection to our skin by mixing our SSS shader with a glossy shader. To do so, we append a **Mix Shader** on top of the SSS to the Material output wire.
8. In the second shader input, we can bring a **Glossy BSDF** shader and change the **Roughness** value to **0.392** so that the reflection is less sharp.
9. For the **Fac** input of the **Mix Shader**, we will add a **Fresnel** node. This skin shader will be sufficient for our needs, but note that we can go much deeper in the subject by creating a shader

with multiple maps that corresponds to the different skin parts, such as sub-dermal and epidermal.



The skin material nodes

The SSS effect on the ears to the right and on the hand to the left in viewport rendered mode will look like the following:



Eye material

We are now going to create a less difficult material, that is, the one of the eye corneas. This will be like a glass material, but we will optimize it a little bit because the default glass shader is so physically accurate that it also casts shadows of the glass itself. These take a long time to render and are rarely

visible. In order to apply our material, we will model a simple cornea in the eye mesh itself. The front part of the cornea is extruded a little bit. This allows us to catch the reflections rays better:

1. We will first need to add another material slot and a material that we rename as **Cornea**. As you can see, we already have three slots that correspond to the white part and the pupil of the eyes. Feel free to replace it with one material with a texture.
2. Next, we can select the cornea piece of the mesh in the **Edit Mode** with the **L** key and press the **Apply** button in order to apply the cornea material on this part of the mesh.
3. In the node editor, we will replace the default Diffuse shader with a **Glass BSDF** shader.
4. We will now mix the **Glass BSDF** shader with a **Transparent BSDF** shader. A transparent shader simply lets every ray to pass through.
5. Now, in the **Fac** input of the **Mix** Shader, we will plug a **Light Path** node (press *Shift* + *A* and select **Input**) with the **Is Shadow Ray** output. This will tell the render engine to use the transparent shader for the incoming shadow rays and the glass shader for the others. At this point, we should have a nice reflecting eye.

The fur of the rat

Now, let's dive into a complex section about fur creation. In order to create a convincing fur, we will have to create a complex material, have a perfect hair particle combing, and correct lighting. If one of these three parameters is sloppy, it won't look great. Let's start with the particle systems:

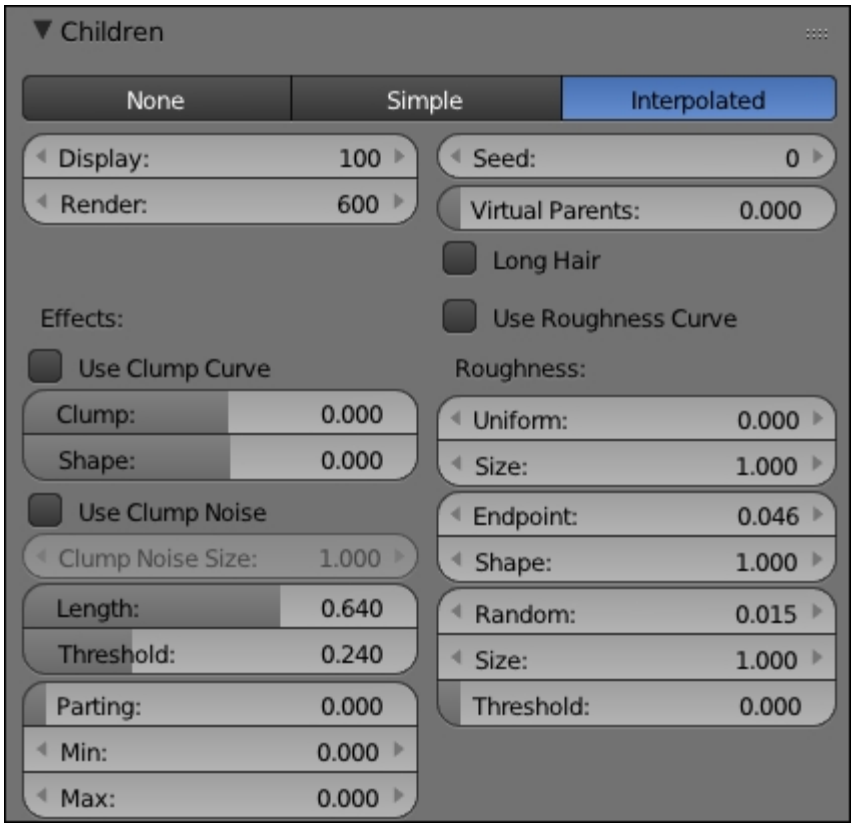
1. In order to add more realism, we are going to create three particle systems. Let's first select the character in the **Edit Mode** and add the main particle system in the Particle tab of the Properties editor. We will name both the system and its settings **Basic_FUR**.
2. We will change the system's type from **Emitter** to **Hair**. In the **Emission** subpanel, we will change the **Number** to **500**, which correspond to the guiding hairs. We can also change the hair length to **0.140**.
3. In the **Children** section, we will choose the **Interpolated** method. The number of children that will follow the guiding hairs is too low. We will change the **Render** option to **600**, so each guide will have 600 children. We can also change the display option to **100** to preview the result in the viewport.
4. In the **Children** subpanel, we can change the **Length** setting to **0.640** and the **Threshold** to **0.240**. This will add some randomness to the length of the children.
5. Then, in the **Roughness** section, we can change the **Endpoint** value to **0.046** and the **Random** to **0.015**. **Endpoint** will spread the tips of each hair strand.
6. Now, we will create a Vertex group that will determine where the fur will be located on the rat. In the **Object data** tab of the Properties editor, in the **Vertex Groups** section, we will start by locking all our skinning groups by selecting the black arrow button and choosing the **Lock All** option. This will ensure that we don't change them inadvertently. We can now add a new group with the + button and name it **Fur**. In the **Edit Mode**, we can select the hands, nose, ears, tail, mouth, and eye contour. We invert the selection with *Ctrl* + *I* and press the **Assign** button with a weight of **1.0**.
7. Back in **Particle** tab, in the **Vertex Groups** section, we can set the **Density** field to our new vertex group. We should now only have hairs on the needed parts.
8. In order to improve our system, we will create a new vertex group named **Fur_Length** that will affect the length of the hair on certain parts. To create the group, we can duplicate the previous **Fur** group with the corresponding option in the black arrow drop-down menu. We can then use

the weight paint tools in order to subtract weight from different parts. In our case, the head is green and the arms and the legs are orange and blue under the belt.

9. In the **Vertex Groups** section of the **Particle Settings** tab, we can change the **Length** field to this new group.

Now, we need to comb our particles as follows:

1. To do so, we can use the **Particle Mode** (located in the same drop-down menu of the **Object Mode** or **Edit Mode**). By pressing *T*, we open the **Brush** panel where we can use the **Comb** brush to comb the character's hair.
2. We will use the same shortcuts as the sculpt mode for the brush settings. The **Add** brush is nice in order to add new strands when you find some gaps. When using this brush, it's best to check the **Interpolate** option so that it smoothly blends with the others.
3. In the header of the 3D view, you have three new buttons (to the right of the layers) that allow you to select the particle in different ways. With the **Path** mode (the first one), you can only control them with brushes, while with the **Point** mode (the middle one), you have access to each point of the hair, and with the **Tip** mode (the last one), you only control the tip. With the last two modes, you can grab and rotate your character's hair with **G** and **R**. In the left panel, we can also activate the **Children** option in order to see the children.
4. Now let's add another particle system and name it **Random_FUR**. We can then copy the settings of the first one by choosing it in the **Settings** field and pressing the 2 button to make a unique copy of it. Now, we can safely change the setting without affecting the other system. We can click on the **Advanced** option.
5. We will start by changing the amount of guiding hairs to 50 and their length to **0.1**.
6. In the **Emit from** section, we will choose **Verts** and check the **Random** option.
7. In the **Physics** subpanel, we can change the **Brownian** value to **0.090** to add a little bit of randomness.
8. In the children section, we will change the **Render** and **Display** sliders to **50**.
9. We will then change the **Length** to **0.288** and the **Threshold** to **0.28**.
10. We will then ensure that the **Density** field still contains the **Fur** group, but we will remove the **Length** field.



The children settings of the Basic_Fur system.

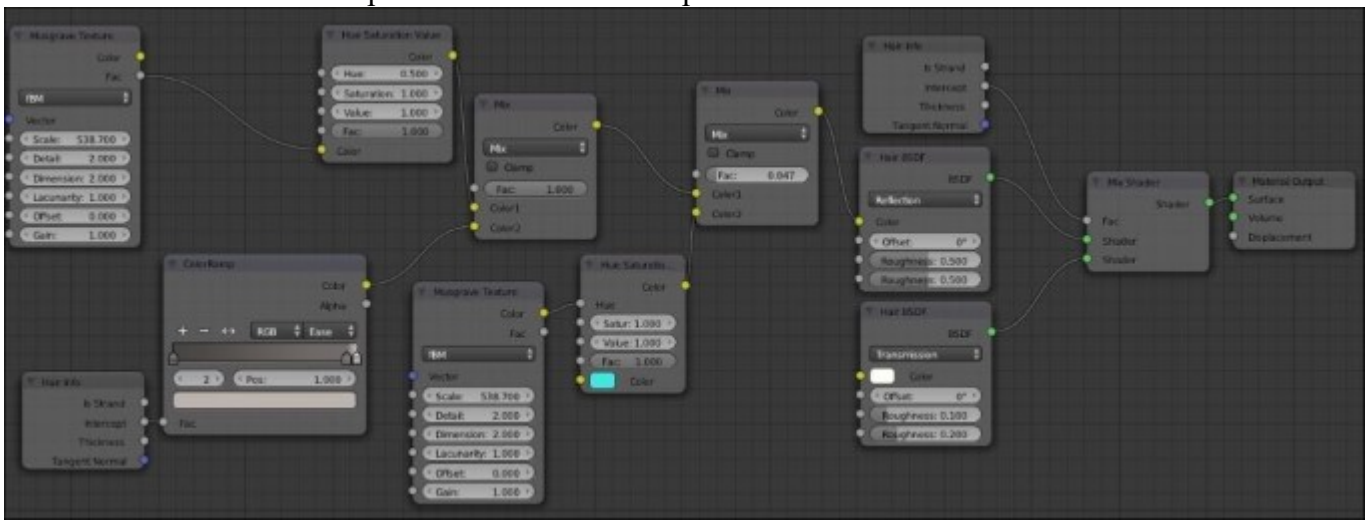
11. Just as we did for the first two systems, we will add a new system for the fur of the ears. This is very subtle. It has a short hair length and a vertex group for the **Density** field. It also has only 20 children.

The Particle Edit Mode

You are now done with the particle systems. It's now time to create the materials and change the strand thickness and shape. This will be done as follows:

1. Let's add a new slot in the material tab and a new material named **Fur**.
2. In the node editor, we will delete the default **Diffuse** shader and add two **Hair BSDF** shaders. The first one will be of the **Reflection** type with **RoughnessU** and **RoughnessV** set to **0.500**, and the second will be of the **Transmission** type with **RoughnessU** to **0.1** and **RoughnessV** to **0.2**. We will mix them with **Mix Shader**. The **Fac** input will be plugged with the **intercept** output of a **Hair info** node.

3. We will then add **Musgrave Texture** node with a scale of **538**. We will add a **HueSaturationValue** node with the **Fac** output of the texture connected to the **color** input. We will then mix the result with a **MixRGB** node and **ColorRamp**. The **Fac** input of the color ramp is fed with the **intercept** output of a Hair info node. This will add a gradient map on each strand.
4. We will then plug the output of the **MixRGB** node into another **MixRGB** node. The second input of this node will be fed with a **HueSaturationValue** node. The **Hue** input will receive the color output of a **Musgrave** texture node. This will add color variety to the strand. But in order to lessen the coloring effect, we will change the **Fac** of the **MixRGB** node to **0.047**.
5. Finally, we can plug the result of the last **MixRGB** node in the **color** input of the first **Hair BSDF** node. Our hair material is now completed!
6. We now have to change the **Render** settings of each particle system. In the **Render** subpanel of the particle settings of each system, we will choose our fur material and activate the **B-Spline** option with a value of **4**. This will smoothen the render of the hairs.
7. Then, for the **Basic_Fur** system in the **Cycles Hair Settings** subpanel, we will change the root to **0.5** with a scaling of **0.01** and a shape of **-0.09**. We will use the same settings for the **Random_Fur,** except for the root that we set to **0.15**. Again, we will use the same settings for the **Ear_Fur** system, except for the root that we set to **0.05**.
8. We can now add temporary lights in our `RatCharacter.blend` file in order to do test renders. We will set our samples to 300 and render a preview of our rat.



The fur material in the Node editor

The image with a preview render with low render settings is a follows:

Now you have the knowledge to create even more complex materials with Cycles! We are now going to show you how to render the first shot of the sequence, and what's nice with the link is that we will see our fur and materials in each shot file.

The Raw rendering phase

Previously, we have seen how to render an image in Cycles. It is quite different for an animation. It's best to first do a raw render of the shots with the following settings:

1. We will start by adjusting the device. If you have a good graphics card, remember to check the **GPU** device option.
2. Let's now adjust the samples in the **Render** panel (**Properties** | **Render** | **Sampling**). The skin needs enough samples to reduce a noisy effect. 100 or 150 samples are enough to have an idea, but consider setting a higher value for the final render.
3. Still in the **Sampling** tab, we will put **1.00** to **Clamp Direct** and **1.00** to **Clamp Indirect**. It allows us to reduce the noise, but you may lose a little bit of the bright colors.
4. We should remember to check the **Cache BVH** and the **Static BVH** options in the **Performance** tab. It allows us to optimize the render time.
5. You can make a test by just rendering a frame (*F12*). Pay attention to the time it takes to complete the rendering process for only one frame. Thus, you can calculate the time needed for a shot.
6. In the **Passes** tab (**Properties** | **Render Layers** | **Passes**), we will verify whether **Combined** and **Z** passes are checked.

   ### Note

   Passes in Cycles

   Passes are a decomposition of the 3D image rendered in Cycles. We can also render passes with Blender Internal but in a different way. Once the rendering calculation is over, we can combine these passes and combine each pass together to create the final image with directly compositing in Blender or another software. The passes allow us to get a control to considerably improve an image and make changes even after the image is rendered. So, it gives more fine-tuning opportunities and saves us a lot of render time.

   If you want to explore all the passes of Cycles, visit this link:

   http://wiki.blender.org/index.php/Doc:UK/2.6/Manual/Render/Cycles/Passes

7. Now that the image quality parameters are set, we must choose an output format of our animation. In the **Output** tab, we will choose the **OpenEXR MultiLayer** format.

   This format has the advantage of containing all active passes with a lossless compression. The passes are a decomposition of the rendered picture (Diffuse, Shadows, Ambient Occlusion, and so on). In our case, we are going to save some time by only rendering the combined and the Z passes. The combined pass corresponds to the final image with all the different passes already combined, and the Z pass gives us a black and white image corresponding to the depth of the scene.
8. In the **Output** tab, we must choose an **Output** path. We will write the following address: `//Render\01\`. We will press *Enter* to validate the address. The `//` symbols create a file just next to the blend file.

9. It is a raw render, so we uncheck the **Compositing** option in the **Post Processing** tab (**Properties**|**Processing**).

Note

OpenEXR

This is a high dynamic-range (HDR) image file format created and used for special effects in the VFX industry. It is now a standard format supported by most of 3D and compositing softwares. The OpenEXR Multilayer format is a variation. It can hold unlimited layers and passes.

If you want more information about OpenEXR in Blender, visit this link:

http://blender.org/manual/data_system/files/image_formats.html#openexr

You are now ready to render the animation. You can press the **Animation** button to start rendering. We must repeat this process for each shot.

Enhance a picture with compositing

Now that we have a raw render, it is time to learn how to improve it using the compositing tools of Blender.

Introduction to nodal compositing

Blender is a complete tool that also allows compositing. This is the ability to edit an image or a sequence after the rendering phase. You probably have already tried compositing, maybe unknowingly. For example, Adobe Photoshop© is a software that allows us to composite a single image. Unlike Adobe Photoshop©, Blender uses a nodal system that provides a great flexibility. We can make changes at any point without the loss of information. Let's try this:

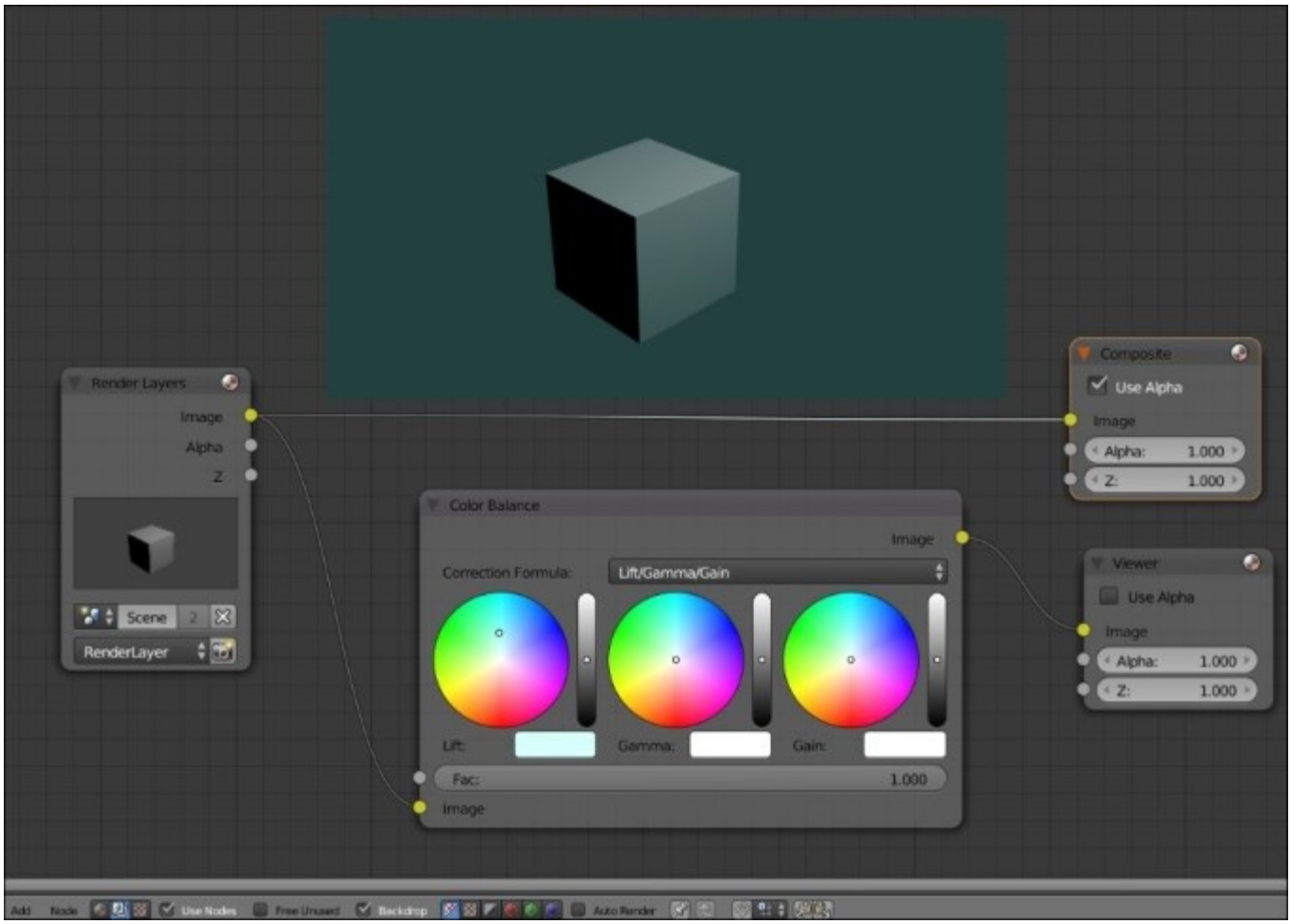
1. For a first approach of nodal compositing, let's open a new Blender scene.
2. In order to access the compositing mode, you must open the **Node Editor**. This is the same as the Node Editor for materials.
3. We must then check the **Compositing** button near the **Shader** button in the **Header** options. It is a small icon button symbolizing an image over another.
4. We must also check the **Use Nodes** button.

We have now two nodes, a Render Layer node and a Composite node.

1. We can split our scene to open the 3D View and make a render of the cube in the middle of the scene (**F12**).
2. We can also add a **Viewer** node (press *Shift* + *A* and select **Output** | **Viewer**). This node will allow us to visualize the compositing result directly in the **Node Editor**. You only need to connect the **Image** output socket of the **Render Layer** node to the **Image** input socket of the **Viewer** node and check the **Backdrop** option of the **Header**.

Now the render image appears behind the nodes. It will be pretty useful to do compositing in full screen. If you want to move the render image, use *Alt* and MMB. Two other interesting short keys are *V* to zoom in and *Alt* + *V* to zoom out.

1. We will add a **Color Balance** (press *Shift* + *A* and select **Color** | **Color Balance**) and connect it between the **Render Layers** node (to the **Image** output socket) and the **Viewer** node (to the **Image** input socket).
2. If we change the lift color, the render image is directly updated.



3. We can also replace the render of the cube by any other picture, by adding an **Image** node (press *Shift* + *A* and select **Input** | **Image**), and connecting the **Image** output socket to the **Image** input socket of the **Color Balance** node.

Note

Looking at a texture node through the Viewer

There is a node that can help you to better visualize what the compositing looks like at a certain point. You can press *Ctrl* + *Shift* and right-click on any node to append a ViewNode and connect to it.

The possibilities of compositing in Blender are enormous. For instance, you can easily use keying techniques that are often needed in the movie industry. It consists of replacing a green or a blue screen behind an actor by a virtual set. Compositing is an art and it take too long to explain everything, but we will see some of its basic concepts so that we can improve the shots of our sequence.

Now, we are going to work on the first shot of the sequence:

1. As with any other image file, we must add an **Image** node (press *Shift* + *A* and select **Input** | **Image**) and connect it to the viewer.

2. We press the **Open** button of the **Image** node, and we take the corresponding OpenEXR Multilayer file. We also check **Auto-refresh**.

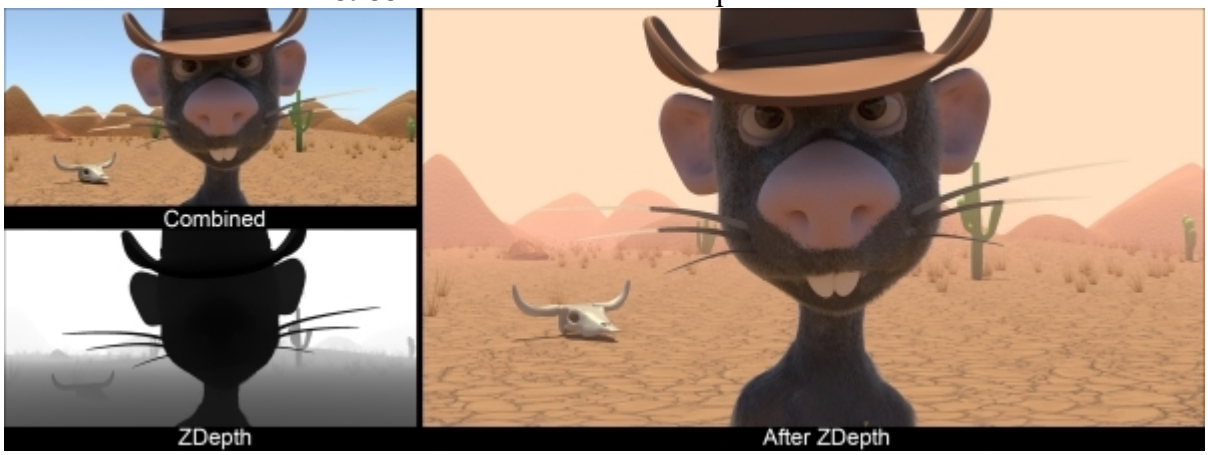
Depth Pass

The following is the combined output socket of the Image node, and there is a Depth output socket. This pass will allow us to simulate an atmospheric depth. It is an effect that can be observed when we look at a distant landscape and a kind of haze is formed. The Depth pass is a visual representation of the Z-Buffer on a grayscale. The objects near the camera will have a gray value close to black, unlike the distant elements, which will have a value close to white. This pass could serve for other things such as masking or blurring the focal depth. It all depends on the context. A controlled atmospheric effect may bring realism to the image.

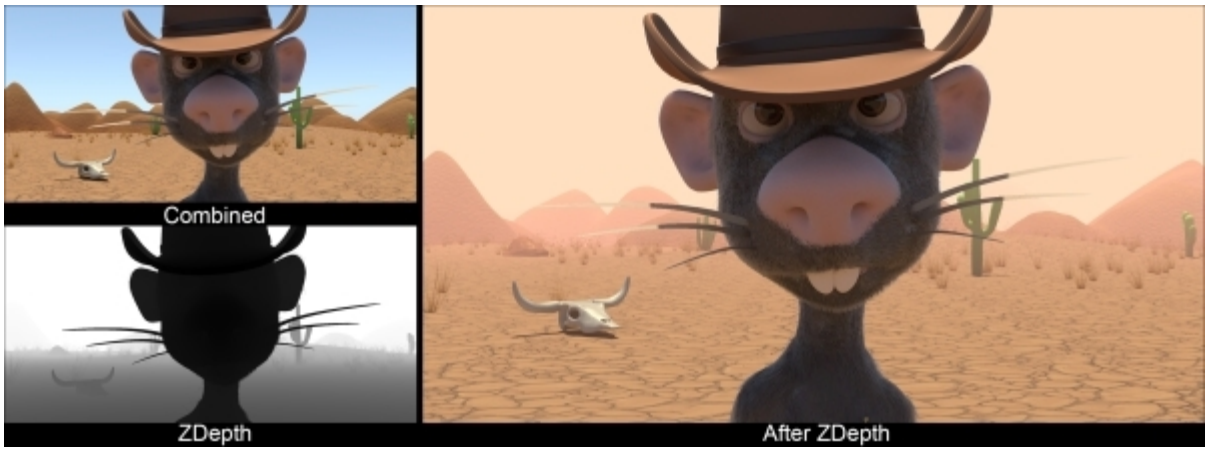
1. We will start by adding a **Normalize** node (press *Shift* + *A* and select **Vector** | **Normalize**). This allows us to clamp all pixel values between 0 and 1. We cannot visualize the Depth pass without a Normalize node.
2. We will add **RGB Curves** (press *Shift* + *A* and select **Color** | **RGB Curves**) to change the contrast of the ZDepth pass and control its strength effect.
3. We will then add a **Mix** node (press *Shift* + *A* and select **Color** | **Mix**) with a **Mix** blend mode.

Now that we have added the nodes, we are going to connect them as follows:

1. We will need to connect the **Z** output of the **Image** node to the input of the **Normalize** node and the output of the **Normalize** to the **Image** input of the **RGB Curves** node.
2. We will also connect the **Image** output of the **RGB Curves** node to the **Fac** input of the **Mix** node and the **Combined** output socket of the **Image** node to the first **Image** input socket.
3. We must adjust the RGB Curves node by adding another point to the curve. The point is located at **X: 0.36667** and **Y: 0.19375**.
4. We will also need to change the color of the second **Image** input socket of the **Mix** node. The hex code of the color is **D39881**. It will color the white pixels.



A render before and after the ZDepth pass



Color correction of the shot

One of the most important aspects of compositing is color calibration. Fortunately, there are easy-to-use tools in Blender to do that.

1. In order to quickly change the hue, we will add a **Color Balance** node (press *Shift +A* and select **Color** | **Color Balance**). The hue corresponds to the color tint of the image.
2. We must be very careful to slightly change the value of **Lift**, **Gamma**, and **Gain**. They become strong quickly. The RGB values of Lift are **R: 1.000**, **G: 0.981**, and **B: 0.971**. The RGB values of Gamma are **R: 1.067**, **G: 1.08**, and **B: 1.068**. The RGB values of Gain are **R:1.01**, **G: 0.998**, and **B: 0.959**.

Note

There are two correction formulas: **Lift/Gamma/Gain** and **Offset/Power/Slope**. These are the two ways to get the same result. For each one, there are three color wheels and a value controller (**Fac**). You can modify the darker, the mid-tone, and the highlight values separately.

If you want more information about the color balance node in Blender, visit this link :

https://www.blender.org/manual/composite_nodes/types/color/color_balance.html

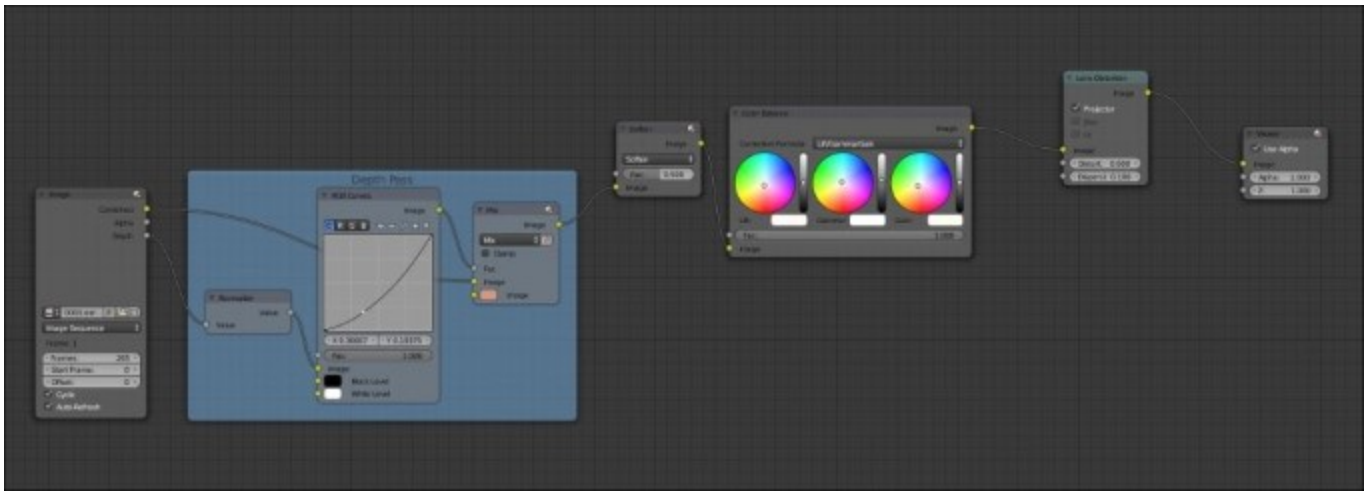
A render image with an adjustment of the **Color Balance** node will look like this:

Before finishing the compositing, let's add a few effects.

Adding effects

We will add a **Filter** node (press *Shift* + *A* and select **Filter** | **Filter**). We must connect the **Image** output socket of the **Mix** node to the **Image** input socket of the **Soften** node, and we will connect the image output socket of the **Soften** node to the **Image** input socket of the **color balance** node**.** We will keep the filter type to **Soften** with **Fac** of **0.500**. This blends the pixels so that the image is less sharp. A photo is never perfectly sharp.

We will add then a **Lens distortion** node (press *Shift* + *A* and select **Distort** | **Lens Distortion**). We must connect the **Image** output socket of the **Color Balance** to the **Image** input socket of the **Lens Distortion** node and the **Image** output socket of the **Lens Distortion** node to the **Image** input socket of the **Viewer** node. We will check the **Projector** button. The **distort** option is at **0.000,** and the **dispersion** option at **0.100**. This node usually allows us to make a distortion effect such as a fish eye, but in our case, it will allow us to make a chromatic aberration. This adds a soft and nice effect.



The nodes of compositing

We have now completed the appearance of the shot.

A render with final compositing

Compositing rendering phase

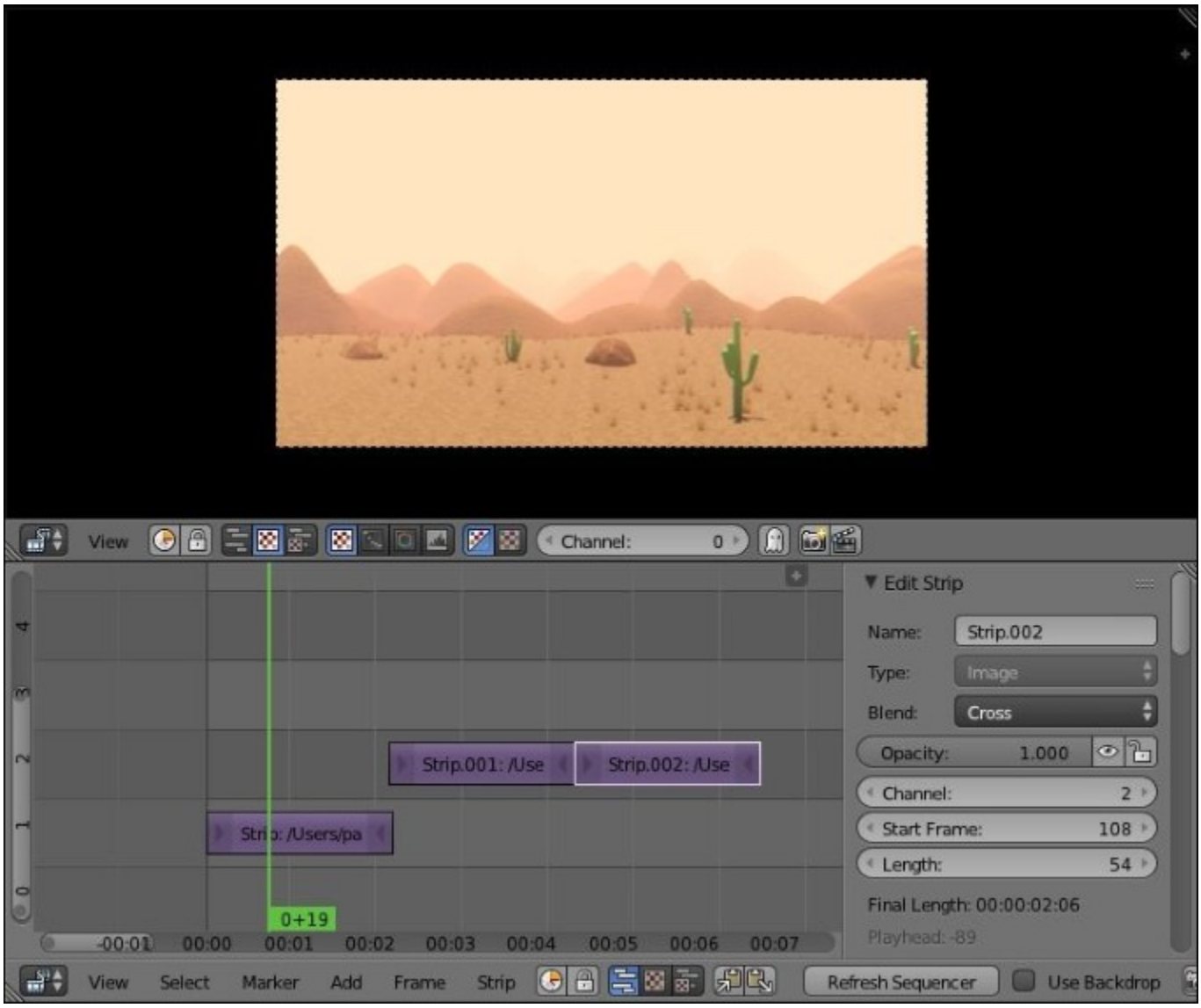
We are now ready to make the render of our compositing:

1. In the **Output** options (**Properties** | **Render** | **Output**), we must change the Output path. We will write the following address: //**Render\01\Compositing**. We will press *Enter* to validate the address.

2. We will also change the Output format to **TGA**.
3. It is a compositing render, so we will check the **Compositing** option in the **Post Processing** tab (**Properties** | **Processing**).
4. Now, we are ready to render the scene. We will use the same process for each scene.

Editing the sequence with the VSE

Now that we've rendered and done some compositing on each shot, it's time to bring back the whole sequence in one final place. In this section, we are going to do a basic video editing with the VSE.



Two VSE, one is set to the Image Preview (top), the other to the Sequencer (bottom)

Introduction to the Video Sequence Editor

The **VSE** or Video Sequence Editor is a method of video editing in Blender. It is really simple to use and could be very powerful. The best way to use it is to use the **Video Editing** layout located in the menu bar. We usually don't use the Graph editor here, so we can join it back. We have now an interface with two Video Sequence Editors and the Timeline at the bottom. On the head of the VSE, we have three icons that are used to display **Sequencer**, the place where we edit strips, and **Image Preview,** where we see the result of our editing, or both. In Sequencer, we can add different types of strips with *Shift + A*. We are mainly using **Image**, **Movie**, or **Sound**. We can import Image Sequences with the **Image** option. You can select a strip with RMB and move it with *G*. When you have a strip selected, you have access to two buttons to the left and right represented with arrows that define the start and the end of the strip. You can cut a strip by placing the timeline where you want the split to be and pressing *K* (for knife).

We are not going to go deep into every setting of the VSE, but for each selected strip, you have some options in the right panel of the editor (*N*), such as the **opacity** of an image or movie strip or the volume of an **audio** strip. Of course, you can animate each option by right-clicking on them and choosing the **Insert Keyframe** option, or by simply pressing *I* while hovering over them. You can press *Ctrl* to snap a selected strip to another strip. You can also use the *Shift + D* shortcut in order to duplicate a selected strip.

Edit and render the final sequence

Let's now create the editing of our sequence with the shots that we've composited and rendered before:

1. In order to edit our sequence, we will open a new fresh file. We will also change the layout of our interface so that we have two **VSEs**, one with **Image Preview** and the other with **Sequencer**.
2. Now, we can add our first shot by pressing *Shift + A* in the sequencer and by choosing **Image**. In the file browser, we will go to the **Render** folder and select the **01-compositing** folder in order to select every `.targa` file with *A*. We will now have a new strip that corresponds to the first shot. We will repeat the same process with the other shots.
3. Now, we can move each shot one after the other to create continuity. Be sure that each strip is snapped to the previous one by pressing the *Ctrl* key while moving them.
4. We will also need to readjust the animation start and end in the timeline from the beginning of sequence to its end frame.
5. If you want, you can add **Sound** strips in order to add music or sound effects.
6. We are now ready to render our final edited sequence. To do this, we will change the **Output** path in the **Render** tab of the Properties editor, and we will choose the **H.264** file type. In the **Encoder** section, we will use the **Quicktime** type, and we will, finally, press the render button.

Summary

First, congratulation for arriving at this point of the module; we hope you've learned a lot of things and that you can now realize all the ideas that you ever dreamed about. In this last chapter, we learned how to finalize our sequence by creating advance materials. We also learned how to use the particle system to create a complex fur. Then we set up our render with the OpenEXR MultiLayer format and discovered what passes are. After this, we saw the power of nodal compositing by changing the color balance and adding effects to our shot. As a bonus, we learned how to use the VSE in order to edit our full sequence. Be aware that we didn't explain a lot of things, and you can go deeper into each subject. We recommend that you practice a lot and skim the web and the other books of the PacktPub collection in order to extend your knowledge. Remember, you can learn a tool, but creativity is one of the most important things in this field. We wish you a successful continuation.

Part 2. Module 2

Blender 3D Cookbook

Build your very own stunning characters in Blender from scratch

Chapter 1. Modeling the Character's Base Mesh

In this chapter, we will cover the following recipes:

- Setting templates with the Images as Planes add-on
- Setting templates with the Image Empties method
- Setting templates with the Background Images tool
- Building the character's base mesh with the Skin modifier

Introduction

In this chapter, we are going to do two things: set up templates to be used as a reference for the modeling, and build up a base mesh for the sculpting of the character.

To set up templates in a Blender scene, we have at least three different methods to choose from: the **Images as Planes** add-on, the **Image Empties** method, and the **Background Images** tool.

A base mesh is usually a very low poly and simple mesh roughly shaped to resemble the final character's look. There are several ways to obtain a base mesh: we can use a ready, freely downloadable mesh to be adjusted to our goals, or we can model it from scratch, one polygon at a time. What's quite important is that it should be made from all quad faces.

To build the base mesh for our character, we are going to use one of the more handy and useful modifiers added to Blender: the **Skin** modifier. However, first, let us add our templates.

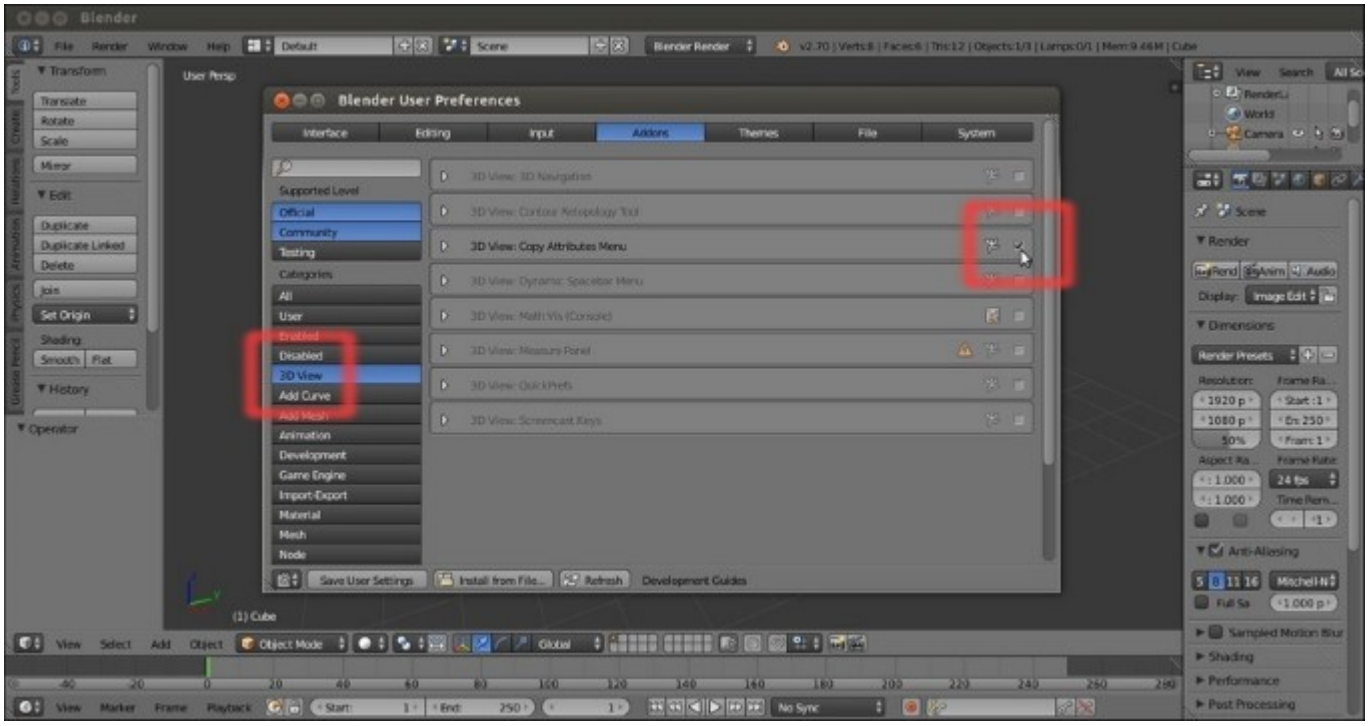
Setting templates with the Images as Planes add-on

In this recipe, we'll set the character's templates by using the **Images as Planes** add-on.

Getting ready

The first thing to do is to be sure that all the required add-ons are enabled in the preferences; in this first recipe, we need the **Images as Planes** and **Copy Attributes Menu** add-ons. When starting Blender with the factory settings, they appear gray in the **User Preferences** panel's **Add-ons** list, meaning that they are not enabled yet. So, we'll do the following:

1. Call the **User Preferences** panel (*Ctrl* + *Alt* + *U*) and go to the **Add-ons** tab.
2. Under the **Categories** item on the left-hand side of the panel, click on **3D View**.
3. Check the empty little checkbox on the right-hand side of the **3D View: Copy Attributes Menu** add-on to enable it.
4. Go back to the **Categories** item on the left-hand side of the panel and click on **Import-Export**.
5. Scroll down the add-ons list to the right-hand side to find the **Import-Export: Import Images as Planes** add-on (usually, towards the middle of the long list).
6. Enable it, and then click on the **Save User Settings** button to the left-bottom of the panel and close it.



The User Preferences panel with the Categories list and the Addons tab to enable the several add-ons

There are still a few things we should do to prepare the 3D scene and make our life easier:

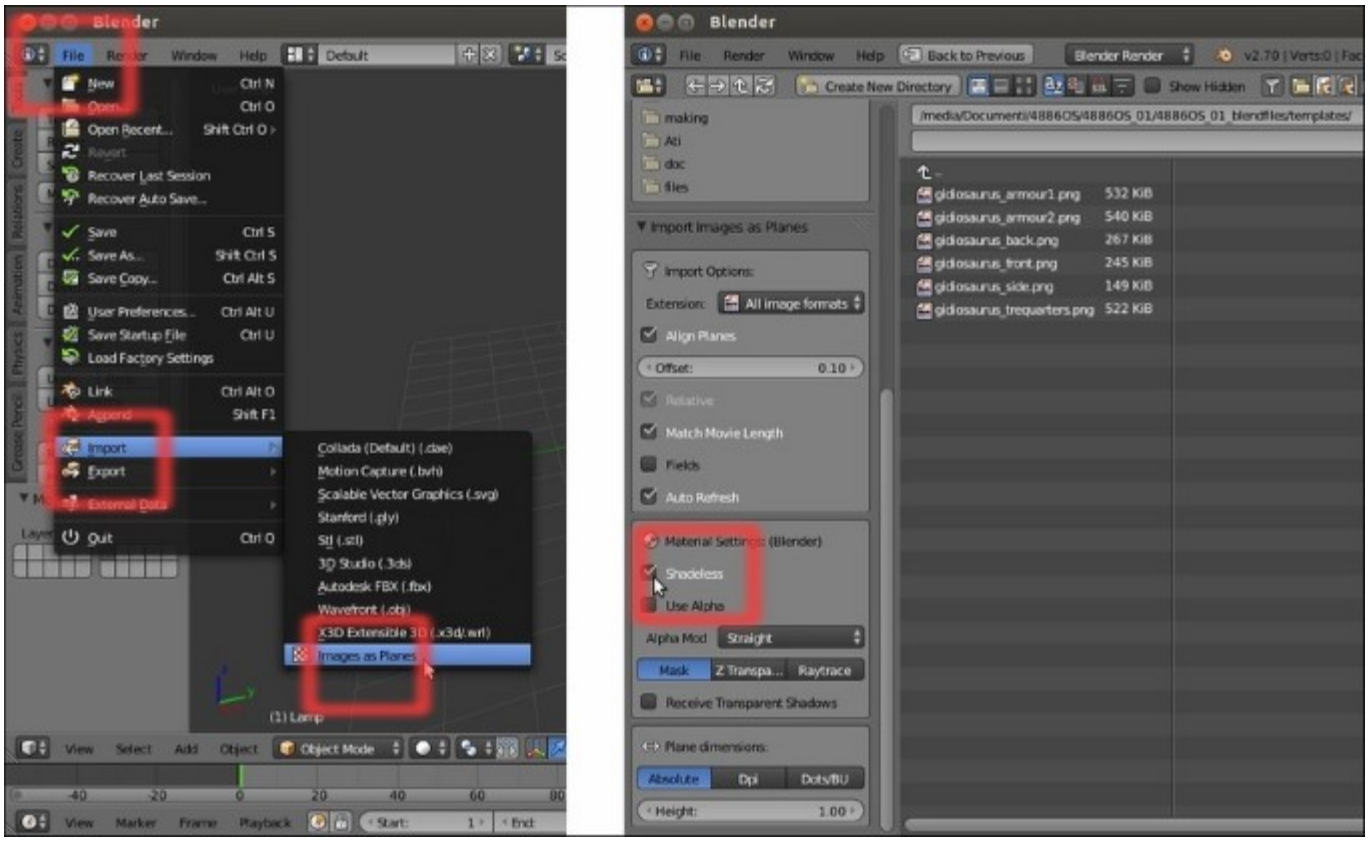
7. Delete the already selected **Cube** primitive.
8. Select the **Lamp** and the **Camera** and move them on to a different layer; I usually have them on the **sixth** layer (*M* key), in order to keep free and empty both the first and second rows of the left layer's block.
9. The **Outliner** can be found in the top-right corner of the default workspace. It shows a list view of the scene. Set **Display Mode** of the **Outliner** to **Visible Layers**.
10. Lastly, save the file as `Gidiosaurus_base_mesh.blend`.

How to do it…

Although not strictly necessary, it would be better to have the three (at least in the case of a biped character, the **Front**, **Side**, and **Back** view) templates as separated images. This will allow us to load a specific one for each view, if necessary. Also, to facilitate the process, all these images should be the same height in pixels.

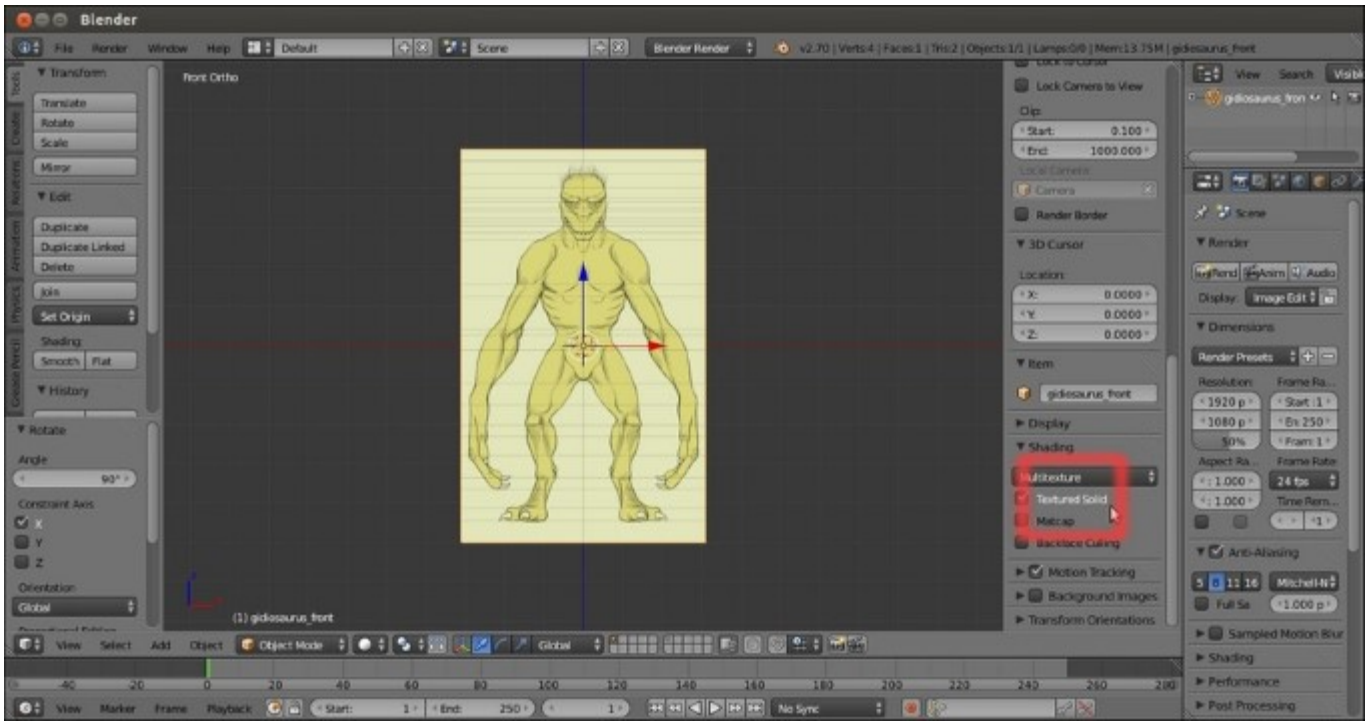
In our case, the required three views are provided for you in the files that accompany this module. You will find them in the `templates` folder. The **Import Images as Planes** add-on will take care of loading them into the scene:

1. Left-click on **File** | **Import** | **Images as Planes** in the top-left menu on the main header of the Blender UI.
2. On the page that just opened, go to the **Material Settings** column on the left-hand side (under the **Import Images as Planes** options) and enable the **Shadeless** item. Then, browse to the location where you placed your `templates` folder and load the `gidiosaurus_front.png` image:



The Import pop-up menu and the material settings subpanel of the Import Images as Planes add-on

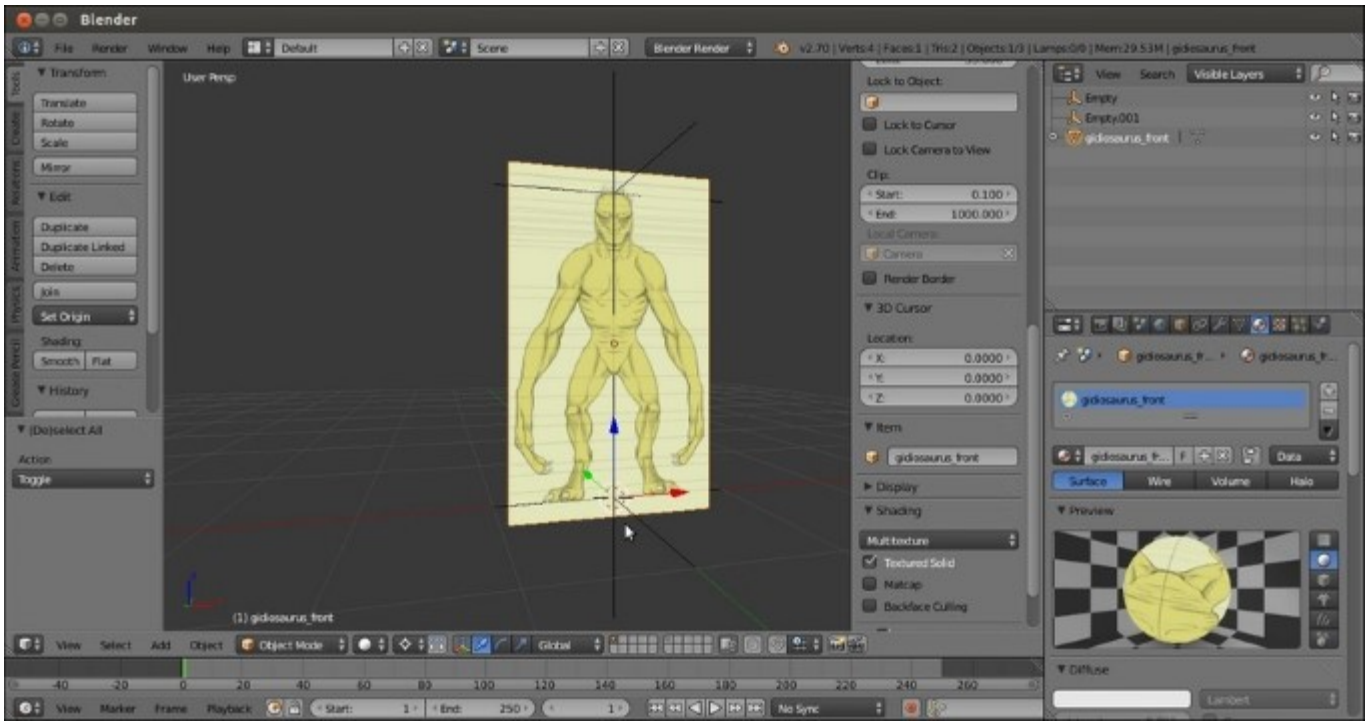
3. Rotate 90 degrees on the *x* axis (*R* | *X* | **90** | *Enter*) of the **Plane** that just appeared at the center of the scene (at the **3D Cursor** location, actually; to reset the position of the **3D Cursor** at the center of the scene, press the *Shift* + *C* keys).
4. Press *N* to call the **Properties** sidepanel on the right-hand side of the active 3D window, and then go to the **Shading** subpanel and enable the **Textured Solid** item.
5. Press *1* on the numpad to go to the **Front** view:



The imported plane with the relative UV-mapped image

Now, we know that our **Gidiosaurus** is a **2.5** meters tall beast. So, assuming that **1 Blender Unit** is equal to **1 meter**, we must scale the plane to make the character's front template **two and a half Blender Units** tall (Note that it is not the plane that must be 2.5 units tall, it's the character's shape inside the plane).

6. Add an **Empty** to the scene (*Shift* + *A* | **Empty** | **Plain Axes**).
7. Duplicate it and move it **2.5** units up on the *z* axis (*Shift* + *D* | *Z* | **2.5** | *Enter*).
8. Go to the **Outliner** and click on the arrows on the side of the names of the two **Empties** (**Empty** and **Empty.001**), in order to make them gray and the **Empties** not selectable.
9. Select the **Plane** and move it to align the bottom (feet) guideline to the horizontal arm of the first **Empty** (you actually have to move it on the *z* axis by **0.4470**, but note that by pressing the *Ctrl* key, you can restrict movements to the grid and with *Ctrl* + *Shift*, you can have even finer control).
10. Be sure that the **3D Cursor** is at the object origin, and press the period key to switch *Pivot center for rotation/scaling* to the **3D Cursor**.
11. Press *S* to scale the **Plane** bigger and align the top-head guideline to the horizontal arm of the second **Empty** (you have to scale it to a value of **2.8300**):



The properly scaled plane in the 3D scene

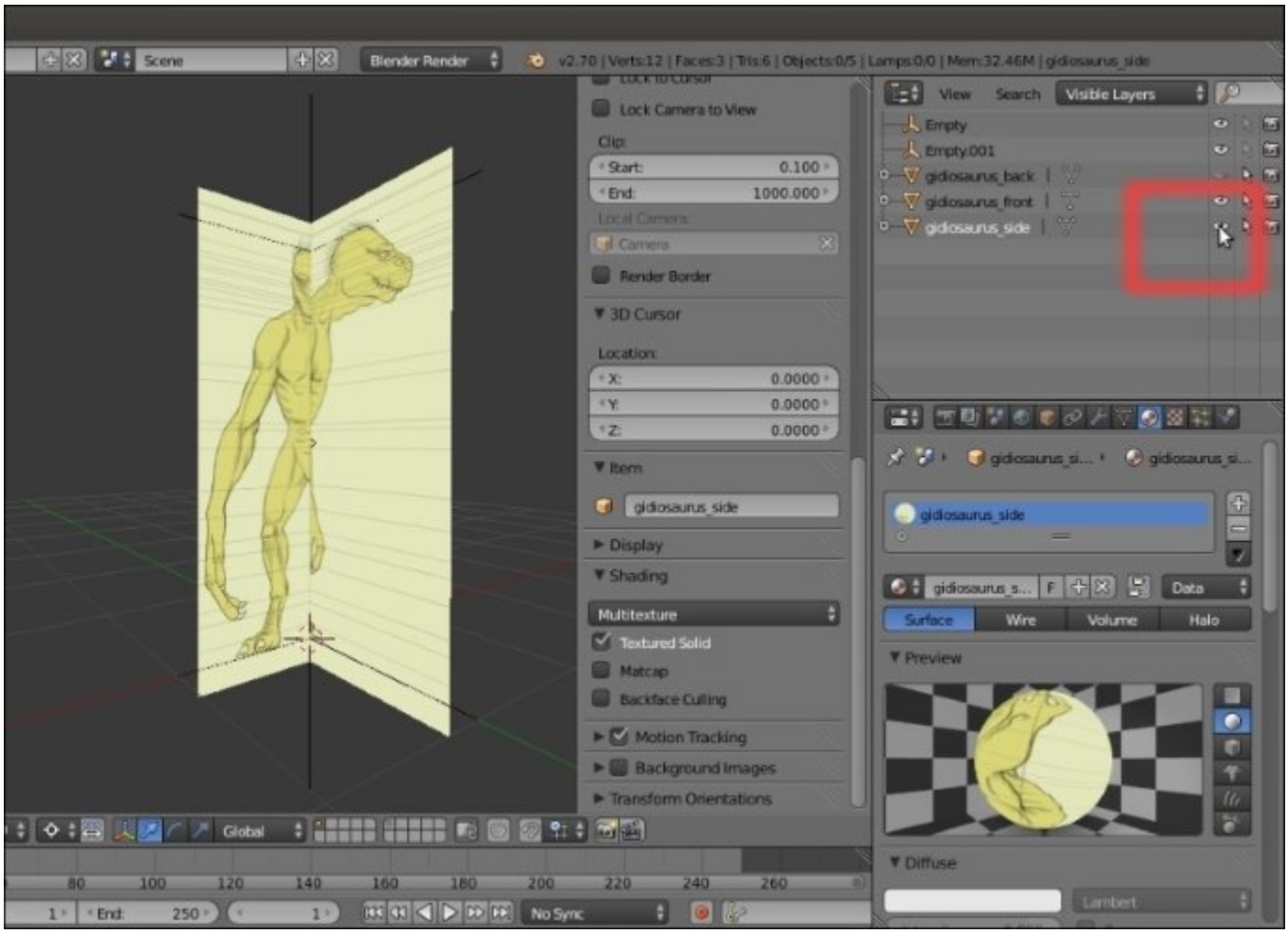
12. Left-click again on **File** | **Import** | **Images as Planes** in the top-left menu on the main header of the Blender UI.
13. Browse to the location where you placed your `templates` folder and this time load the `gidiosaurus_side.png` image.
14. *Shift* + right-click on the first **Plane** (`gidiosaurus_front.png`) to select it and make it the active one. Then, press *Ctrl* + *C* and from the **Copy Attributes** pop-up menu, select **Copy Location**.
15. Press *Ctrl* + *C* again and this time select **Copy Rotation**; press *Ctrl* + *C* one more time and select **Copy Scale**.
16. Right-click to select the second **Plane** (`gidiosaurus_side.png`) in the 3D view, or click on its name in the **Outliner**, and rotate it 90 degrees on the *z* axis (*R* | *Z* | **90** | *Enter*).
17. Optionally, you can move the second **Plane** to the second layer (*M* | second button on the **Move to Layer** panel).
18. Again, left-click on **File** | **Import** | **Images as Planes**, browse to the `templates` folder, and load the `gidiosaurus_back.png` image.
19. Repeat from step 12 to step 15 and move the third **Plane** on a different layer.
20. Save the file.

How it works…

We used a Python script, which is an add-on, to import planes into our scene that are automatically UV-mapped with the selected image, and inherit the images' height/width aspect ratio.

To have the textures/templates clearly visible from any angle in the 3D view, we have enabled the **Shadeless** option for the **Planes** materials; we did this directly in the importer preferences. We can also set each material to shadeless later in the **Material** window.

We then used another add-on to copy the attributes from a selected object, in order to quickly match common parameters such as location, scale, and rotation:



The template planes aligned to the x and y axis (Front and Side views)

The imported **Planes** can be placed on different layers for practicality; they can also be on a single layer and their visibility can be toggled on and off by clicking on the eye icon in the **Outliner**.

Setting templates with the Image Empties method

In this recipe, we'll set the character's templates by using **Image Empties**.

Getting ready

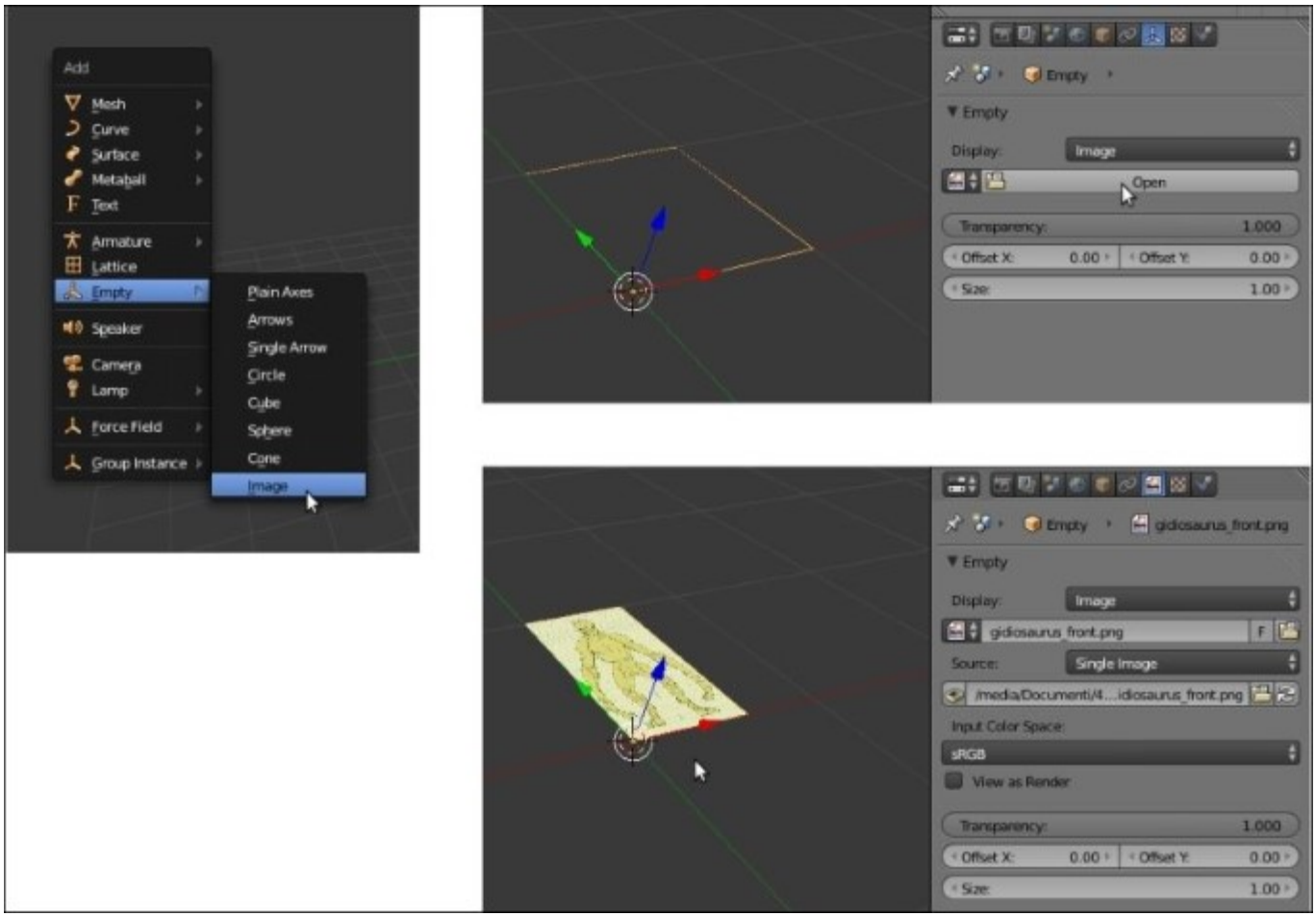
For this and the following recipes, there is no need for any particular preparations. Anyway, it is handy to prepare the two **Empties** to have markers in the 3D view for the **2.5** meters height of the character; so we'll do the following:

1. Start a brand new Blender session and delete the already selected **Cube** primitive.
2. Select the **Lamp** and **Camera** and move them on a different layer; I usually have them on the **sixth** layer, in order to keep free and empty both the first and second rows of the left layer's block.
3. Add an **Empty** to the scene (*Shift* + *A* | **Empty** | **Plain Axes**).
4. Duplicate it and move it **2.5** units up on the *z* axis (*Shift* + *D* | *Z* | **2.5** | *Enter*).
5. Go to the **Outliner** and click on the arrows on the side of the names of the two **Empties** (**Empty** and **Empty.001**), in order to make them gray and the **Empties** not selectable.
6. Save the file as `Gidiosaurus_base_mesh.blend`.

How to do it…

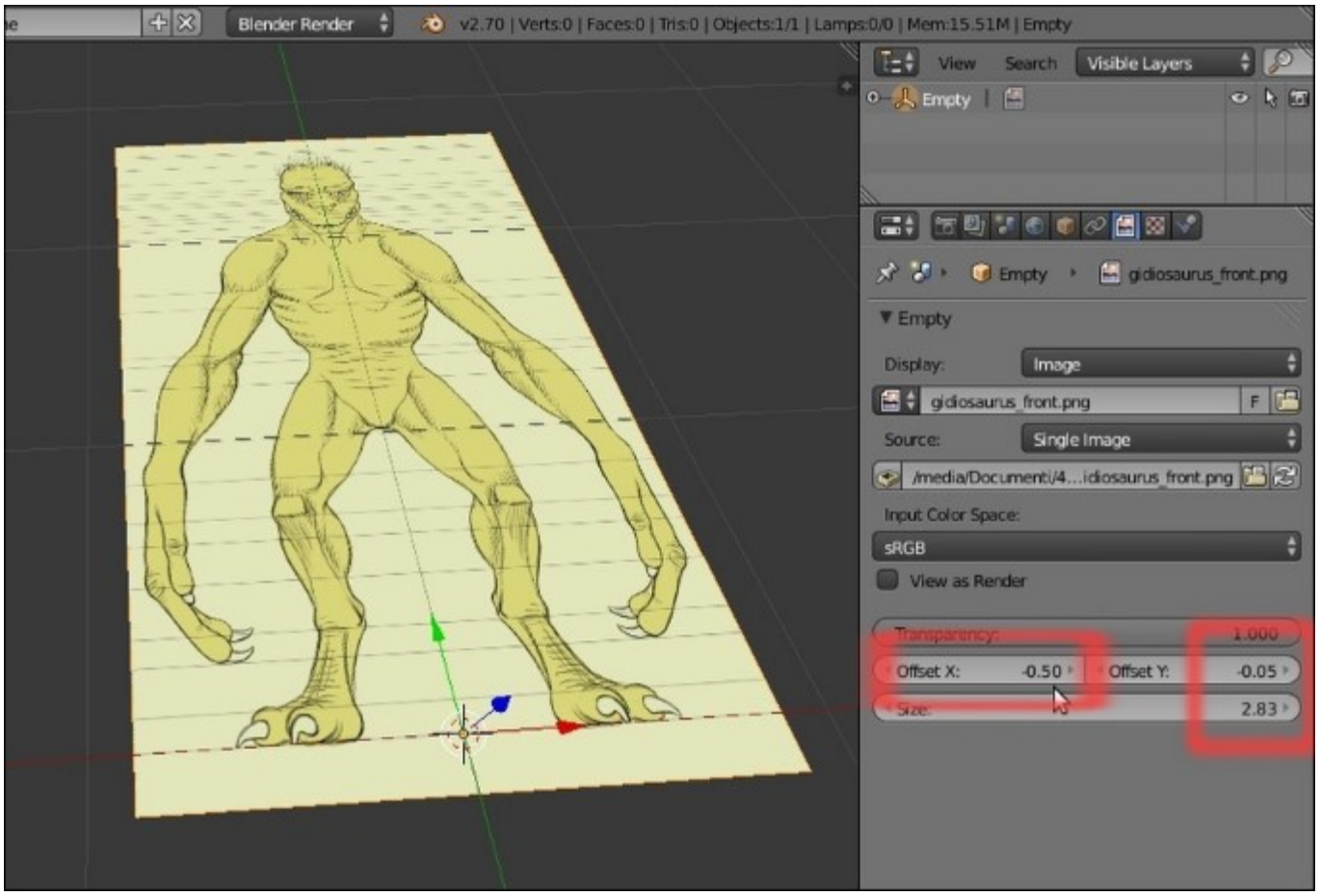
So, now we are going to place the first **Image Empty** in the scene:

1. Add an **Empty** to the scene (*Shift* + *A* | **Empty** | **Image**; it's the last item in the list).
2. Go to the **Object Data** window in the main **Properties** panel on the right-hand side of the Blender UI; under the **Empty** subpanel, click on the **Open** button.
3. Browse to the `templates` folder and load the `gidiosaurus_front.png` image.



The Add pop-up menu and the Image Empty added to the 3D scene, with the settings to load and set the image

4. Set the **Offset X** value to **-0.50** and **Offset Y** to **-0.05**. Set the **Size** value to **2.830**:

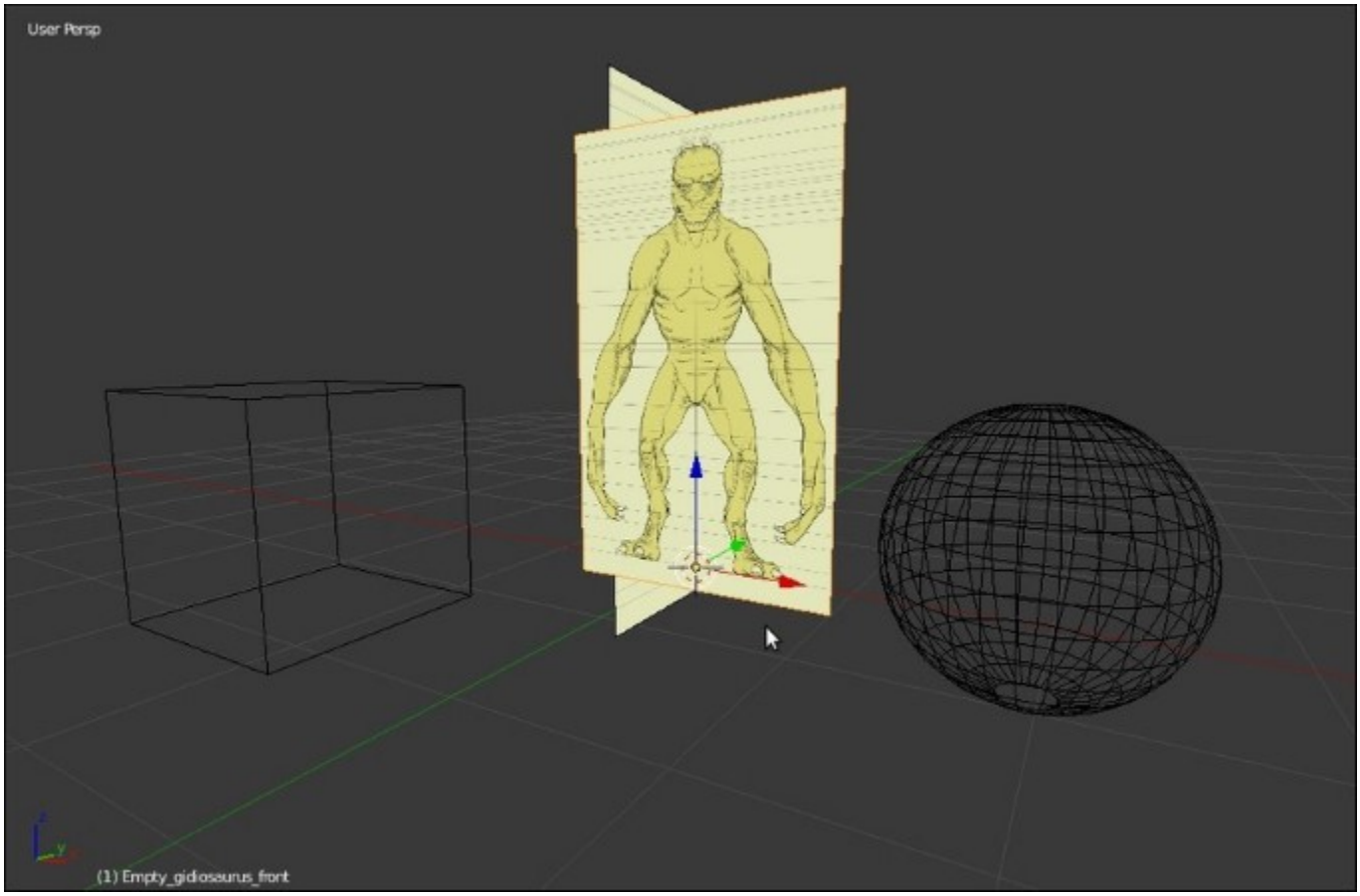


The Offset and Size settings

5. Rotate the **Empty** 90 degrees on the *x* axis (*R* | *X* | **90** | *Enter*).
6. Go to the **Outliner** and rename it `Empty_gidiosaurus_front`.
7. Duplicate it (*Shift* + *D*), rotate it 90 degrees on the *z* axis, and in the **Outliner**, rename it as `Empty_gidiosaurus_side`.
8. In the **Empty** subpanel under the **Object Data** window, click on the little icon (showing **3** users for that data block) on the right-hand side of the image name under **Display**, in order to make it a single user. Then, click on the little folder icon on the right-hand side of the image path to go inside the `templates` folder again, and load the `gidiosaurus_side.png` image.
9. Reselect **Empty_gidiosaurus_front** and press *Shift* + *D* to duplicate it.
10. Go to the **Empty** subpanel under the **Object Data** window, click on the little icon (showing **3** users for that datablock) on the right-hand side of the image name under **Display**, in order to make it a single user. Then, click on the little folder icon on the right-hand side of the image path to go inside the `templates` folder again, and this time load the `gidiosaurus_back.png` image.
11. Go to the **Outliner** and rename it `Empty_gidiosaurus_back`.

How it works…

We have used one of the most underrated (well, in my opinion) tools in Blender: **Empties,** which can show images! Compared to the **Images as Planes** add-on, this has some advantages: these are not 3D geometry and the images are also visible in the 3D view without the **Textured Solid** option enabled (under **Shading**) and in **Wireframe** mode.



The Image Empties appear as textured also in Wireframe viewport shading mode

Exactly, as for the imported **Planes** of the former recipe, the visibility in the 3D view of the **Image Empties** can be toggled on and off by clicking on the eye icon in the **Outliner**.

Setting templates with the Background Images tool

In this recipe, we'll set the character's templates by using the **Background Images** tool.

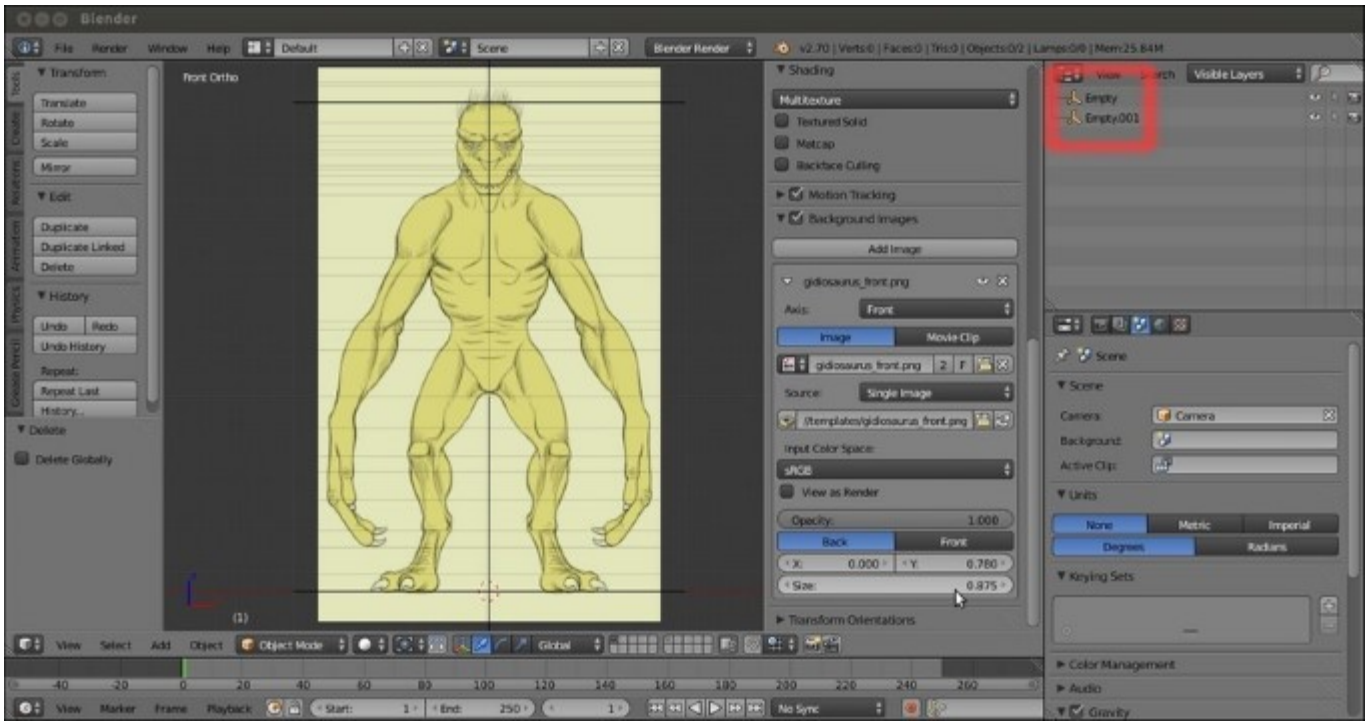
Getting ready

As in the former recipe, no need for any particular preparations; just carry out the preparatory steps as mentioned in the *Getting ready* section of the previous recipe.

How to do it…

So let's start by adding the templates as background images; that is, as reference images only visible in the background in **Ortho** view mode and, differently from the previous recipes, not as 3D objects actually present in the middle of the scene:

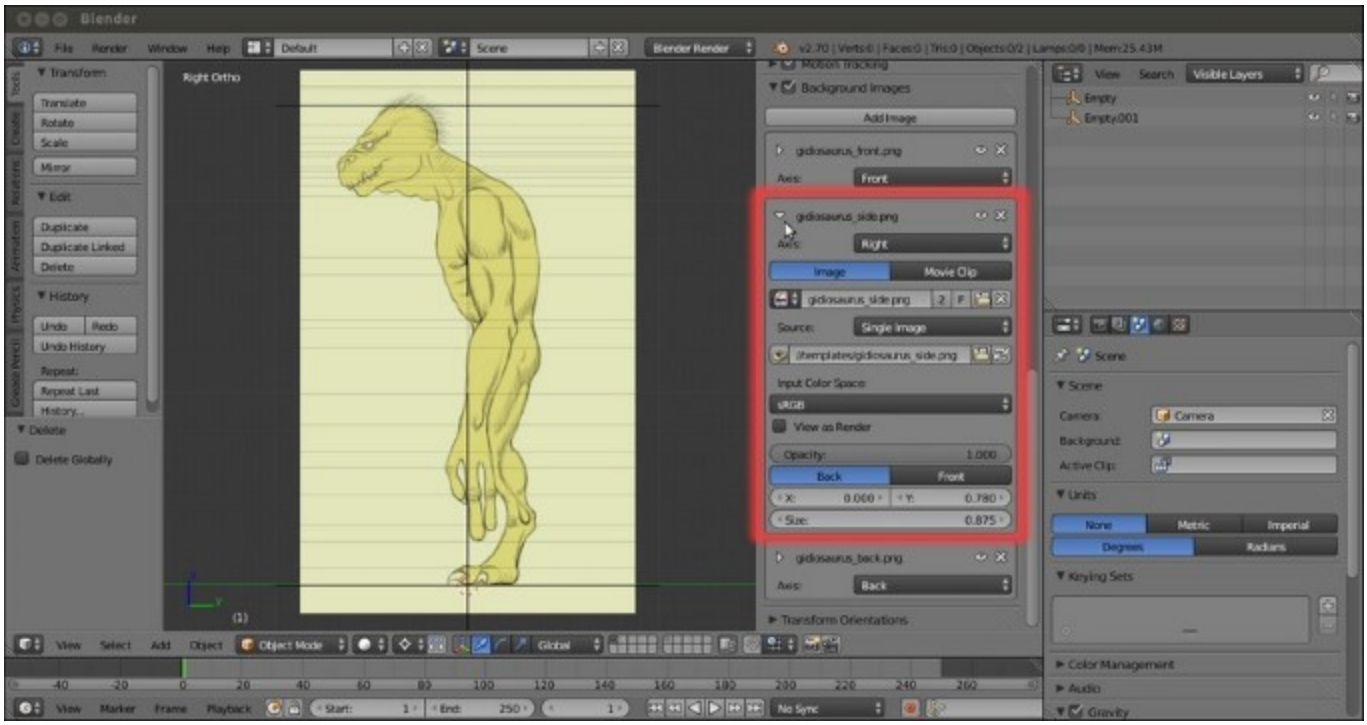
1. Press *1* on the numpad to switch to the orthographic **Front** view and press *Alt* + *Home* to center the view on the **3D Cursor**.
2. If not already present, press *N* to bring up the **Properties** sidepanel to the right-hand side of the 3D window; scroll down to reach the **Background Images** subpanel and enable it with the checkbox. Then click on the little arrow to expand it.
3. Click on the **Add Image** button; in the new option panel that appears, click on the **Open** button and browse to the `templates` folder to load the `gidiosaurus_front.png` image.
4. Click on the little window to the side of the **Axis** item and switch from **All Views** to **Front**, and then set the **Opacity** slider to **1.000**.
5. Increase the **Y** offset value to make the bottom/feet guideline of the reference image aligned to the horizontal arm of the first **Empty** (you have to set it to **0.780**).
6. Scale **Size** smaller, using both the **Empties** that we set as references for the **2.5** meters height of the creature (you actually have to set the **Scale** value to **0.875**).



The background image scaled and positioned through the settings in the N sidepanel

7. Click on the little white arrow on the top-left side of the `gidiosaurus_front.png` subwindow to collapse it.
8. Click on the **Add Image** button again; then, in the new option panel, click on the **Open** button, browse to the `templates` folder, and load the `gidiosaurus_side.png` image. Then, set the **Axis** item to **Right**, **Opacity** to **1.000**, **Scale** to **0.875**, and **Y** to **0.780**.
9. Repeat the operation for the `gidiosaurus_back.png` image, set **Axis** to **Back**, and so on.

Press *3* on the numpad to switch to the **Side** view, *1* to switch to the **Front** view, and *Ctrl + 1* to switch to the **Back** view, but remember that you must be in the **Ortho** mode (*5* key on the numpad) to see the background templates:



The N sidepanel settings to assign the background image to a view

Building the character's base mesh with the Skin modifier

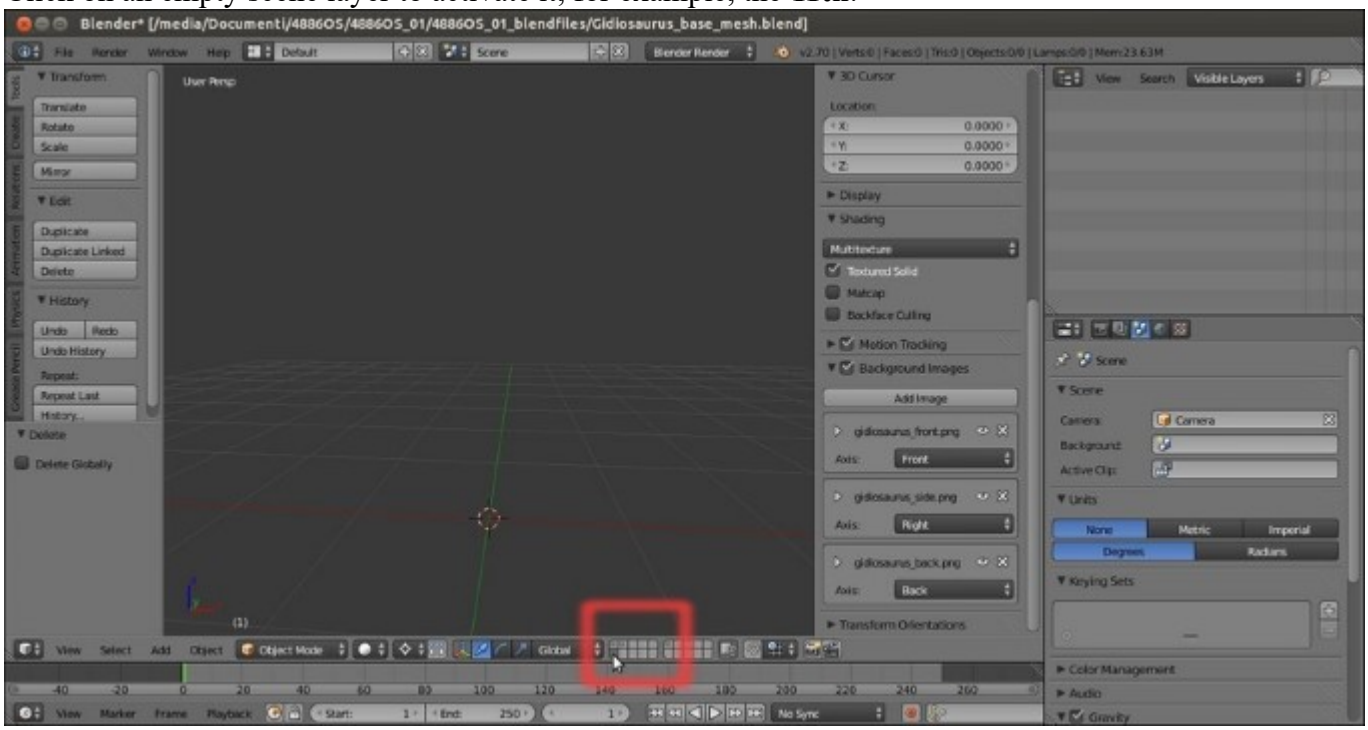
In the previous recipes, we saw three different ways to set up the template images; just remember that one method doesn't exclude the others, so in my opinion, the best setup you can have is: **Image Empties** on one layer (visibility toggled using the eye icons in the **Outliner**) together with **Background Images**. This way you can not only have templates visible in the three orthographic views, but also in the perspective view (and this can sometimes be really handy).

However, whatever the method you choose, now it's time to start to build the character's base mesh. To do this, we are going to use the **Skin** modifier.

Getting ready

First, let's prepare the scene:

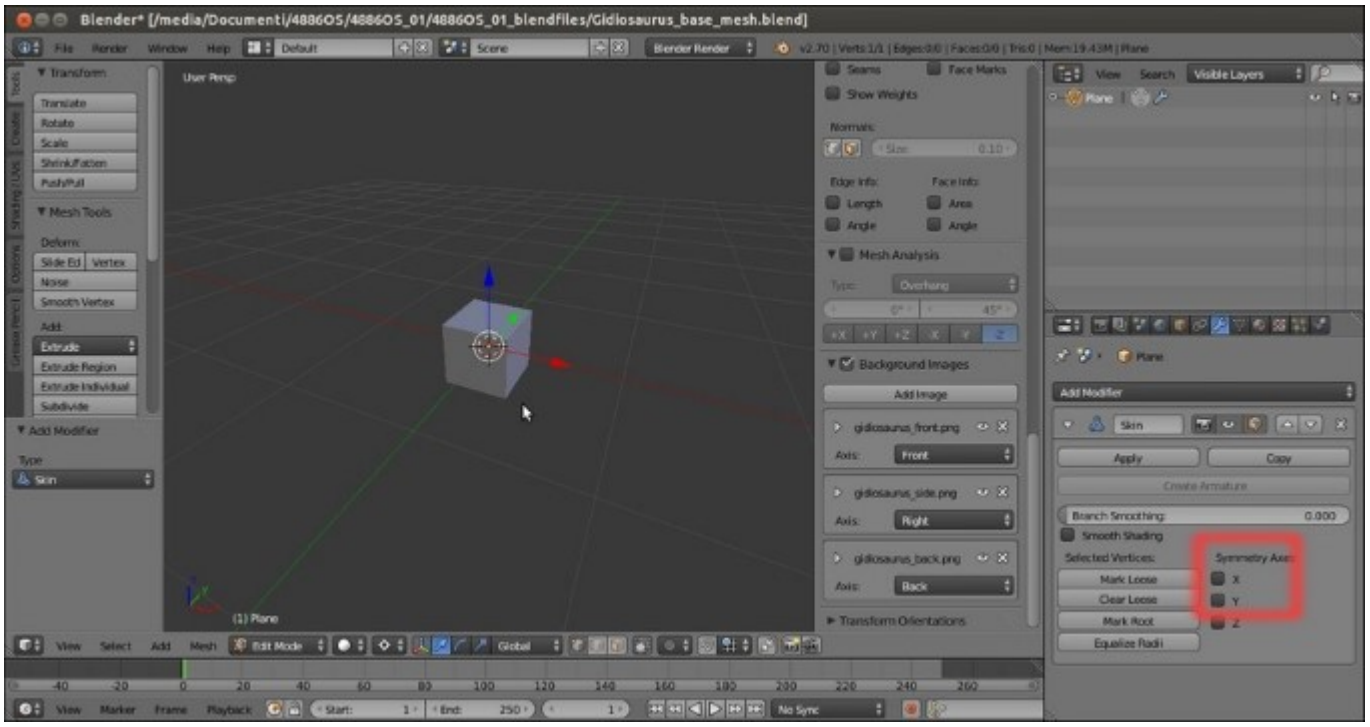
1. In case it's needed, reopen the `Gidiosaurus_base_mesh.blend` file.
2. Click on an empty scene layer to activate it; for example, the **11th**.



The starting empty scene and the scene layer's buttons on the 3D window toolbar

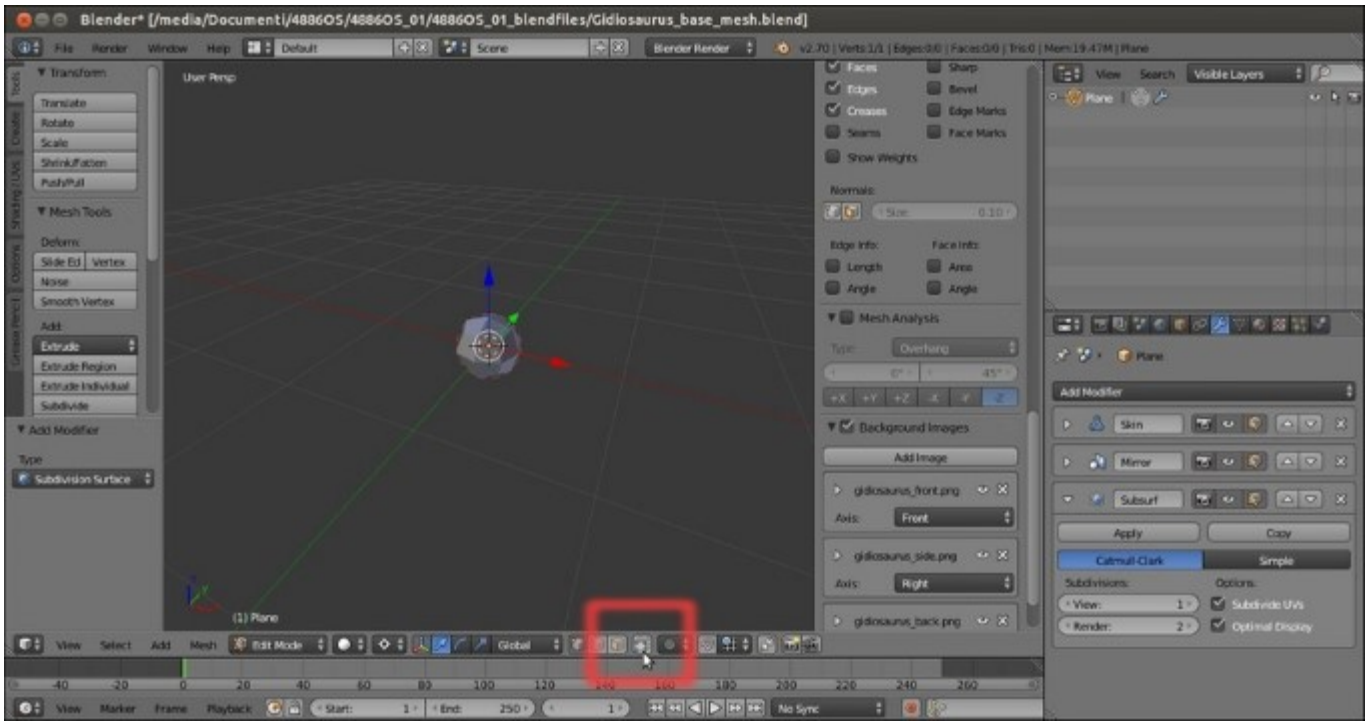
3. Be sure that the **3D Cursor** is at the center of the scene (*Shift* + *C*).
4. Add a **Plane** (press *Shift* + *A* and go to **Mesh** | **Plane**). If you are working with the **Factory Settings**, you must now press *Tab* to go in to **Edit Mode**, and then *Shift* + right-click to deselect just one vertex.

5. Press *X* and delete the three vertices that are still selected.
6. Right-click to select the remaining vertex and put it at the cursor location in the center of the scene (*Shift* + *S*, and then select **Selection to Cursor**).
7. Go to the **Object Modifiers** window on the main **Properties** panel, to the right-hand side, and assign a **Skin** modifier; a cube appears around the vertex. Uncheck **X** under **Symmetry Axes** in the modifier's panel:



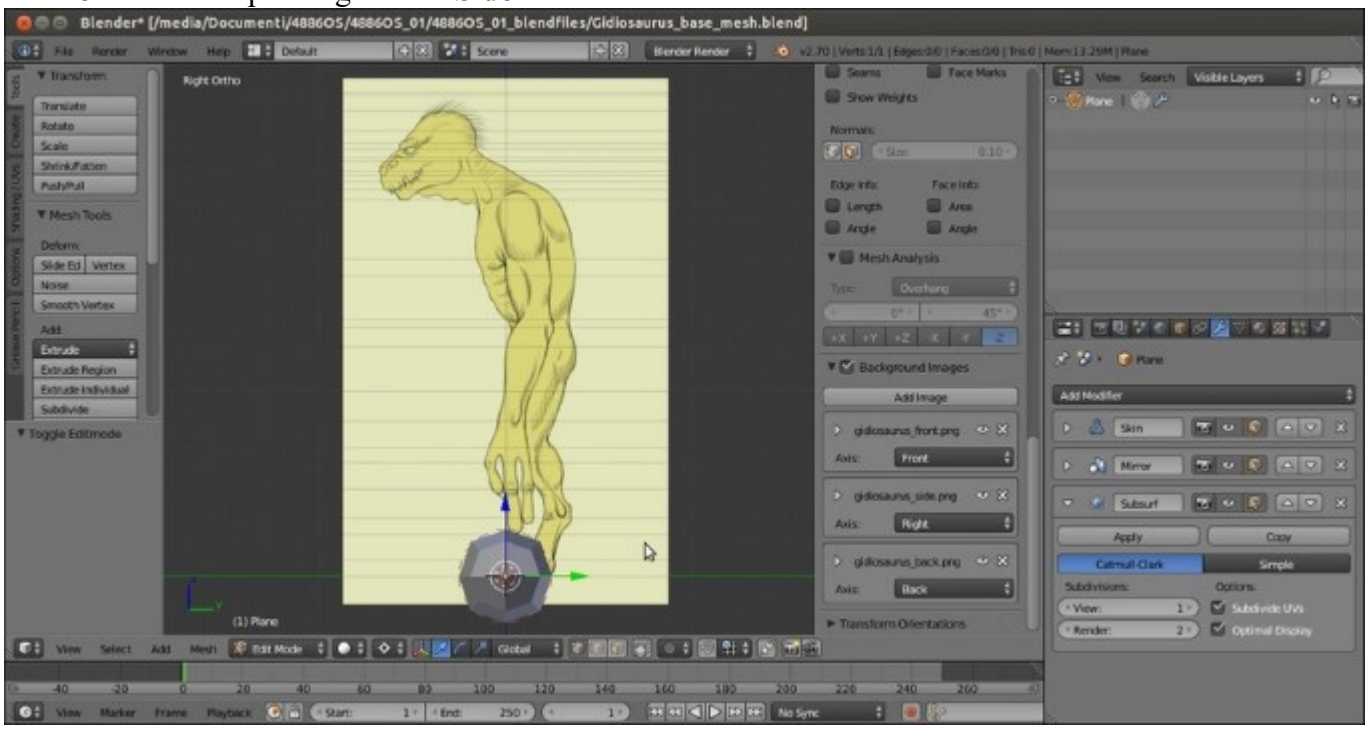
The cube geometry created by just one vertex and the Skin modifier

8. Assign a **Mirror** modifier and check **Clipping**.
9. Assign a **Subdivision Surface** modifier and check **Optimal Display**.
10. Go to the toolbar of the 3D view to click on the *Limit selection to visible* icon and disable it; the icon appears only in **Edit Mode** and in all the viewport shading modes, except for **Wireframe** and **Bounding Box**, and has the appearance of a cube with the vertices selected:



The "Limit selection to visible" button on the 3D viewport toolbar and the cube geometry subdivided through the Subdivision Surface modifier

11. Press *3* on the numpad to go in the **Side** view:

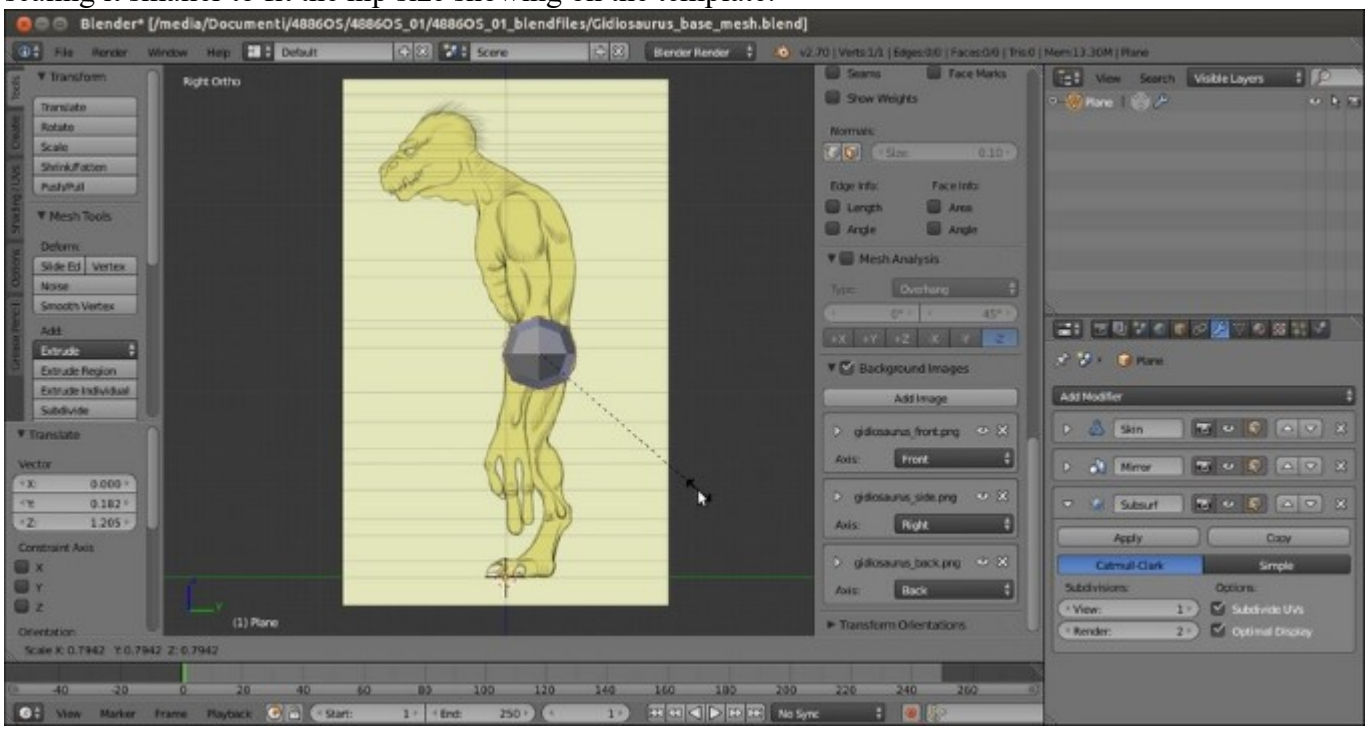


The created geometry and the side-view template reference

How to do it…

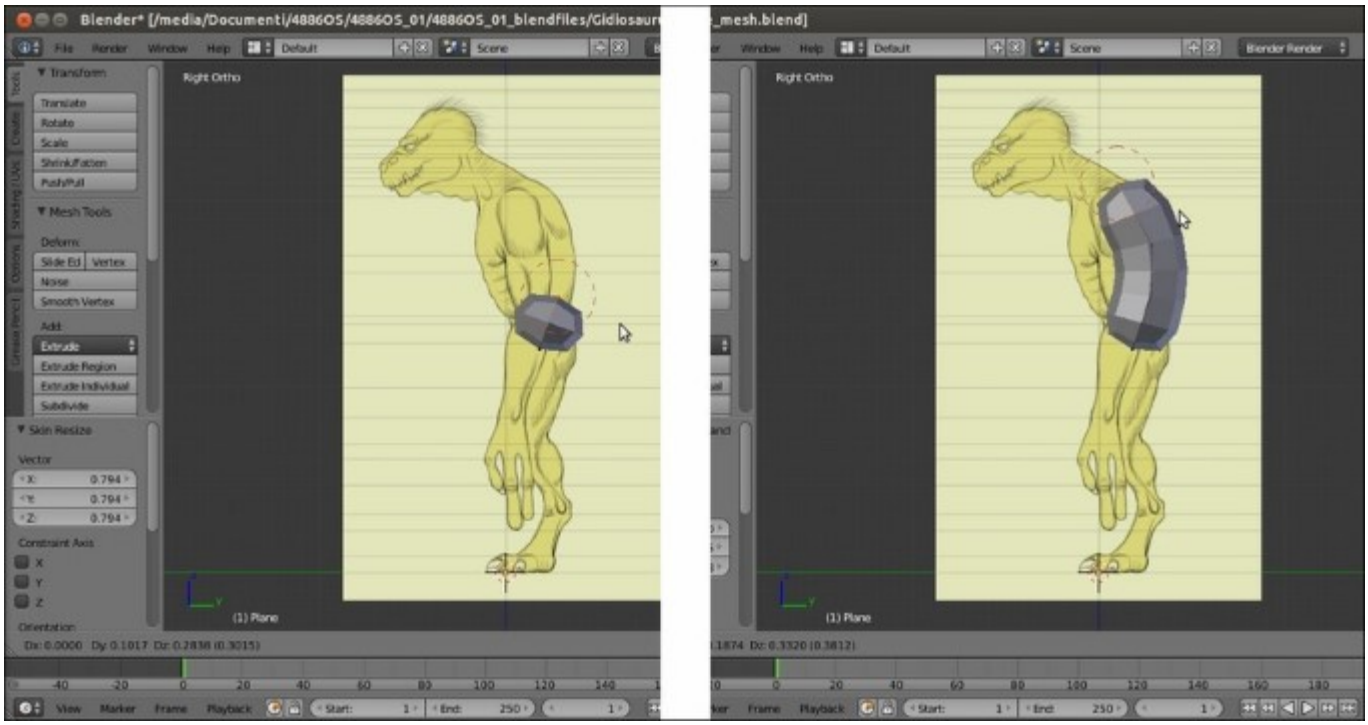
We are now going to move and extrude the vertex according to our template images, working as guides, and therefore generating a 3D geometry (thanks to the **Skin** modifier):

1. Press *G* and move the vertex to the pelvis area. Then, press *Ctrl* + *A* and move the mouse cursor towards the vertex to lower the weight/influence of the vertex itself on the generated mesh; scaling it smaller to fit the hip size showing on the template:



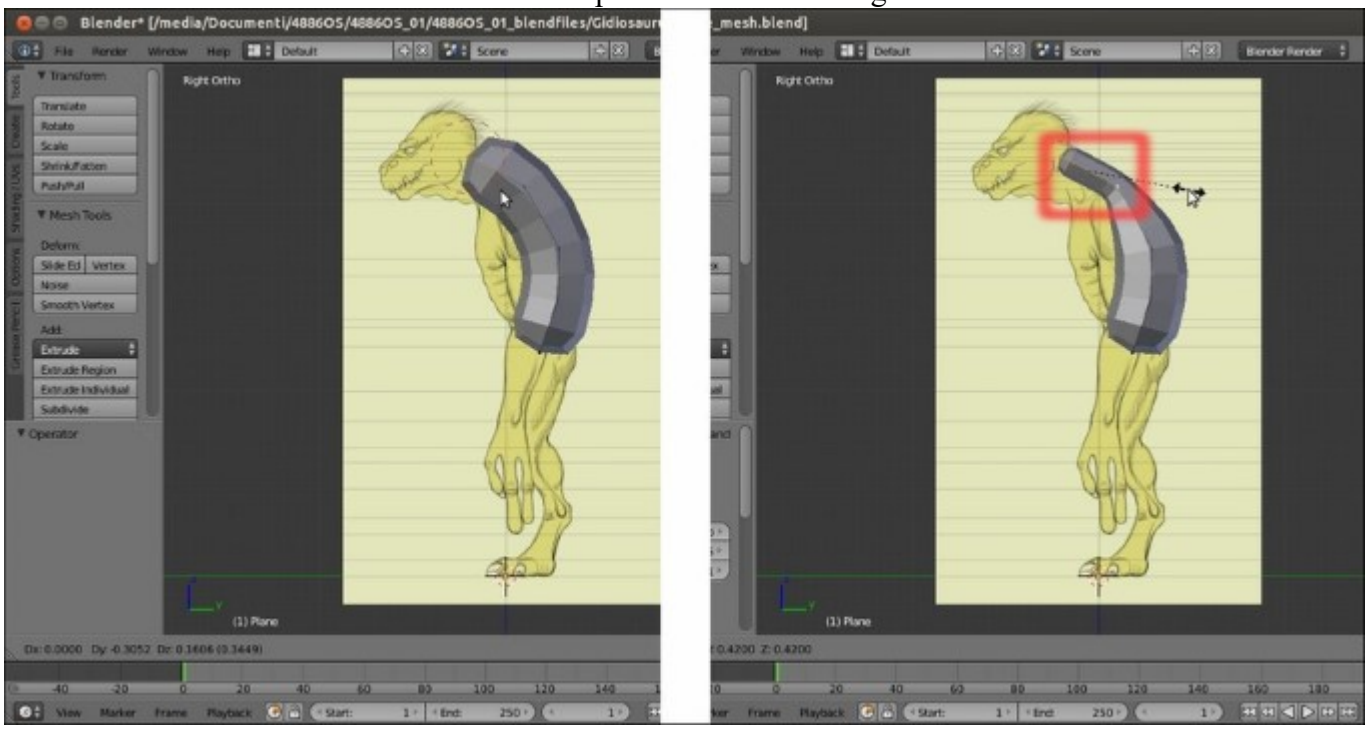
Moving the geometry to the character's pelvis area

2. Press *E* and extrude the vertex by moving it up on the *z* axis; place it at the bottom of the rib cage.
3. Go on extruding the vertex by following the lateral shape of the character in the template. Don't be worried about the volumes; for the moment, just build a *stick-figure* going up the torso:



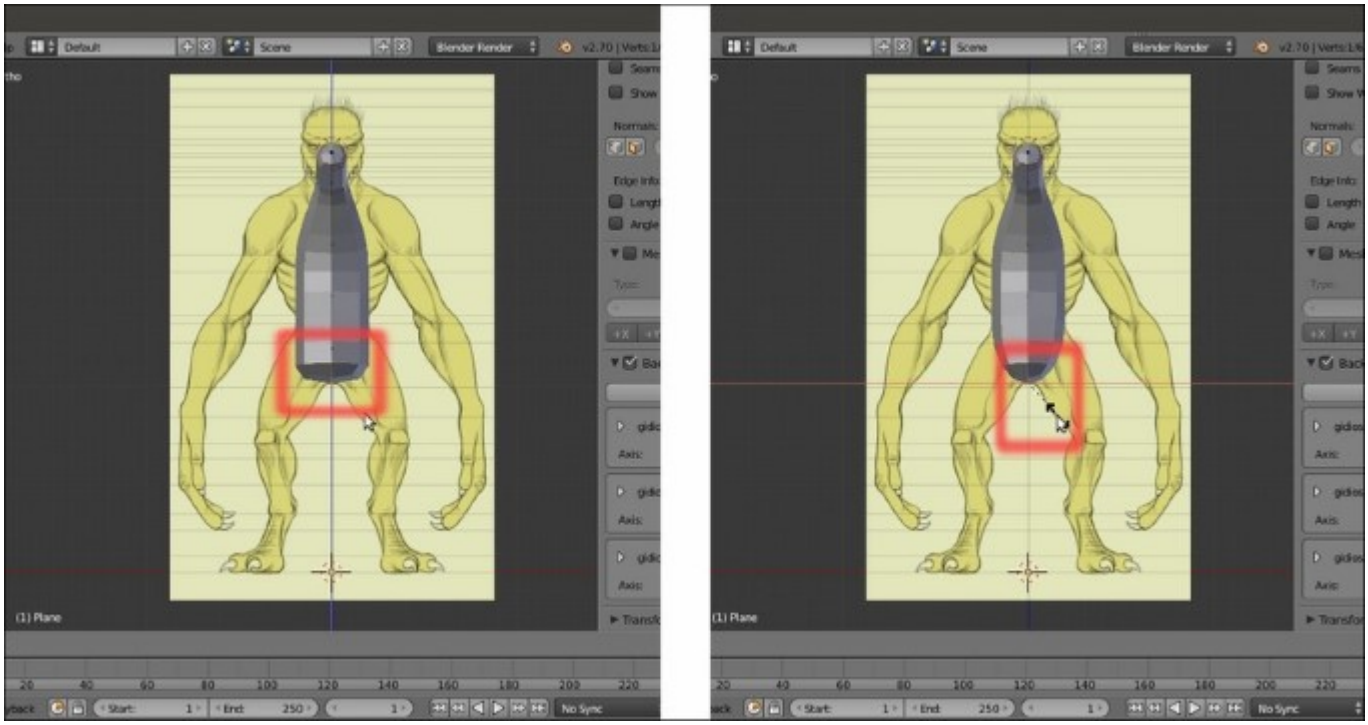
Extruding the vertices to create a new geometry

4. Proceed to the neck and stop at the attachment of the head location.
5. Select the last two vertices you extruded; press *Ctrl + A* and move the mouse cursor towards them to scale down their influence in order to provide a slim-looking neck:



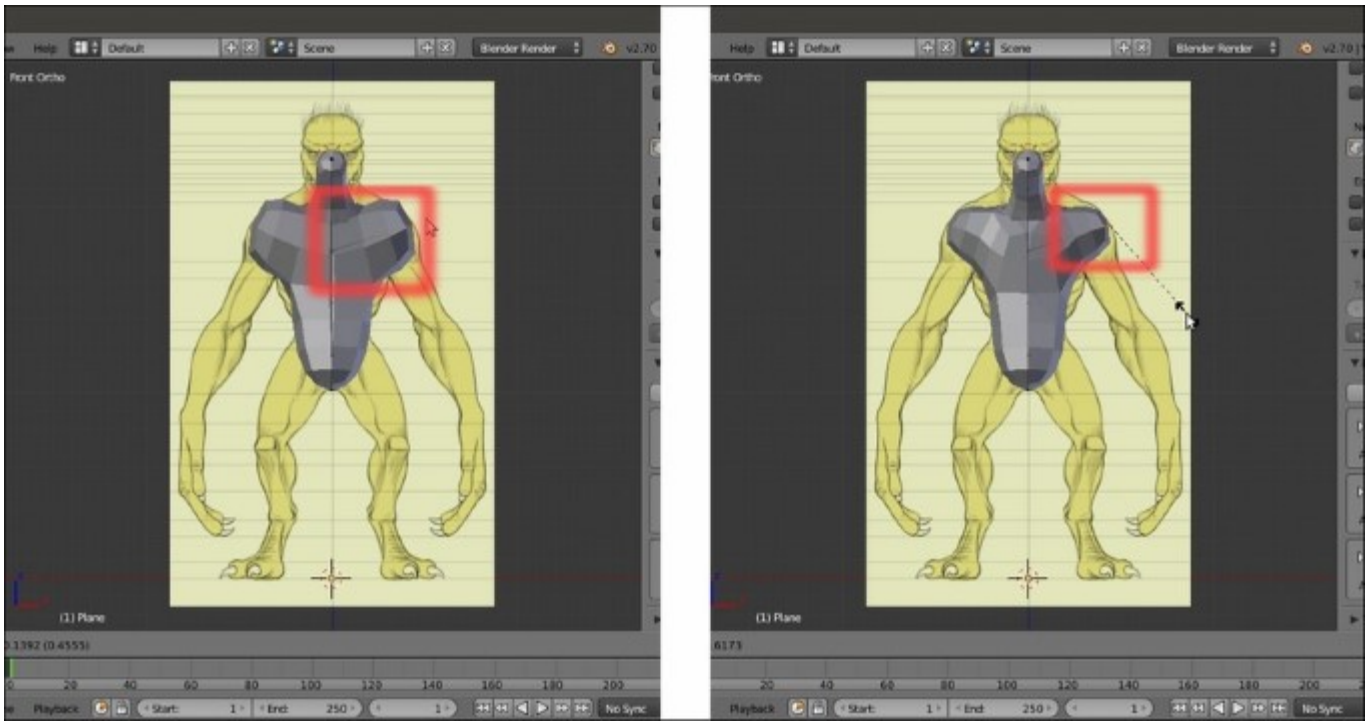
Scaling down the influence of the vertices

6. Press *1* on the numpad to switch to the **Front** view, and then select the bottom vertex and extrude it down to cover the base of the creature's pelvis. Press *Ctrl + A | X* to scale it only on the *x* axis:



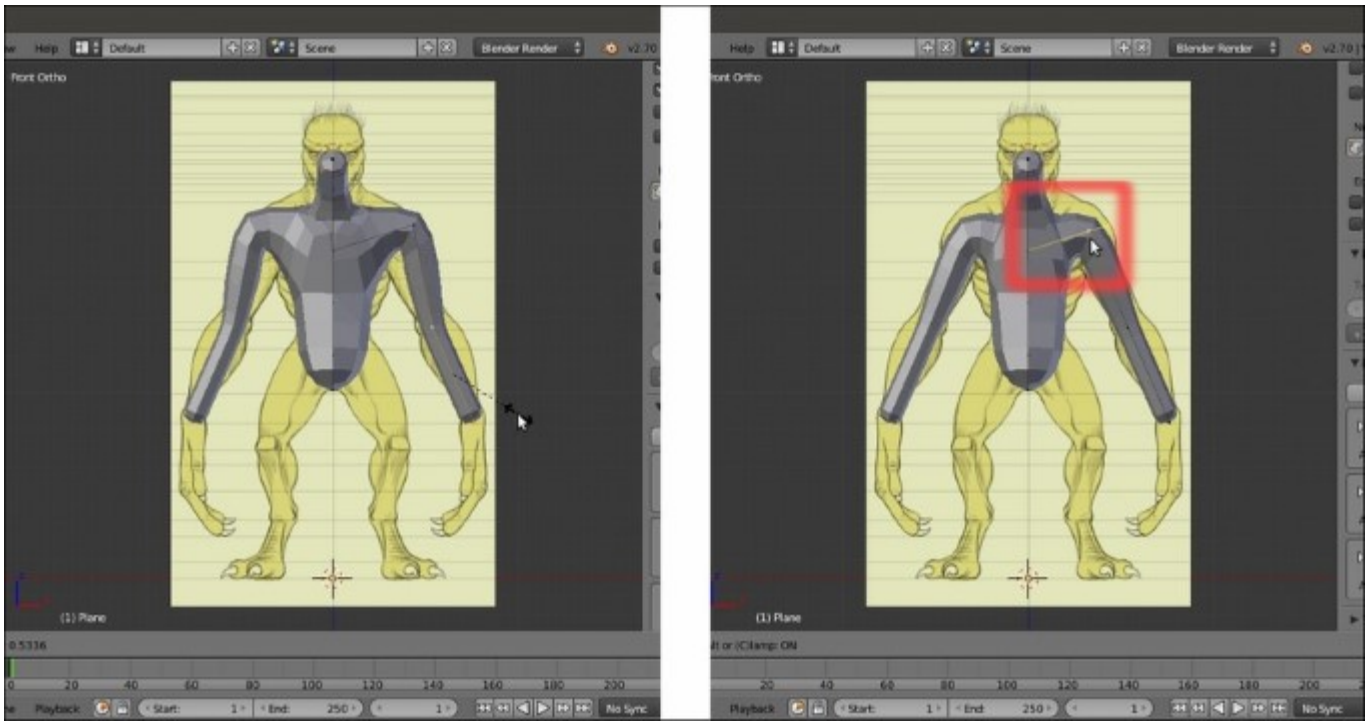
Adjusting the weight of the vertices in the Front view

7. Go to the **Mirror** modifier and uncheck the **Clipping** item.
8. Select the middle thorax vertex and extrude it to the right-hand side to build the shoulder. Press *Ctrl + A* to scale it smaller:



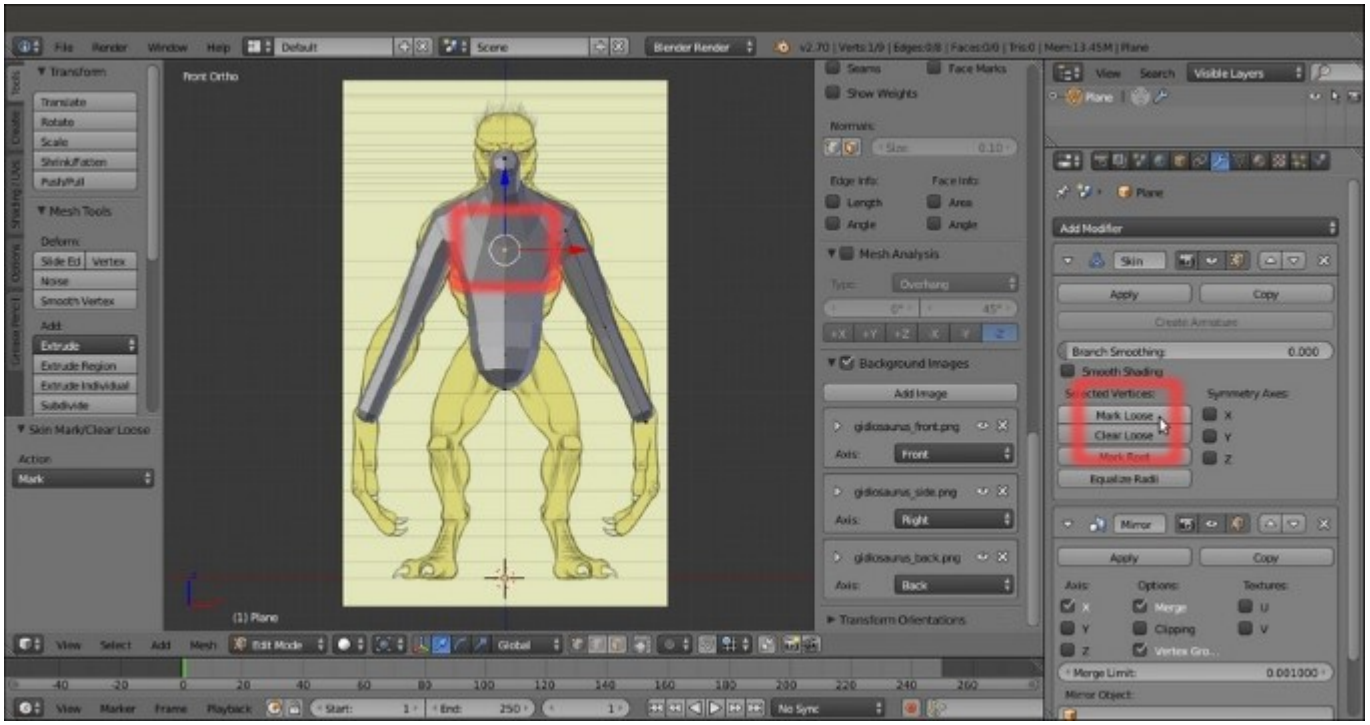
Creating the shoulders

9. Extrude the shoulder vertex, following the arm shape, and stop at the wrist; select the just-extruded arms' vertices and use *Ctrl + A* to scale them smaller.
10. Reselect the shoulder vertex, and use *Shift + V* to slide it along the shoulder's edge in order to adjust the location and fix the area shape:



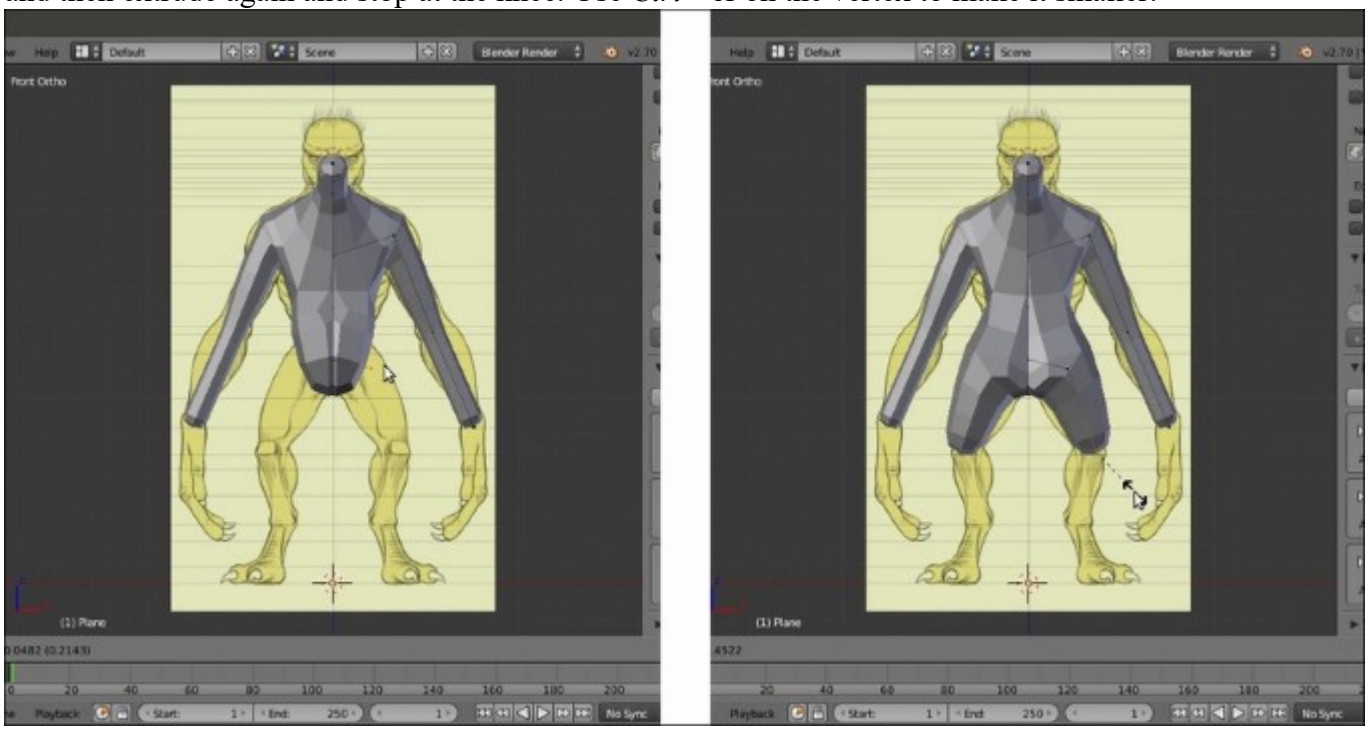
Creating the arms

11. Select the middle thorax vertex we extruded the shoulder from and go to the **Skin** modifier; click on the **Mark Loose** button:



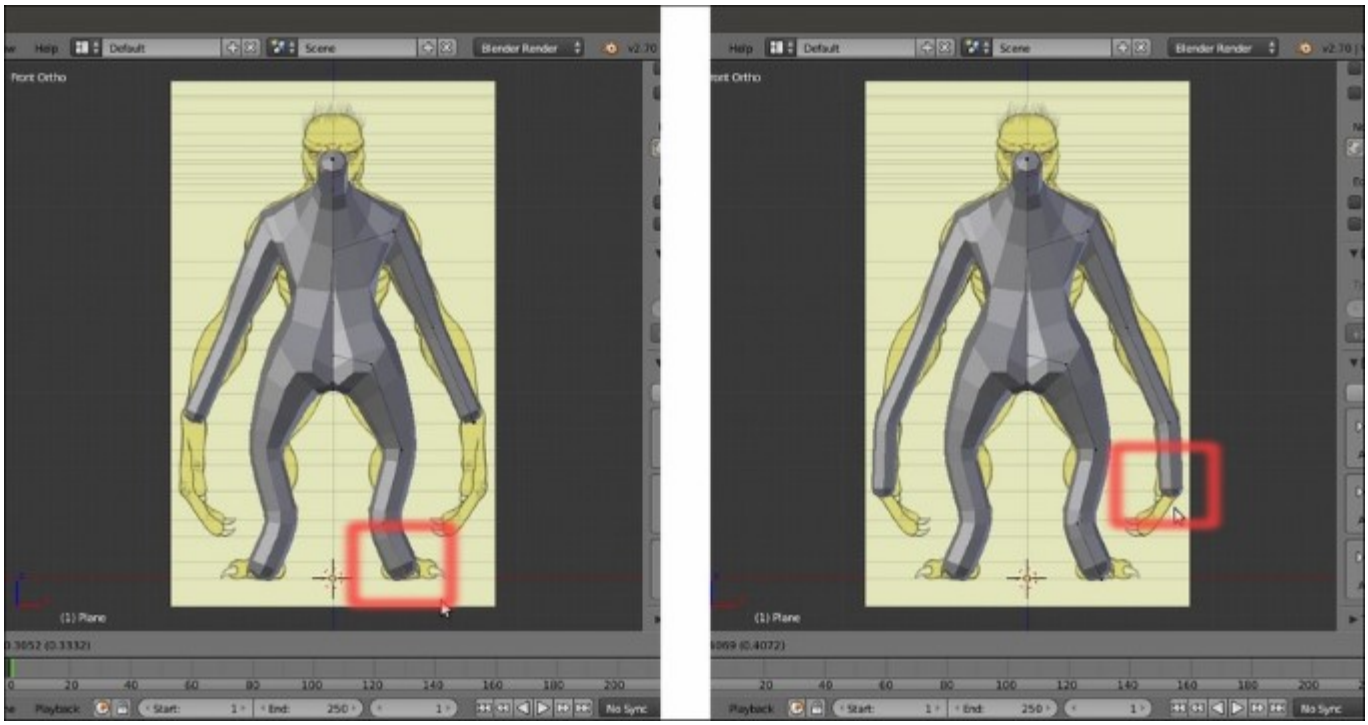
Making a more natural transition from the thorax to the arms

12. Select the second vertex from the bottom and extrude it to the right-hand side to build the hip, and then extrude again and stop at the knee. Use *Ctrl + A* on the vertex to make it smaller:



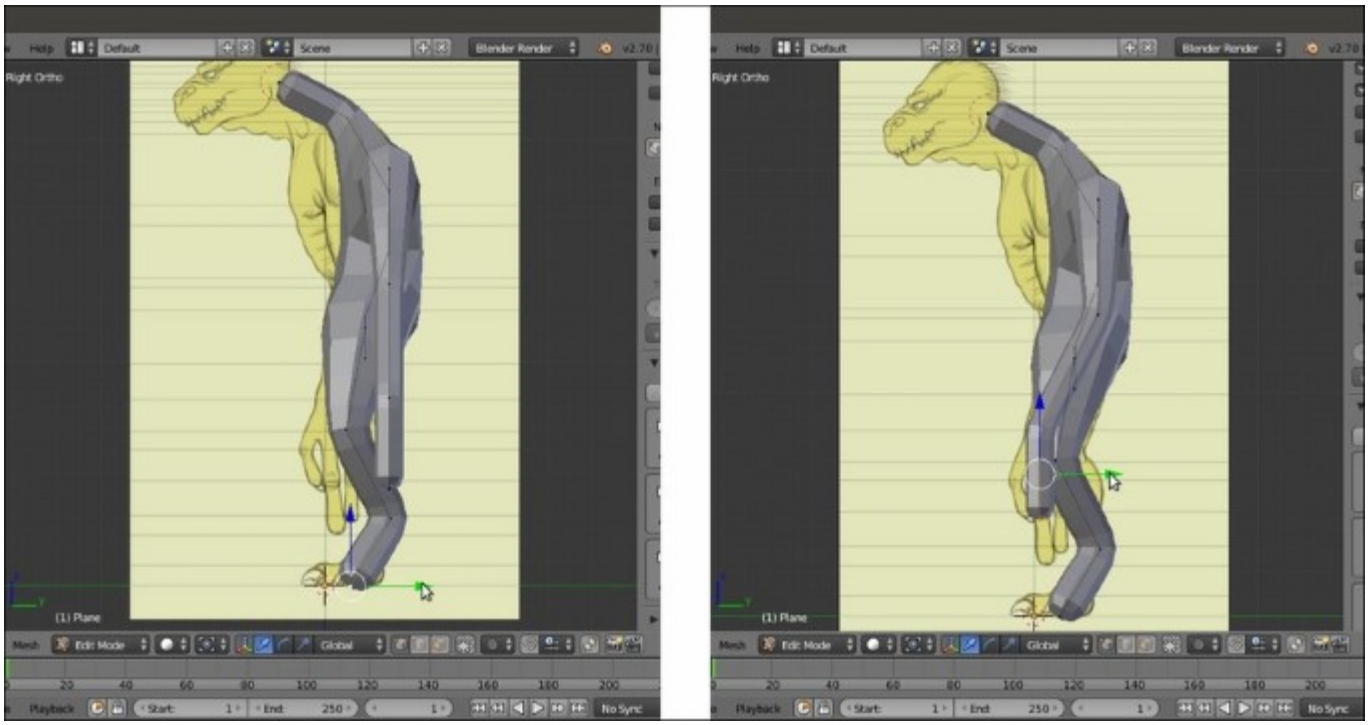
Extruding the thighs

13. Go on extruding the vertex to build the leg. Then, select the wrist vertex and extrude it to build the hand:



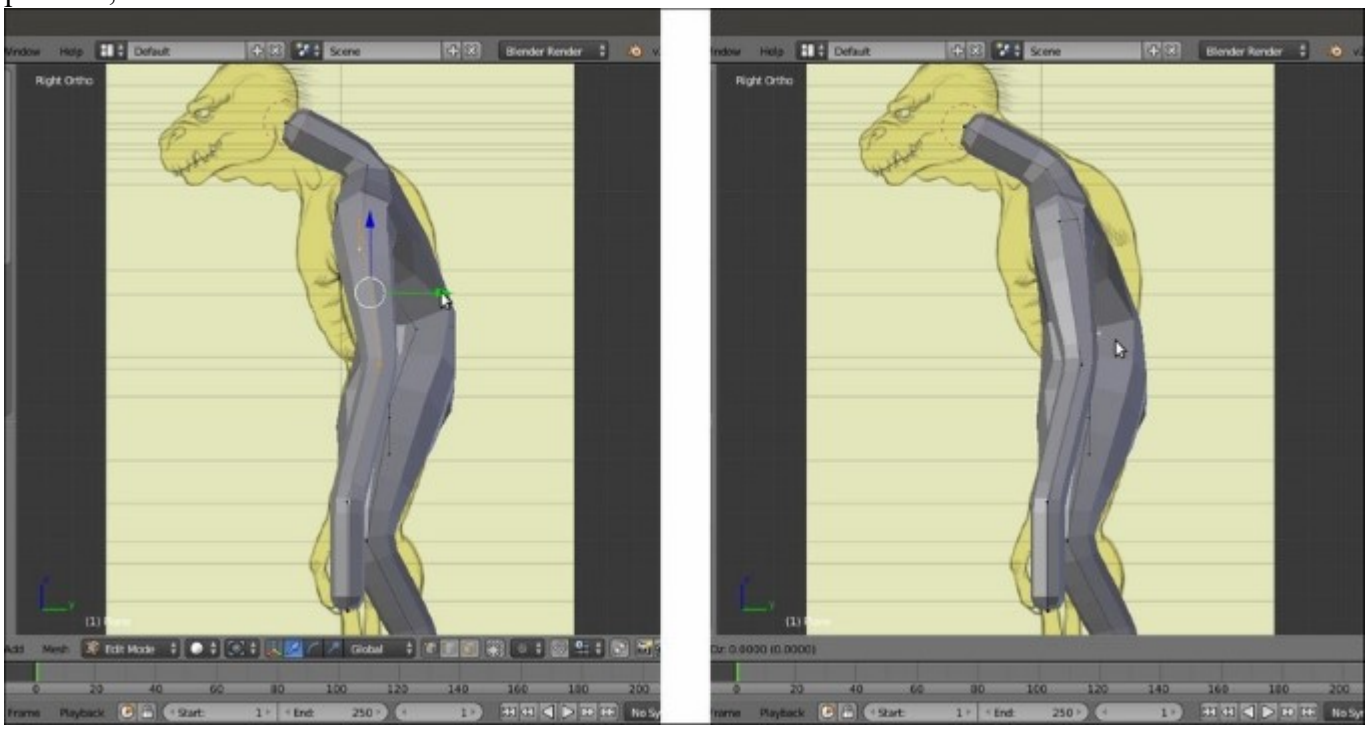
Extruding to complete the leg

14. Press *3* to go to the **Side** view.
15. Individually, select the vertices of the knee, ankle, and foot, and move them to be aligned with the character's posture (you can use the widget for this and, if needed, you can press *Z* to go in to **Wireframe** viewport shading mode); do the same with the vertices of the arm:



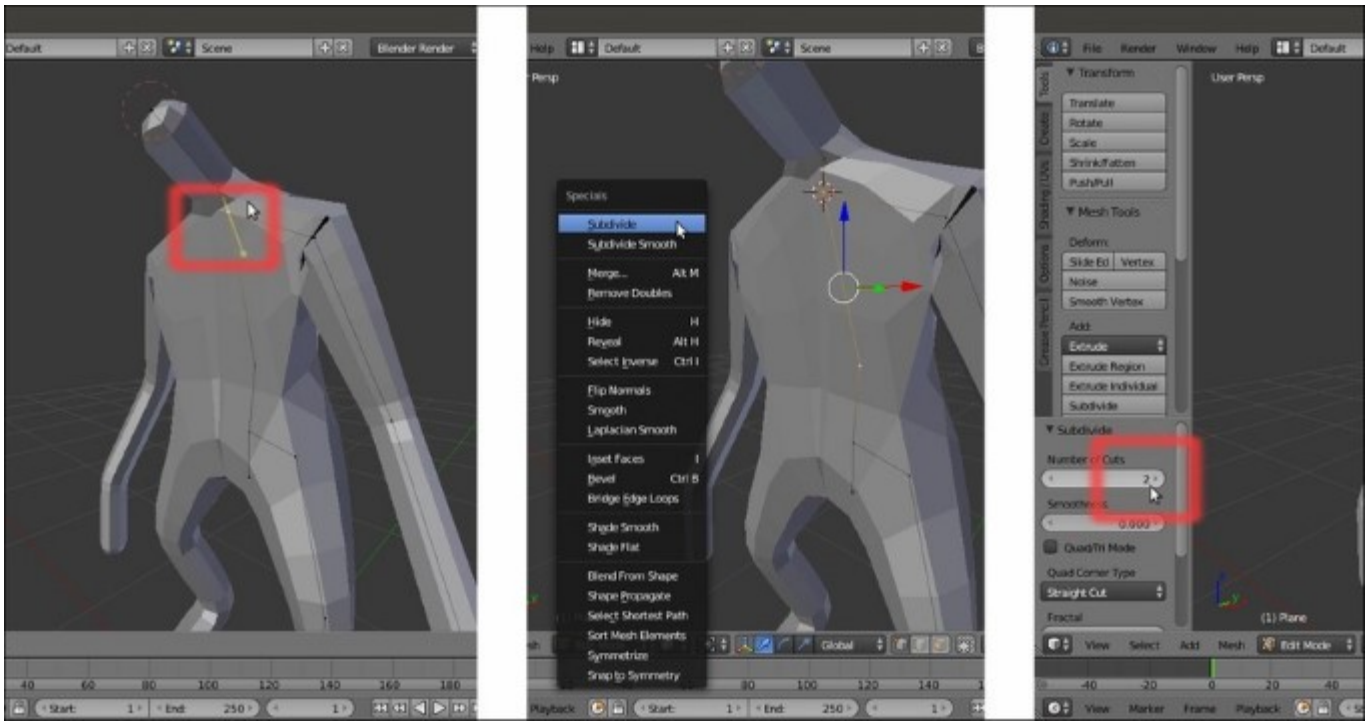
Adjusting the arm's position

16. Select the vertices of the shoulder and elbow, and move them forward according to the template position; do the same with the vertices of the neck and waist:



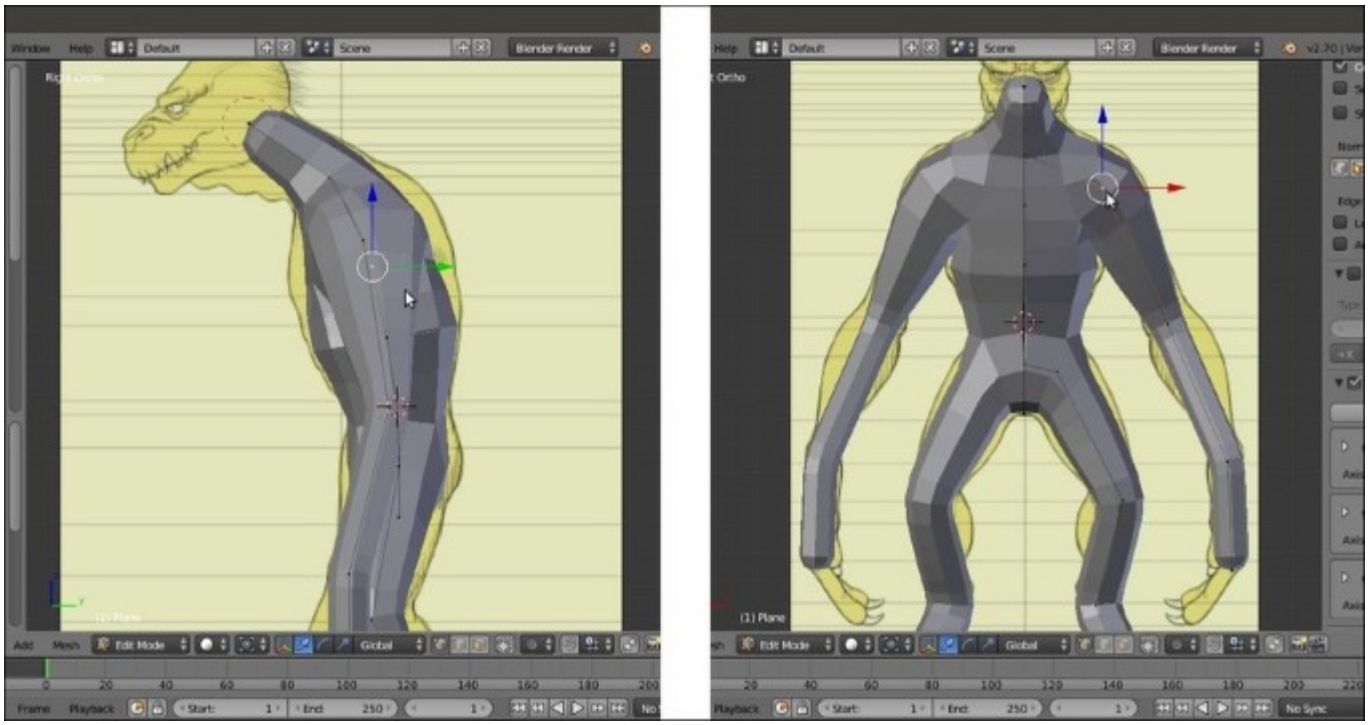
Adjusting the position of the shoulders, thorax, and neck

17. Select the vertex connecting the shoulder to the thorax and use *Shift + V* to slide it upwards, in order to make room for more vertices in the chest area. Use *Shift* to select the vertex at the bottom of the rib cage and press *W*; in the **Specials** pop-up menu, select **Subdivide** and, right after the subdivision, in the option panel at the bottom-left of the Blender UI, set **Number of Cuts** to **2**:



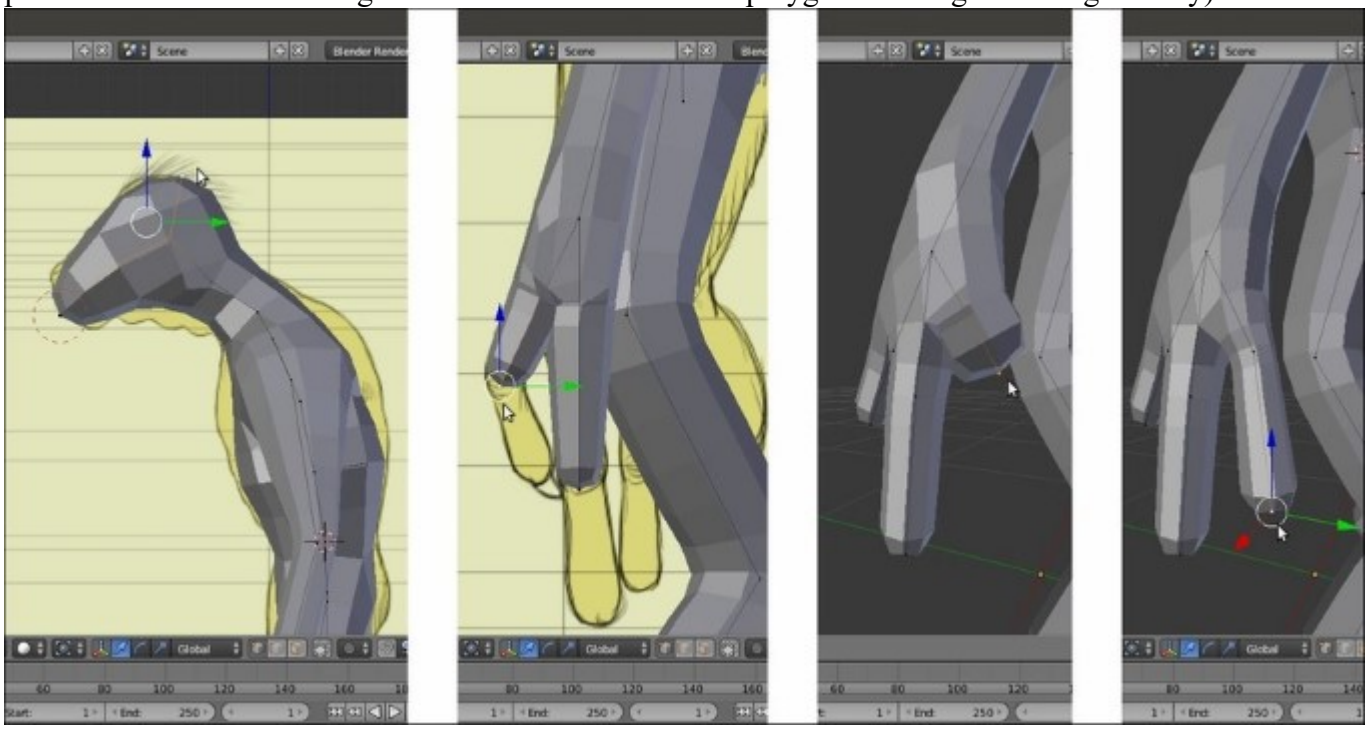
Subdividing an edge

18. In the **Side** view, select the upper one of the new vertices and use *Ctrl + A* to scale it bigger. Adjust the position and scale of the vertices around that area (neck and shoulder) to obtain, as much as possible, a shape that is more regular and similar to the template. However, don't worry too much about a perfect correspondence, it can be adjusted later:



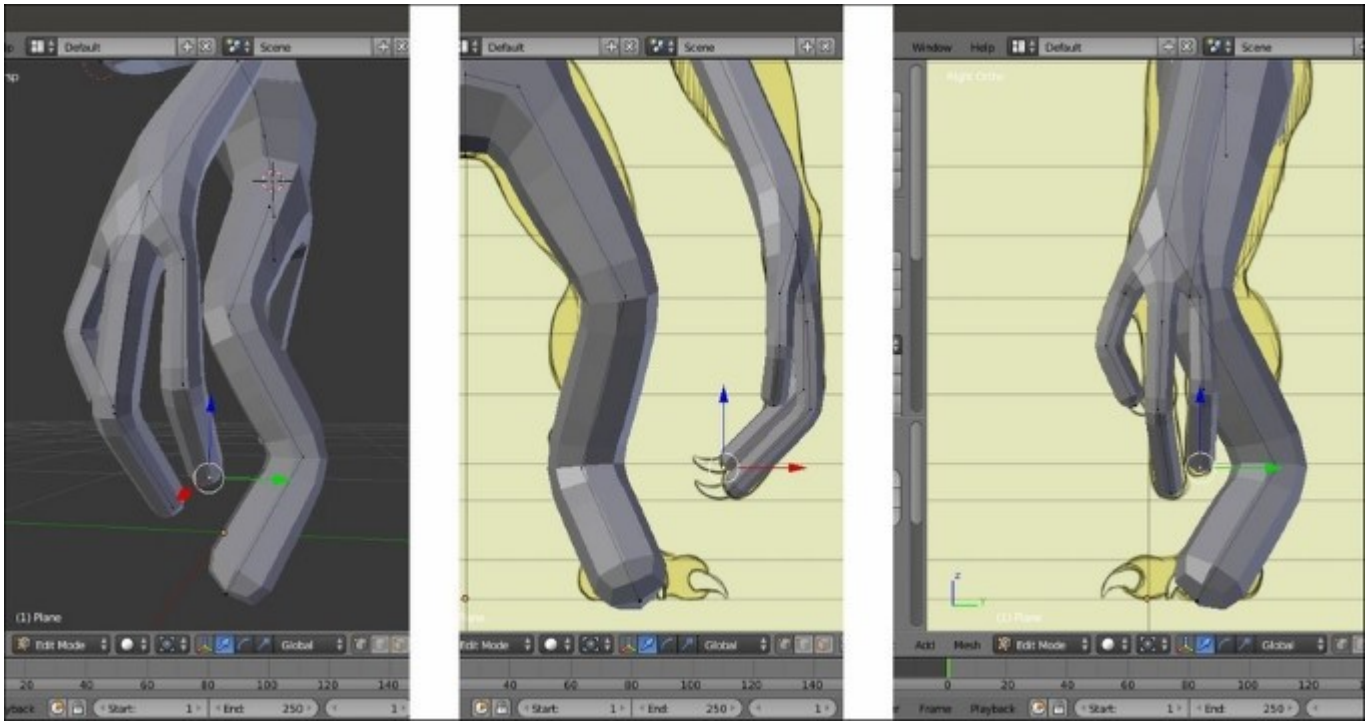
Refining the shoulder's shape

19. Extrude the bulk of the head. Select the last hand vertex and scale it smaller. Then, select the upper hand vertex and extrude two more fingers (scale their influence smaller and adjust their position to obtain a more regular and ordinate flow of the polygons in the generated geometry):



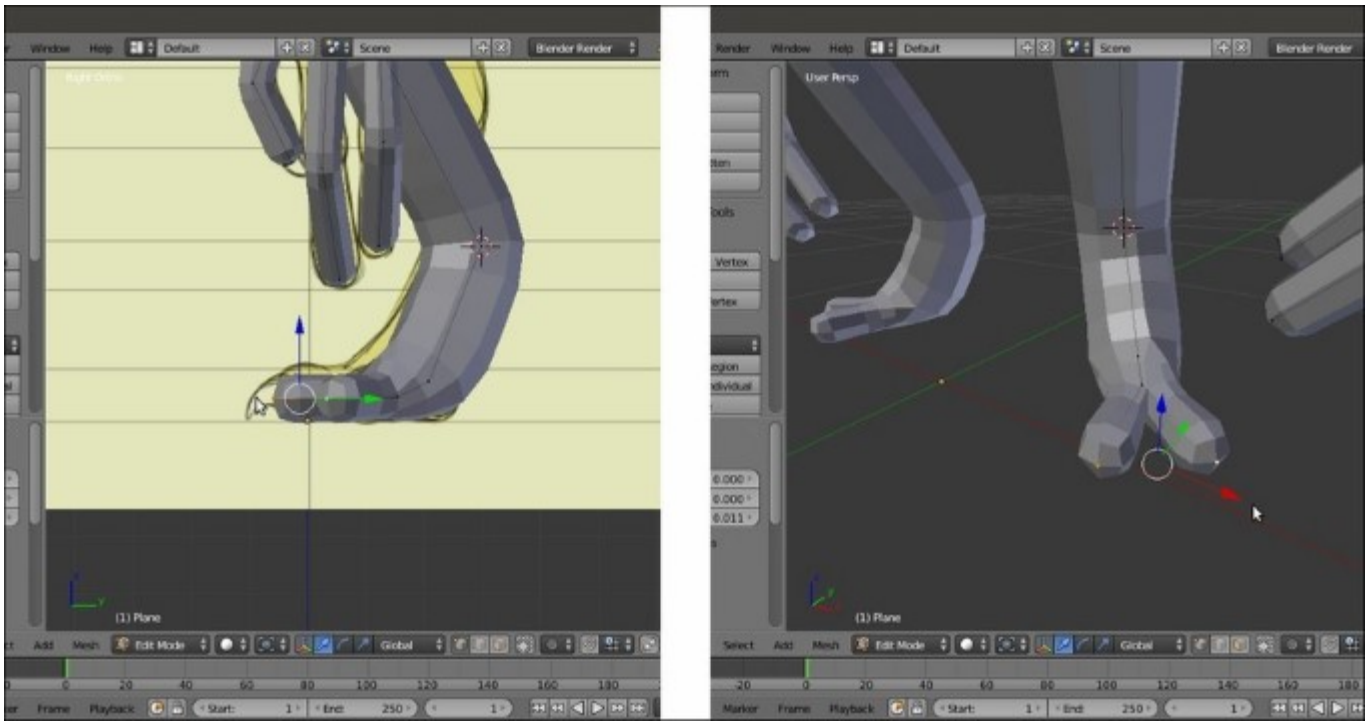
Creating the head, hands, and fingers

20. As always, following the templates as reference, extrude again to complete the fingers; use all the templates to check the accuracy of the proportions and positions, and the **Front**, **Side**, and **Back** views too:



Adjusting the position of the fingers according to the templates

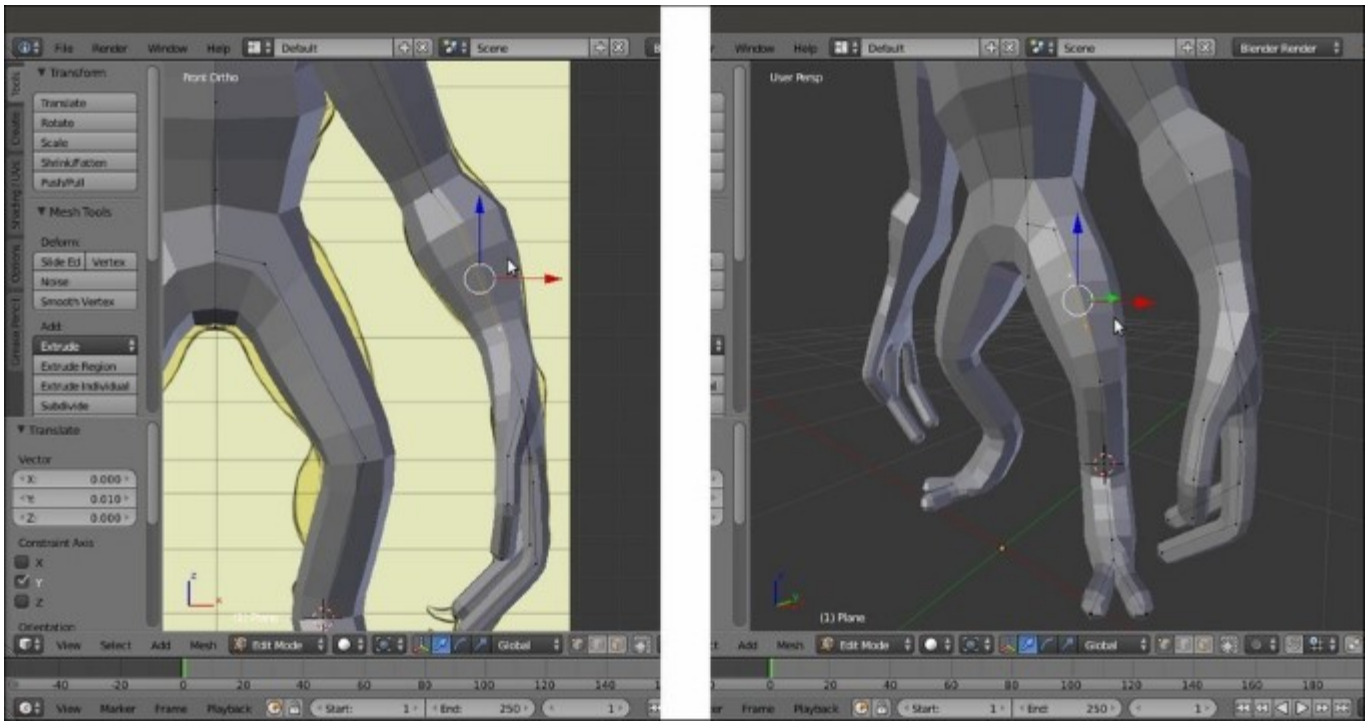
21. Do the same thing for the foot, and we are almost done with the major part of the mesh:



Creating the feet toes

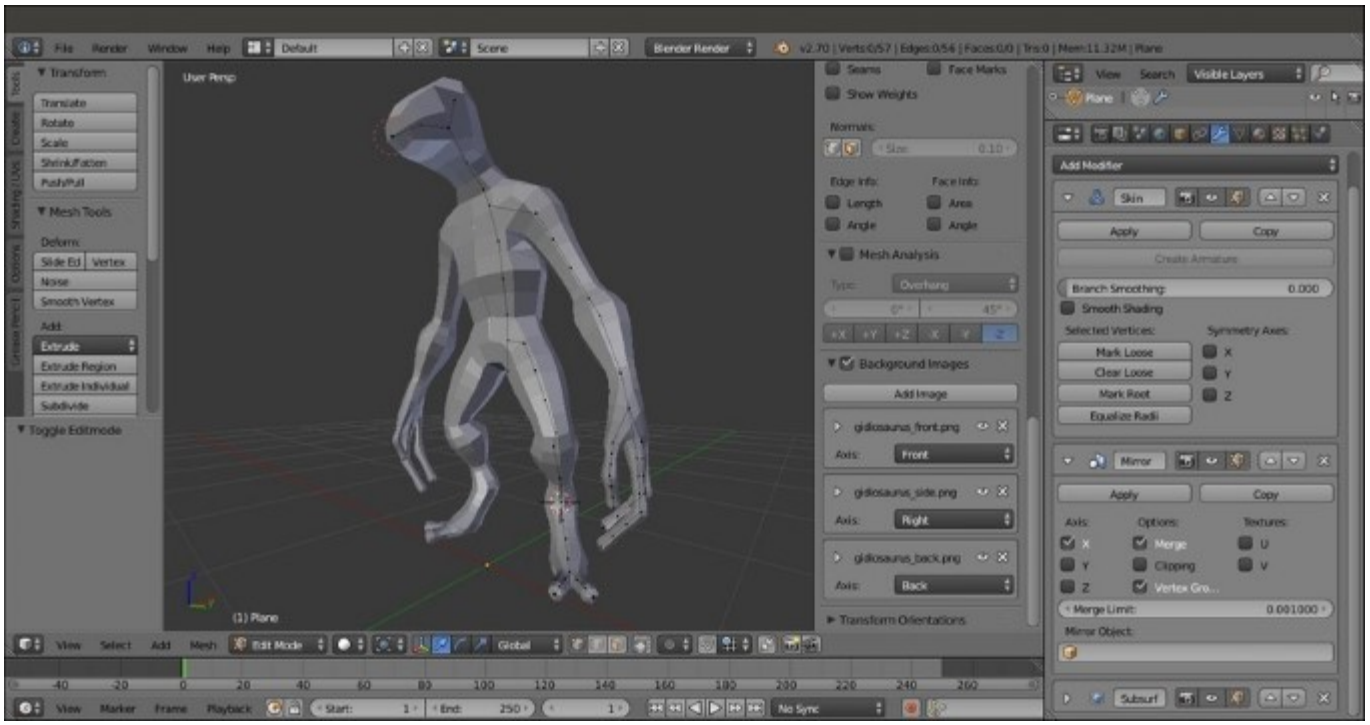
Now, it's only a matter of refining, as much as possible, the mesh's parts to resemble best the final shape of the character. Let's try with the arm first:

22. Select the two extreme vertices of the forearm and press *W* | **Subdivide** | **2** (in the bottom **Tool** panel) to add **2** vertices in the middle. Then, use *Ctrl + A* to scale and move them outward to curve the forearm a little bit. Do the same for the thigh by slightly moving the vertices outward and backward:



Refining the shape of arms and legs

23. Repeat the same procedure with the upper arm, shin, foot, and fingers; any part where it's possible, but don't go crazy about it. The goal of such a technique is just to quickly obtain a mesh that is good enough to be used as a starting point for the sculpting, and not an already finished model:



The completed base mesh

24. Press *Tab* to go out of the **Edit Mode**; go to the **Outliner** and rename the base mesh as `Gidiosaurus`. Then, save the file.

How it works…

The **Skin** modifier is a quick and simple way to build almost any shape; its use is very simple: first, you extrude vertices (actually, it would be enough to add vertices; it's not mandatory to extrude them, but certainly it's more handy than using *Ctrl* + left-click to add them at several locations), and then using the *Ctrl* + *A* shortcut, you scale smaller or bigger the influence that these vertices have on the 3D geometry generated on the fly.

If you have already tried it, you must have seen that the more the complexity of the mesh grows, the more the generated geometry starts to become a little unstable, often resulting in intersecting and overlapping faces. Sometimes this seems unavoidable, but in any case it is not a big issue and can be easily fixed through a little bit of editing. We'll see this in the next chapter.

Chapter 2. Sculpting the Character's Base Mesh

In this chapter, we will cover the following recipes:

- Using the Skin modifier's Armature option
- Editing the mesh
- Preparing the base mesh for sculpting
- Using the Multiresolution modifier and the Dynamic topology feature
- Sculpting the character's base mesh

Introduction

In the previous chapter, we built the base mesh by using the **Skin** modifier and on the base of the reference templates; in this chapter, we are going to prepare this basic mesh for the sculpting, by editing it and cleaning up any *mistakes* the **Skin** modifier may have made (usually, overlapping and triangular faces, missing edge loops, and so on).

Using the Skin modifier's Armature option

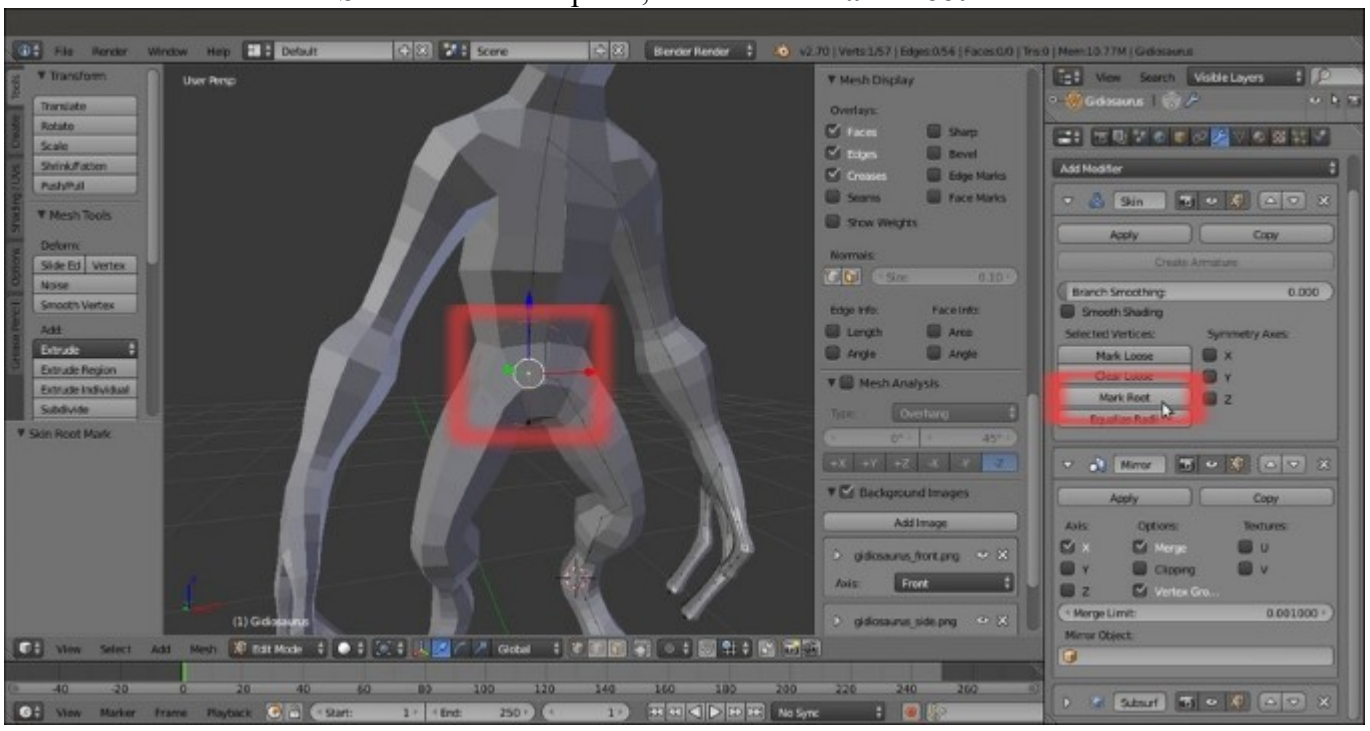
The **Skin** modifier has an option to create an **Armature** on the fly to pose the **generated mesh**. This **Armature** can just be useful in cases where you want to modify the position of a part of the generated mesh.

Note that using the generated **Armature** to pose the base mesh, in our case, is not necessary, and therefore this recipe is treated here only as *an example* and it won't affect the following recipes in the chapter.

Getting ready

So, let's suppose that we want the **arms** to be posed more horizontally and widely spread:

1. If this is the case, reopen the `Gidiosaurus_base_mesh.blend` file and save it with a different name (something like `Gidiosaurus_Skin_Armature.blend`).
2. Select the **Gidiosaurus** mesh and press *Tab* to go into **Edit Mode**; then, select the central pelvis vertex.
3. Go to the **Object Modifiers** window under the main **Properties** panel to the right-hand side of the screen and then to the **Skin** modifier subpanel; click on the **Mark Root** button:



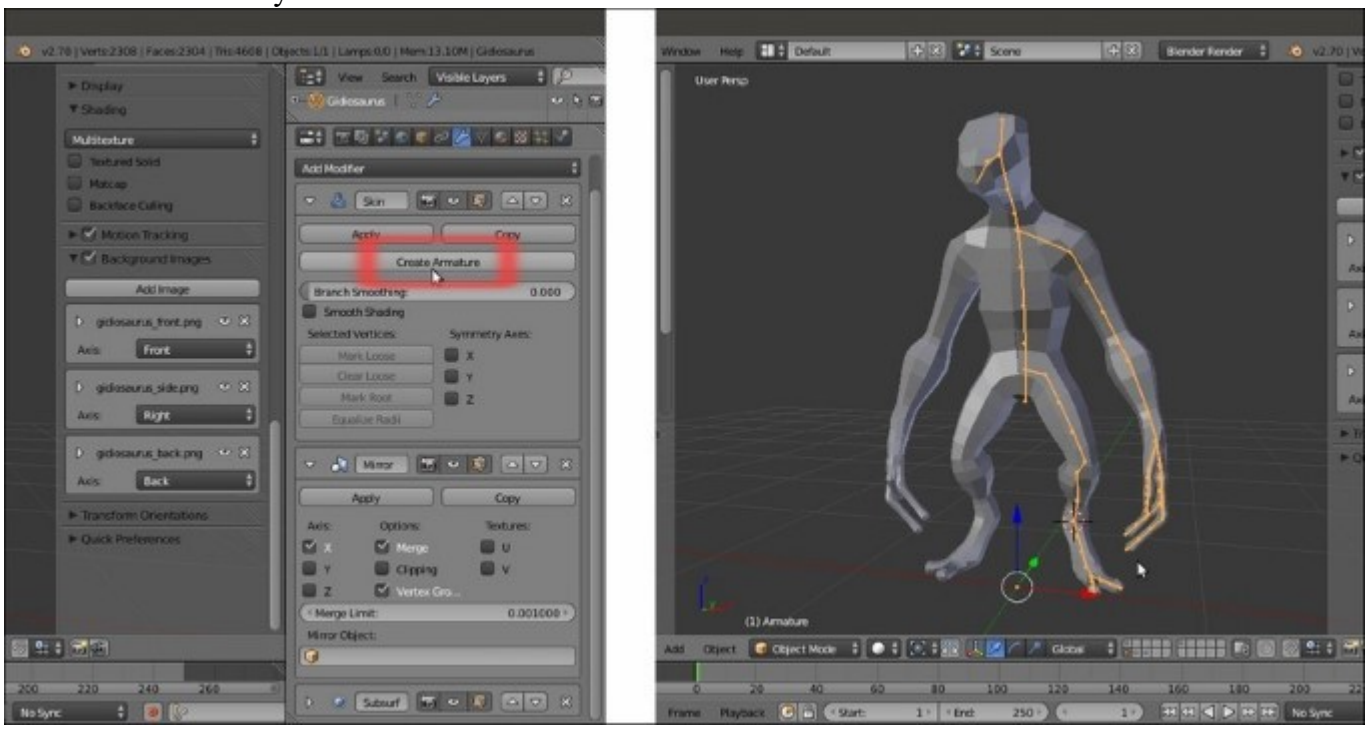
The root vertex

4. Press *Tab* again to exit **Edit Mode**.

How to do it…

Creating the rig (that is the skeleton **Armature** made by bones and used to deform, and therefore, animate a mesh) for our character's base mesh is really simple:

1. Again, in the **Skin** modifier subpanel, click on the **Create Armature** button. The **Armature** is created instantly and an **Armature** modifier is automatically assigned to the mesh; in the modifier stack, move it to the top so that it is above the **Mirror** modifier and our posed half-mesh will be correctly mirrored:



The Armature created by the Skin modifier

2. Press *Ctrl* + *Tab* to enter **Pose Mode** for the already selected **Armature** and then select the upper bone of the **arm**.
3. In the toolbar of the 3D viewport, find the widget manipulators panel, click on the rotation **Transformation manipulators** (the third icon from the left), and set **Transform Orientation** to **Normal**.
4. By using the rotate widget, rotate the selected bone and consequently the arm (be careful that, as already mentioned, the newly created **Armature** modifier is at the top of the modifier stack, otherwise the rotation will not correctly deform the mirrored mesh):

Rotating the arms through the Armature

5. Exit **Pose Mode** and reselect the **Gidiosaurus** mesh.
6. Go to the **Skin** modifier subpanel under the **Object Modifiers** window; click on the **Apply** button to apply the modifier.
7. Go to the **Armature** modifier and click on the **Apply** button to also apply the rig transformations.
8. At this point, we can also select the **Armature** object and delete it (*X* key).

How it works…

By clicking on the **Create Armature** button, the **Skin** modifier creates a bone for each edge connecting the extruded vertices, it adds an **Armature** modifier to the generated base mesh, and automatically assigns vertex groups to the base mesh and skins them with the corresponding bones.

The bones of this **Armature** work in **Forward Kinematics**, which means they are chained following the **child/parent** relation, with the first (**parent**) bone created at the **Root** location we had set at step 3 of the *Getting ready* section.

There's more…

Note that the bones of the **Armature** can be used not only to rotate limbs, but also to scale bigger or smaller parts of the mesh, in order to further tweak the shape of the base mesh.

See also

- http://www.blender.org/manual/modifiers/generate/skin.html
- http://www.blender.org/manual/rigging/posing/editing.html#effects-of-bones-relationships

Editing the mesh

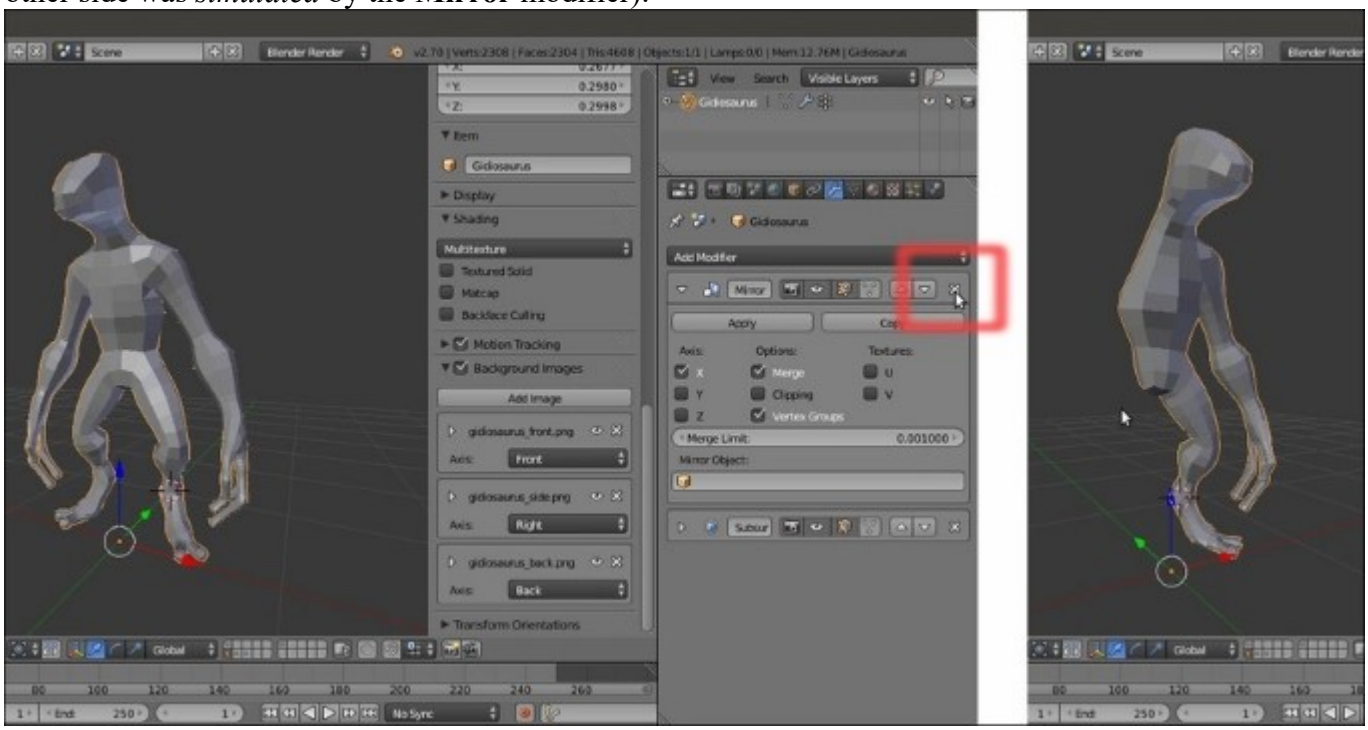
Once we have applied the **Skin** and **Armature** modifiers, we are left with an almost ready-to-use base mesh; what we need to do now is clean the possibly overlapping faces and whatever other mistakes were made by the **Skin** modifier.

Be careful not to be confused by the previous recipe, which was meant only as a possible example; we didn't actually use the **Skin** modifier's **Armature** to change the pose of the base mesh.

Getting ready

Let's prepare the mesh and the view:

1. Go to the **Object Modifiers** window under the main **Properties** panel and then to the **Mirror** modifier subpanel and click on the little **X** icon to the right in order to delete the modifier; you are left with half of the mesh (actually the half that is really generated by the **Skin** modifier; the other side was *simulated* by the **Mirror** modifier):

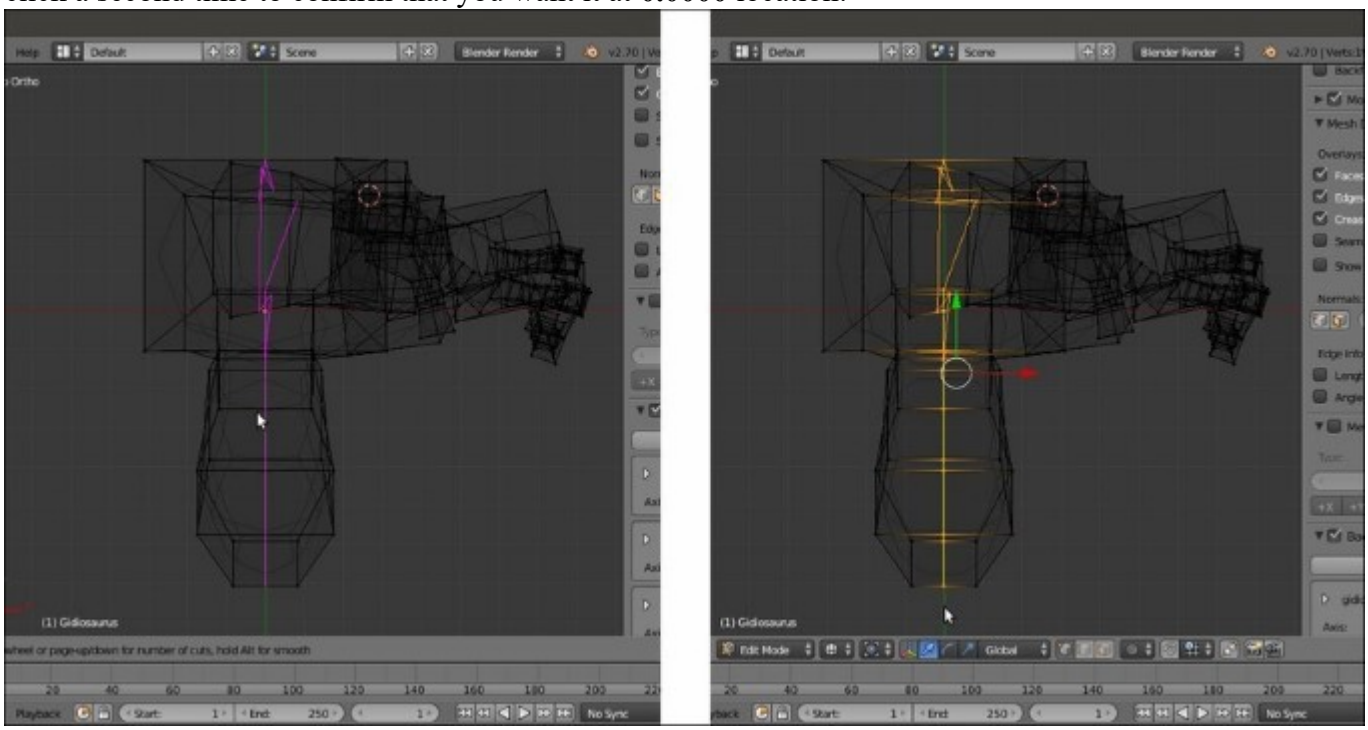


Deleting the Mirror modifier

2. Press *Tab* to go into **Edit Mode**, *7* on the numpad to go into **Top** view, and *Z* to go into the **Wireframe** viewport shading mode.

How to do it…

1. Press *Ctrl* + *R* to add an edge-loop to the middle of the mesh; don't move the mouse, and left-click a second time to confirm that you want it at **0.0000** location:

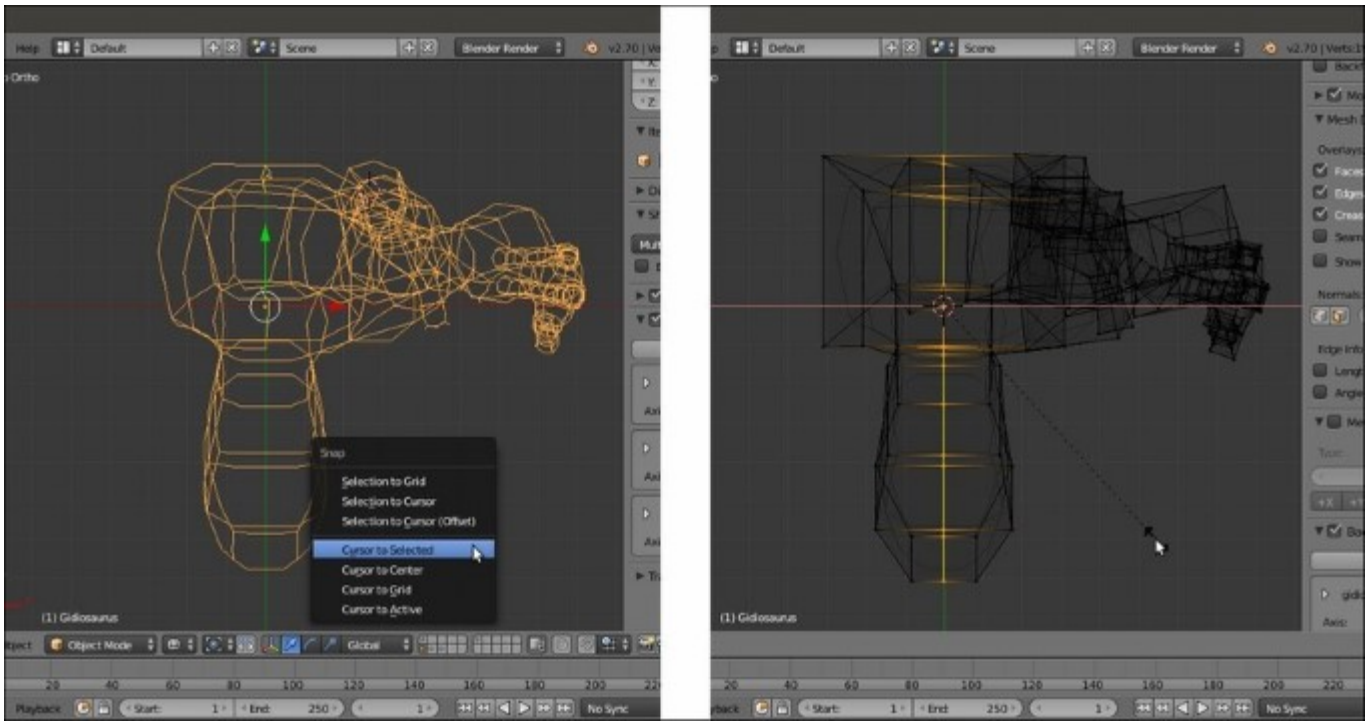


Adding a central edge-loop

Sometimes, depending on the topology created by the **Skin** modifier, you may not be able to make a single clean loop cut by the *Ctrl* + *R* key shortcut. In this case, still in **Edit Mode**, you can press the *K* key to call the **Knife Tool**, left-click on the mesh to place the cuts, and press *Enter* to confirm (press *Shift* + *K* if you want only the newly created edge-loops selected after pressing *Enter*). This way, you can create several loop cuts, connect them together and, if necessary, move and/or scale them to the middle along the *x* axis.

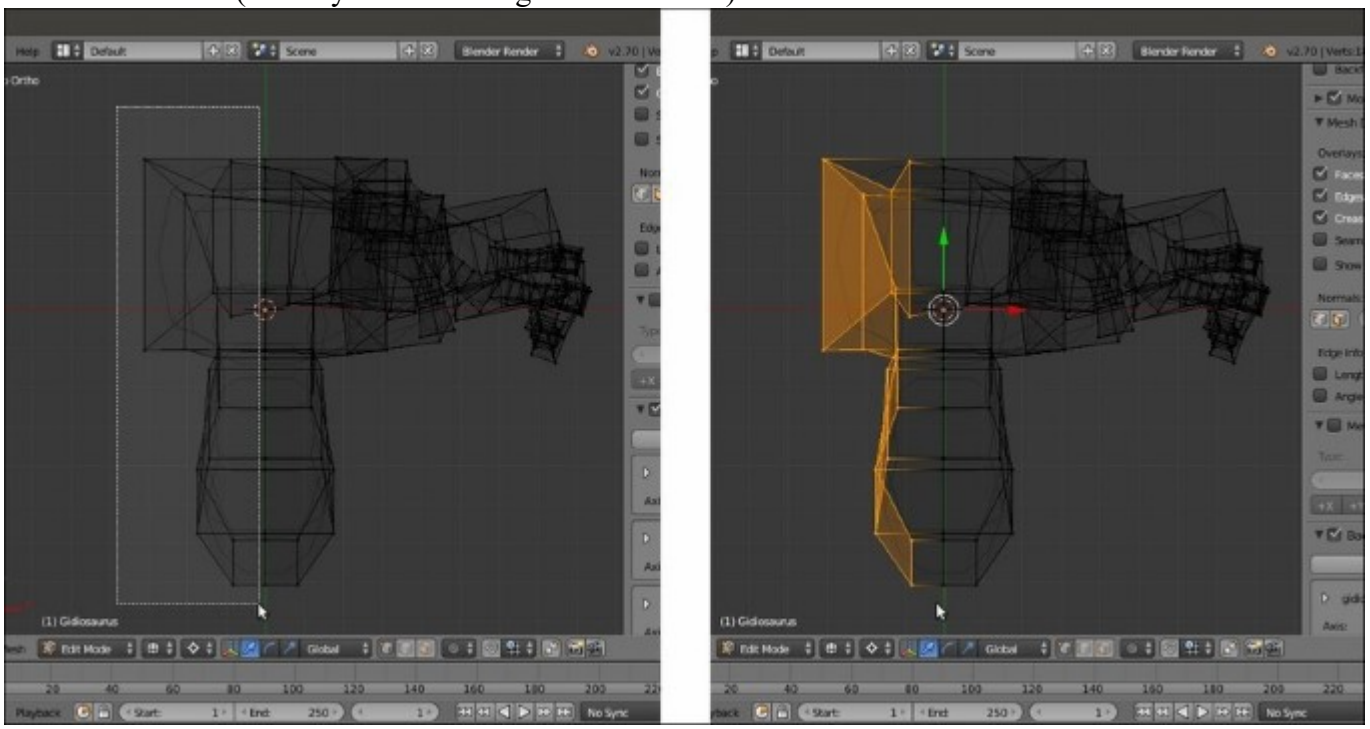
In fact, you can do the following:

2. Go out of **Edit Mode** and press *Shift* + *S*; in the **Snap** pop-up menu, select **Cursor to Selected** (to center the cursor at the middle of the mesh).
3. Press the period (.) key to switch **Pivot Point** to **3D Cursor** and then press *Tab* to go again into **Edit Mode**.
4. With the middle edge-loop already selected, press *S* | *X* | **0** | *Enter* to scale all its vertices to the **3D Cursor** position along the *x* axis and align them at the perfect center:



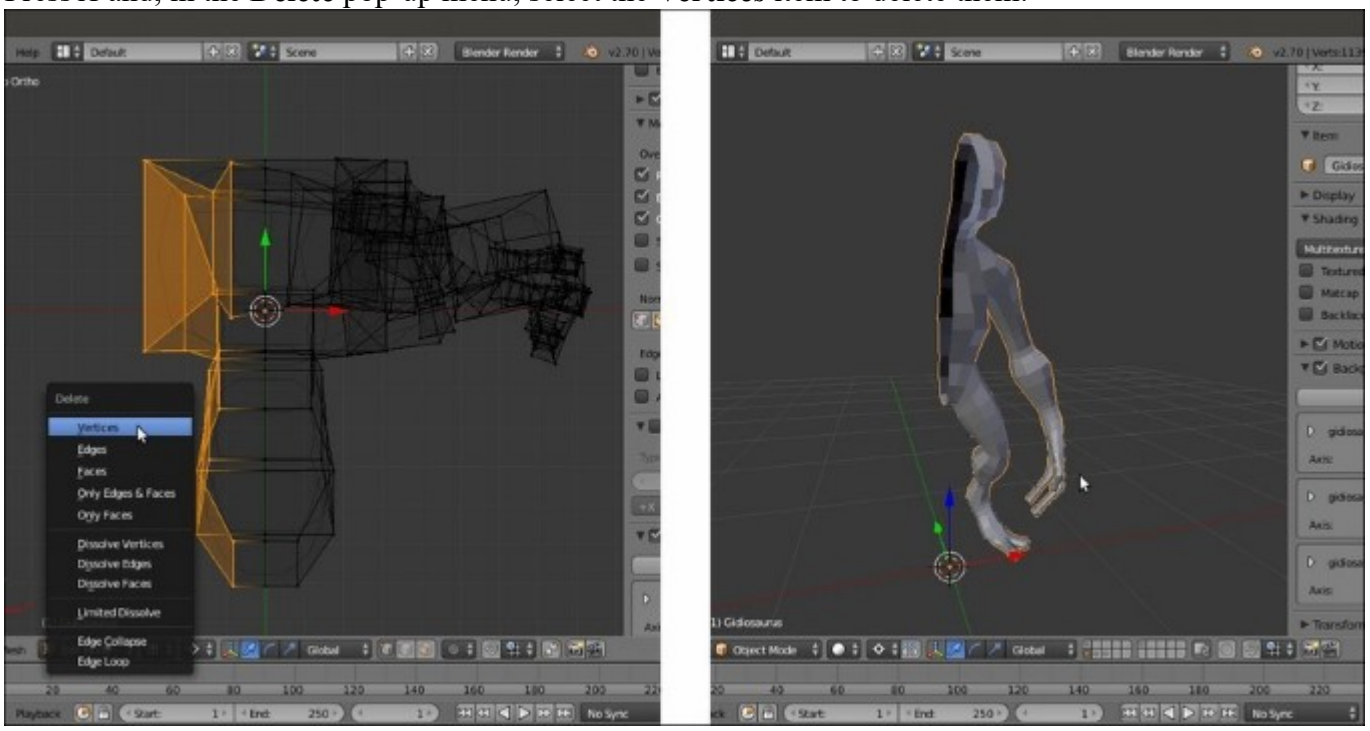
Scaling the central edge-loop vertices along the x axis

5. Press *A* to deselect all the vertices and then press *B* and box-select the vertices on the left-hand side of the screen (actually the mesh's right-side vertices):



Box-selecting the left vertices

6. Press *X* and, in the **Delete** pop-up menu, select the **Vertices** item to delete them:



Deleting the left vertices

7. Go out of **Edit Mode** and, in the **Object Modifiers** window, assign a new **Mirror** modifier (check **Clipping**) to the mesh; move it before the **Subdivision Surface** modifier in the stack.
8. If needed, this is the point where you can manually edit the mesh by converting triangle faces to quads (select two consecutive triangular faces and press *Alt* + *J*), creating, closing, or moving edge-loops (by using the **Knife Tool**, for example, around the arms and legs attachments to the body), and so on.
9. Save the file as `Gidiosaurus_base_mesh.blend`.

Well, in our case, everything went right with the **Skin** modifier, so there is no need for any big editing of the mesh! In effect, it was enough to delete the first **Mirror** modifier (that we actually used mostly for visual feedback) to get rid of all the overlapping faces and obtain a clean base mesh:

The "clean" mesh with new Mirror and Subdivision Surface modifiers

In the preceding screenshot, the base mesh geometry is showing with a level **1** of subdivision; in **Edit Mode**, it is still possible to see the low-level cage (that is, the *real* geometry of the mesh) as wireframe.

There are a couple of triangular faces (that, if possible, we should always try to avoid; quads faces work better for the sculpting) near the shoulders and on the feet, but we'll fix these automatically later, because before we start with the sculpting process, we will also apply the **Subdivision Surface** modifier.

How it works…

To obtain a clean half-body mesh, we had to delete the first **Mirror** modifier and the vertices of the right half of the mesh; to do this, we had also added a middle edge loop. So, we obtained a perfect left-half mesh and therefore we assigned again a **Mirror** modifier to restore the missing half of the body.

Preparing the base mesh for sculpting

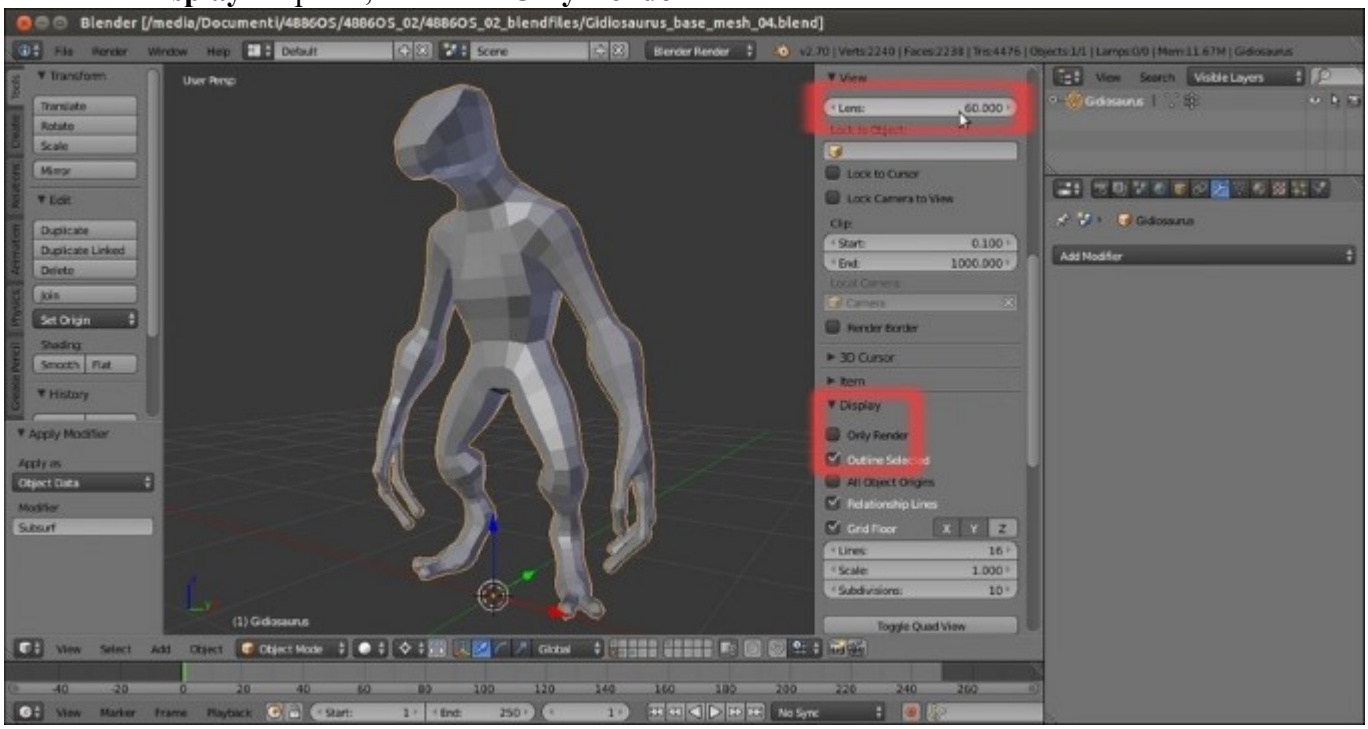
Once we have our base mesh completed, it's time to prepare it for the sculpting.

Getting ready

Open the `Gidiosaurus_base_mesh.blend` file and be sure to be out of **Edit Mode**, and therefore in **Object Mode**.

How to do it…

1. Select the character's mesh and go to the **Object Modifiers** window under the main **Properties** panel to the right.
2. Go to the **Mirror** modifier panel and click on the **Apply** button.
3. If this is the case, expand the **Subdivision Surface** modifier panel, be sure that the **View** level is at **1**, and click on the **Apply** button.
4. Press *Tab* to go into **Edit Mode** and, if necessary, select all the vertices by pressing *A*; then, press *Ctrl* + *N* to recalculate the normals and exit **Edit Mode**.
5. Go to the **Properties** sidepanel on the right-hand side of the 3D view (or press the *N* key to make it appear) and under the **View** subpanel, change the **Lens** angle to **60.000** (more natural looking than **35.000**, which is set by default).
6. Under the **Display** subpanel, check the **Only Render** item:

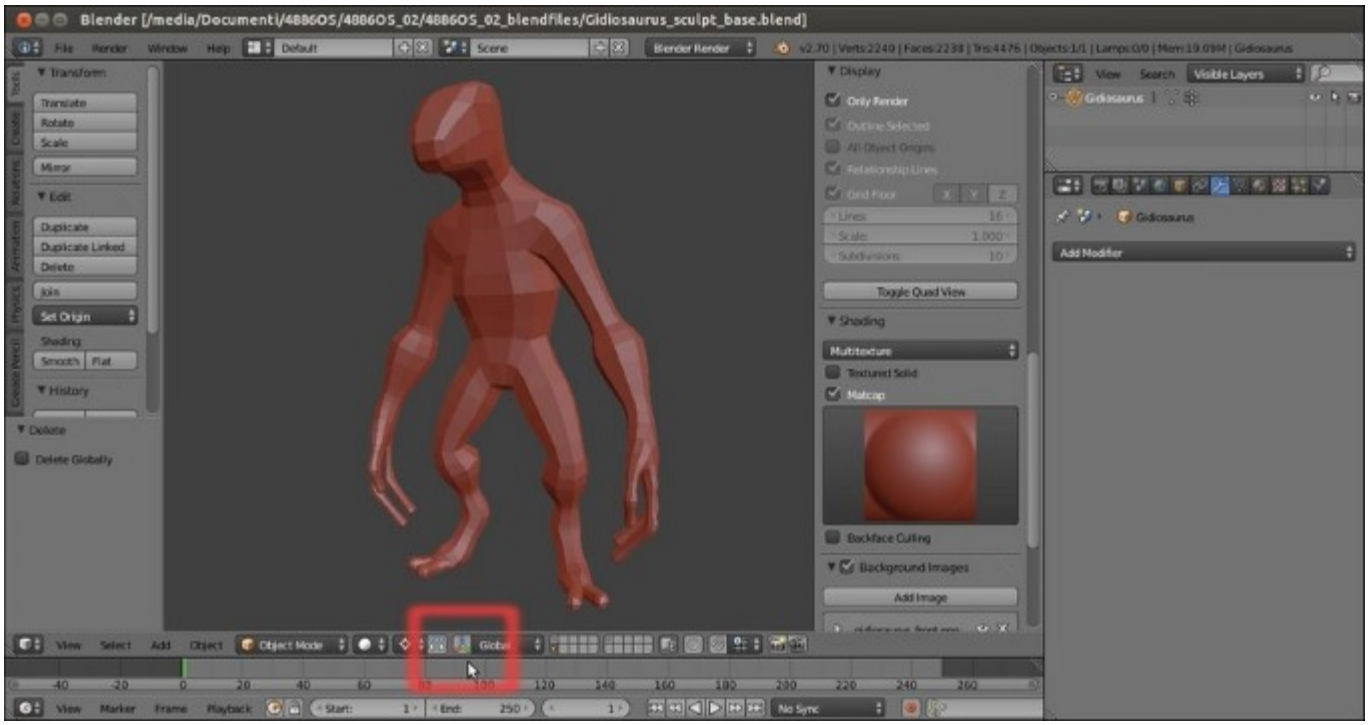


Setting the view through the 3D window N sidepanel

7. Go to the **Shading** subpanel on the sidepanel on the right-hand side of the 3D viewport and check the **Matcap** item.
8. Left-click on the preview window that just appeared and, from the pop-up panel, select the red colored brick material, the one that looks like ZBrush material; obviously, you can choose a different one if you prefer, but in my experience, this is the one that gives the best visual feedback in the 3D view:

The available matcaps menu and the selected Zbrush-like matcap

9. Put the mouse cursor inside the active 3D window and press *Ctrl* + Spacebar to disable the widget:



The matcap assigned to the mesh and the widget button in the 3D window toolbar

10. Press *N* to get rid of the **Properties** 3D window sidepanel.
11. Save the file as `Gidiosaurus_Sculpt_base.blend`.

How it works…

By checking the **Only Render** item in the **Display** subpanel under the **Properties** 3D window sidepanel, all the possible disturbing elements that cannot be rendered (such as the **Grid Floor**, **Empties**, **Lamps**, and so on) are hidden, in order to give a clean 3D viewport ready for sculpting.

Note that with this option enabled, sadly, the **Image Empties** we set in the previous chapter to work as templates for references are not visible—instead, the templates we had set as **Background Images** are perfectly visible in the **3** orthographic views.

Matcaps can in some cases slow the performance of your computer, depending on the hardware; in any case, **Matcaps** is a very useful feature, especially for sculpting, as you can see the mesh shape easily.

Changing the **Lens** angle from **35.000** to **60.000** makes the perspective view look more similar to the natural human field of view.

Using the Multiresolution modifier and the Dynamic topology feature

To be sculpted, a mesh needs a big enough amount of vertices to allow the adding of details; in short, we now need a way to add (a lot of!) geometry to our simple base mesh.

Besides the usual subdividing operation in **Edit Mode** (press *Tab*, then *A* to select all the vertices, then press *W* to call the **Specials** menu, click on **Subdivide**, and then set the **Number of Cuts** value in the last operation subpanel at the bottom of the **Tool Shelf**) and the **Subdivision Surface** modifier, in Blender, there are two other ways to increase the amount of vertices: one is by assigning a **Multiresolution** modifier to the mesh (a nondestructive way) and the other is by using the **Dynamic topology** feature. We are going to see both of them.

Getting ready

As usual, let's start from the last `.blend` file we saved: in this case, `Gidiosaurus_Sculpt_base.blend`.

How to do it…

Let's start with the **Multiresolution** modifier method:

1. First of all, save the file as `Gidiosaurus_Multires.blend`.
2. Select the base mesh and go to the **Object Modifiers** window under the main **Properties** panel on the right-hand side of the screen; assign a **Multiresolution** modifier.
3. Click on the **Subdivide** (*Add a new level of subdivision*) button **3** times; the mesh has now reached **143,234** vertices and **143,232** faces.
4. Check the **Optimal Display** item in the modifier panel:

The mesh with a Multiresolution modifier assigned at level 3 of subdivision

5. On the toolbar of the 3D window, click on the mode button to go into **Sculpt Mode**.
6. On the **Tools** tab on the left-hand side of the screen (if necessary, press the *T* key to make the **Tool Shelf** containing the tabs appear), go to the **Symmetry\Lock** subpanel and click on the **X** button under the **Mirror** item.
7. Click on the **Options** tab and, under the **Options** subpanel, uncheck the **Size** item under **Unified Settings**.
8. Start to sculpt.

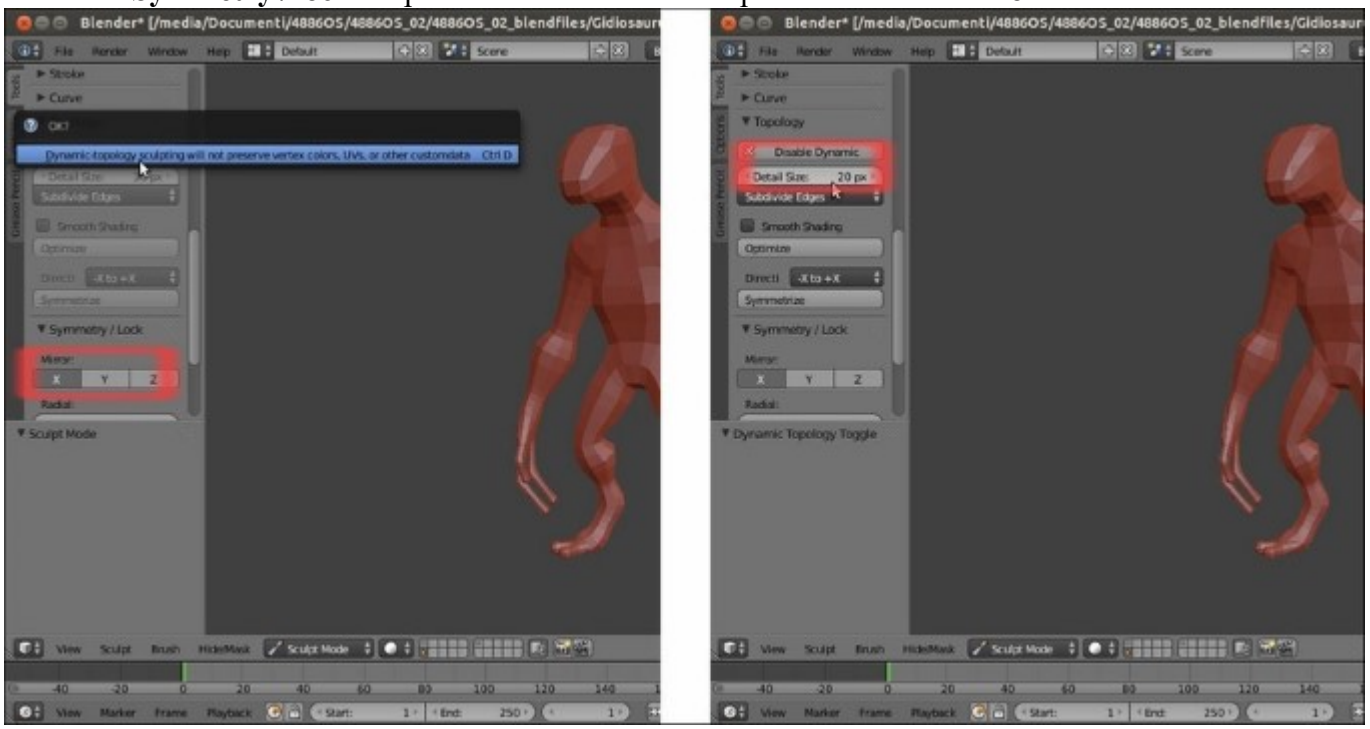
   At this point, to proceed with the sculpting, you should jump to the next recipe, *Sculpting the character's base mesh*; instead, let's suppose that we have already sculpted our base mesh, so let's move ahead:
9. Exit **Sculpt Mode**.
10. Save the file.

Now, let's see the quick and easy preparation necessary to use the **Dynamic topology** feature for sculpting:

1. Reload the `Gidiosaurus_Sculpt_base.blend` blend file.
2. Then, save it as `Gidiosaurus_Dynatopo.blend`.
3. On the toolbar of the 3D window, select **Sculpt Mode**.
4. On the **Tools** tab on the left-hand side of the screen (press the *T* key to make the **Tool Shelf** containing the tabs appear), go to the **Topology** subpanel and click on the **Enable Dyntopo** button; a popup appears to inform you that the **Dynamic topology** feature doesn't preserve any

already existing **Vertex Color**, **UV layer**, or other custom data (only if the mesh has them). Then click on the popup to confirm and go on.

5. Change the **Detail Size** value to **15/20** pixels.
6. Go to the **Symmetry\Lock** subpanel and click on the **X** option under the **Mirror** item:



The dynamic topology tool warning and the settings

7. Start to sculpt.

Again, here you can jump to the next recipe, *Sculpting the character's base mesh*; in any case, remember to save the file.

How it works…

The **Multiresolution** modifier increasingly subdivides the mesh at each level by adding vertices; we have seen that from **2,240** starting vertices of the base mesh, we have reached **143,234** vertices at level **3**, and clearly this allows for the sculpting of details and different shapes. The vertices added by the modifier are *virtual*, exactly as the vertices added by the **Subdivision Surface** modifier are; the difference is that the vertices added by the latter are not editable (unless you apply the modifier, but this would be counterproductive), while it's possible to edit (normally through the sculpting) the vertices at each level of subdivision of a **Multiresolution** modifier. Moreover, it's always possible to go back by lowering the levels of subdivision, and the sculpted details will be stored and shown only in the higher levels; this means that the **Multiresolution** method is a nondestructive one and we can, for example, rig the mesh at level **0** and render it at the highest/sculpted level.

The **Dynamic topology** setting is different from the **Multiresolution** modifier because it allows you to sculpt the mesh without the need to heavily subdivide it first, that is, the mesh gets subdivided on the fly *only where needed*, according to the workflow of the brushes and settings, resulting in a much lower vertex count for the final mesh in the end.

As you can see in the screenshots (and in the `.blend` files provided with this cookbook), starting to sculpt the character with the **Multiresolution** modifier or the **Dynamic topology** is quite different. In the end, the process of sculpting is basically the same, but in the first case, you have an already smoothed-looking mesh where you must add or carve features; in the second case, the low resolution base mesh doesn't change its raw look at all until a part gets sculpted and therefore subdivided and modified, that is, all the corners and edges must first be softened, in order to round an otherwise harsh shape.

Sculpting the character's base mesh

Whatever the method you are going to use, it's now time to start with the effective sculpting process.

However, first, a disclaimer: in this recipe, I'm not going to teach you *how to sculpt*, nor is this an anatomy lesson of any kind. For these things, a book itself wouldn't be enough. I'm just going to demonstrate the use of the Blender sculpting tools, showing what brush I used for the different tasks, the sculpting workflow following the reference templates, and some of the more frequently used shortcut keys.

Getting ready

In this recipe, we'll use the **Dynamic topology** method. If you haven't followed the instructions of the previous recipe, just follow the steps from 12 to 17; otherwise, just open the `Gidiosaurus_Dynatopo.blend` file that is provided.

How to do it…

As usual, it's a good habit to save the file with the proper name as the first thing; in this case, save it as `Gidiosaurus_Dynatopo_Sculpt.blend`.

If you are going to use a graphic tablet to sculpt, remember to enable the tablet pressure sensitivity for both size and strength; in any case, it is better to set the respective sliders to values lower than **100 percent**; I usually set the size slider around **30/35** and the strength slider to **0.500**, but this is subjective:

The tablet pressure sensitivity buttons for the size and the strength

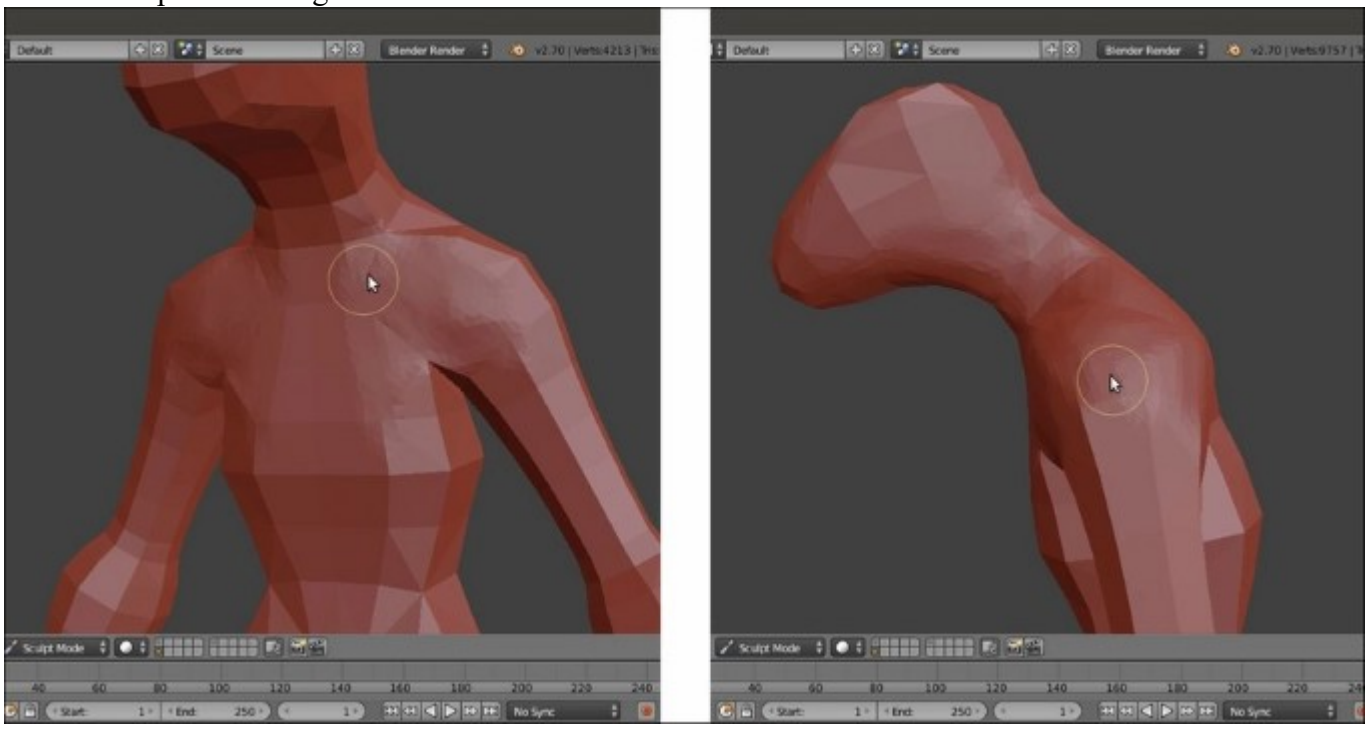
1. If you haven't already, go into **Sculpt Mode** and enable the **Dynamic topology** feature by clicking on the **Enable Dyntopo** button in the subpanel with the same name under the **Tool Shelf** panel or by directly pressing *Ctrl* + *D*.
2. Set the **Detail Size** value to **15**, either by using the slider under the **Enable Dyntopo** button or by pressing *Shift* + *D* and then moving the mouse to scale it bigger or smaller:

Starting to sculpt

3. Click on the **Brush** selection image (*Brush datablock for storing brush settings for painting and sculpting*) at the top of the **Tools** tab under the **Tool Shelf** panel, and, from the pop-up menu, select the **Scrape/Peaks** brush (otherwise press the *Shift* + *3* key shortcut):

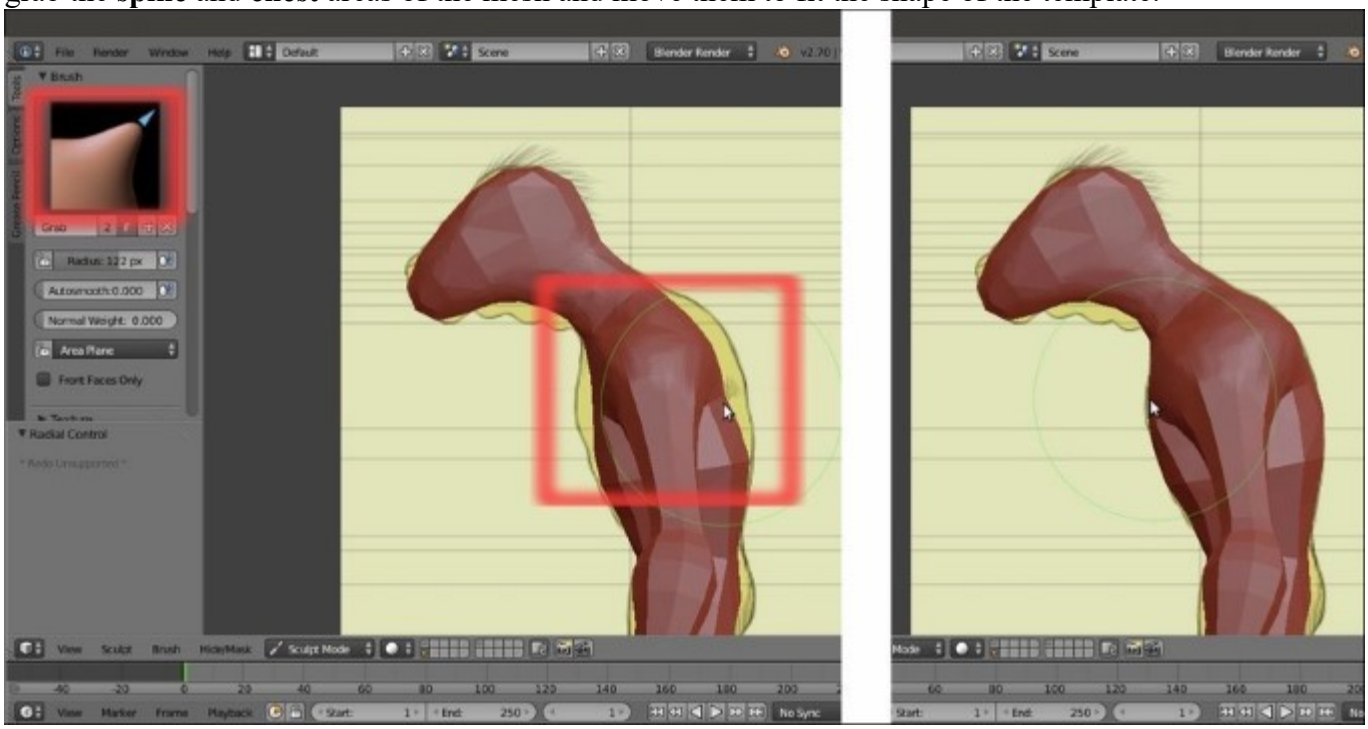
Selecting the Scrape/Peaks brush in the sculpt brushes menu

4. Start to scrape all the edges and soften the corners to obtain a smooth rounded surface:



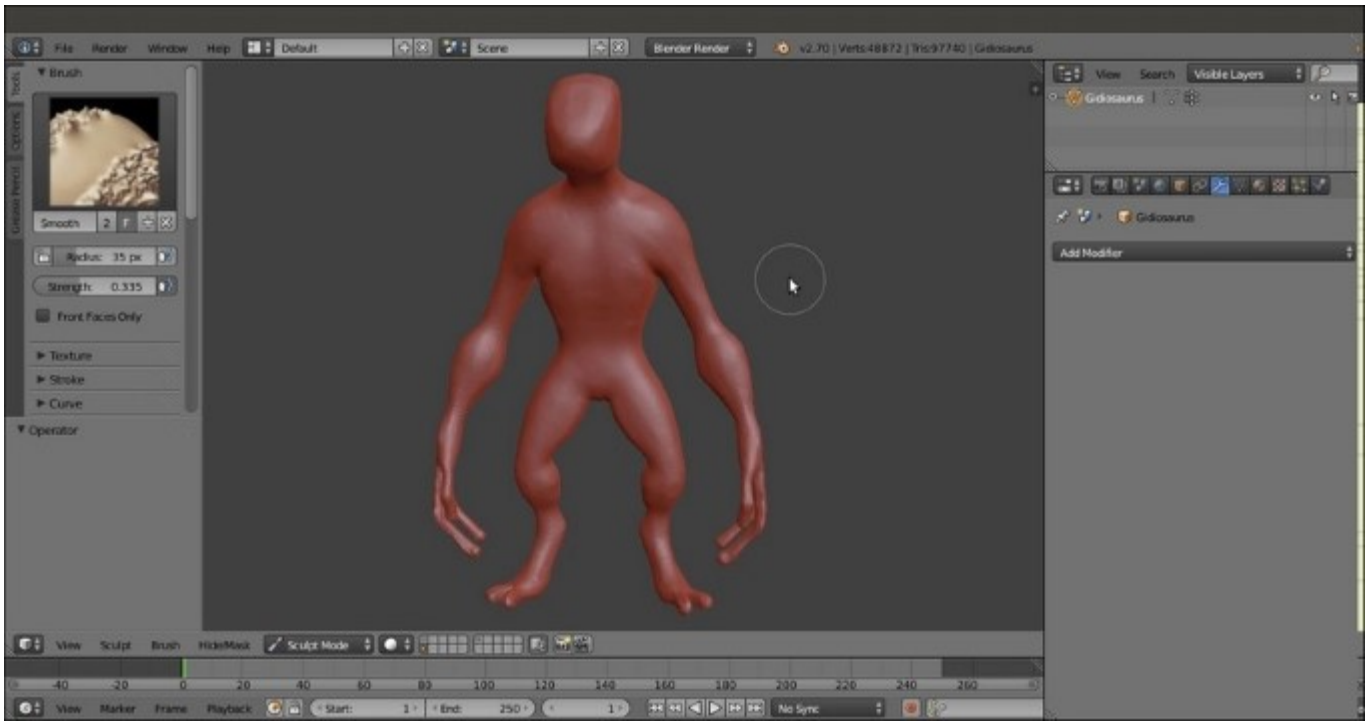
Softening the edges of the mesh

5. Change the brush; select the **Grab** brush (*G* key) and press *3* in the numpad to go into **Side** view; press the *F* key and move the mouse cursor to scale the brush, in this case, to scale it much bigger, around **120** pixels (*Shift* + *F* is to change the strength of a brush, instead).
6. Using the **Background Image** showing in the orthographic view (the *5* key in the numpad), grab the **spine** and **chest** areas of the mesh and move them to fit the shape of the template:



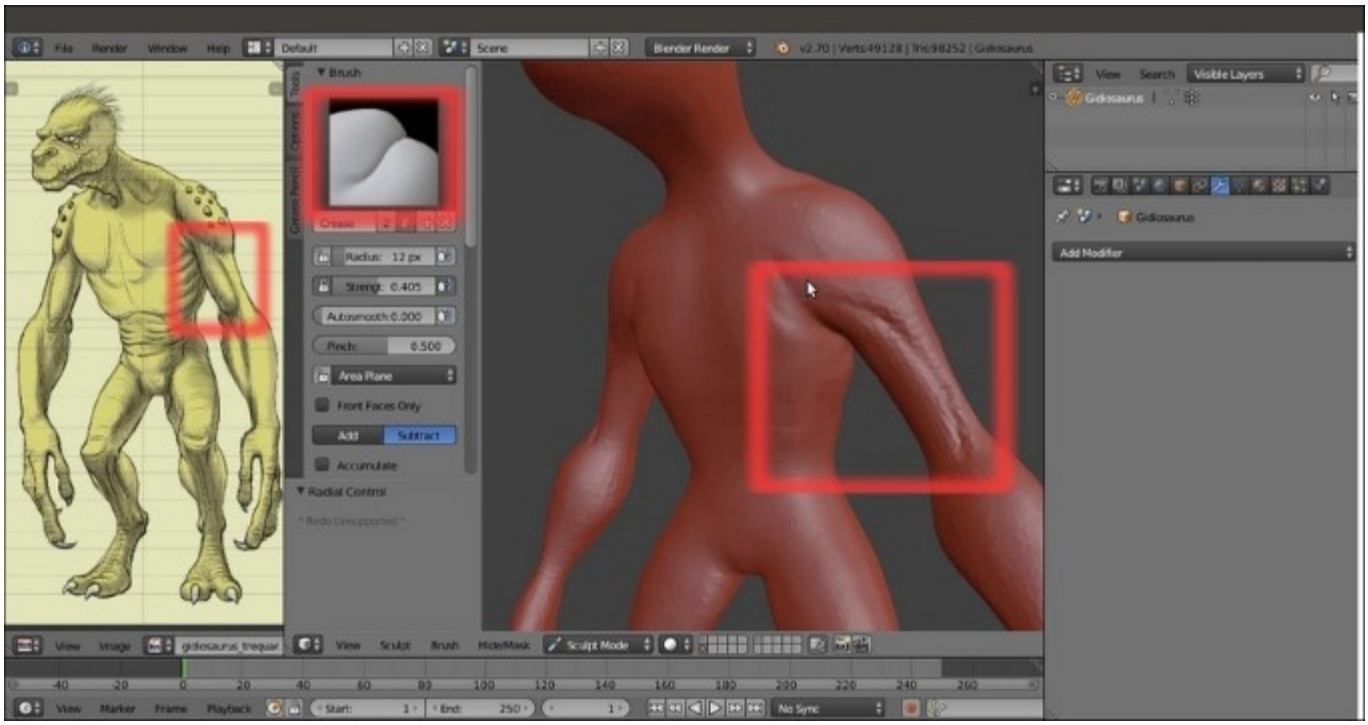
Using the Grab brush to modify the mesh

7. Do the same for the other parts of the mesh that don't fit yet, and do it also in **Front** view (*1* key in the numpad) and **Back** view (*Shift* + *1* in the numpad).
8. Select the **Scrape/Peaks** brush again (*Shift* + *3* keys) and keep on softening the mesh until almost every part gets rounded and more organic-looking; you can also use the **Smooth** (*S* key) brush to further soften the mesh:



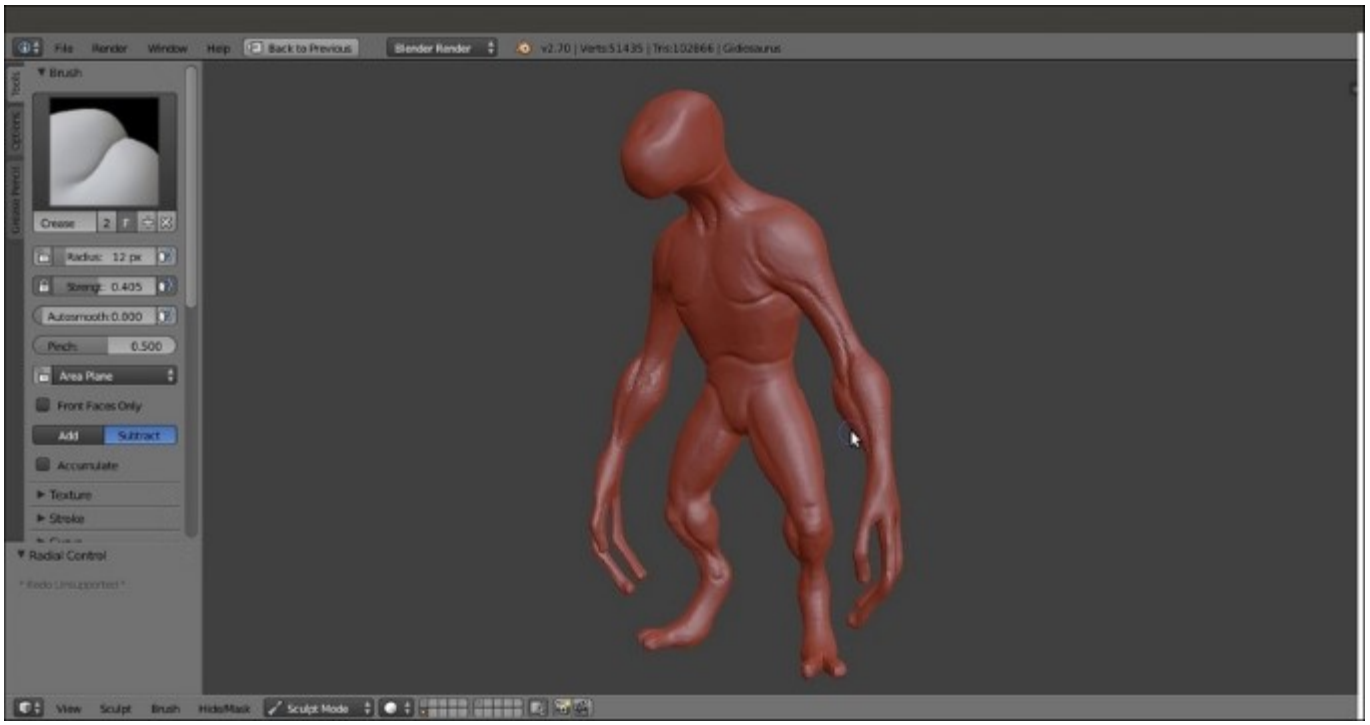
The character is starting to take a shape

9. Open a new window, switch **Editor Type** to **UV/Image Editor** and click on the **Open** button in the toolbar; browse to the `templates` folder and select the `gidiosaurus_trequarters.png` image. Then, click on the little pin icon on the right-hand side of the image name on the window toolbar (*Display current image regardless of object selection*).
10. Select the **Crease** brush (*4* key); using it as a chisel and following the loaded image as a reference, start to outline the character's more important features on the mesh, *drawing* the character's anatomy:



Using the Crease brush as a chisel

11. By pressing *Ctrl* while sculpting, we can temporarily reverse the effect of the brush; so, for example, the **Crease** brush, which usually carves lines in the mesh, can sculpt ridges and spike protrusions. We can use this to add details to the **elbow bones** and **knees** on the fly.
12. By pressing the *Shift* key while sculpting instead, we can temporarily switch whatever brush we are using with the **Smooth** brush, in order to instantly soften any newly added detail or feature.

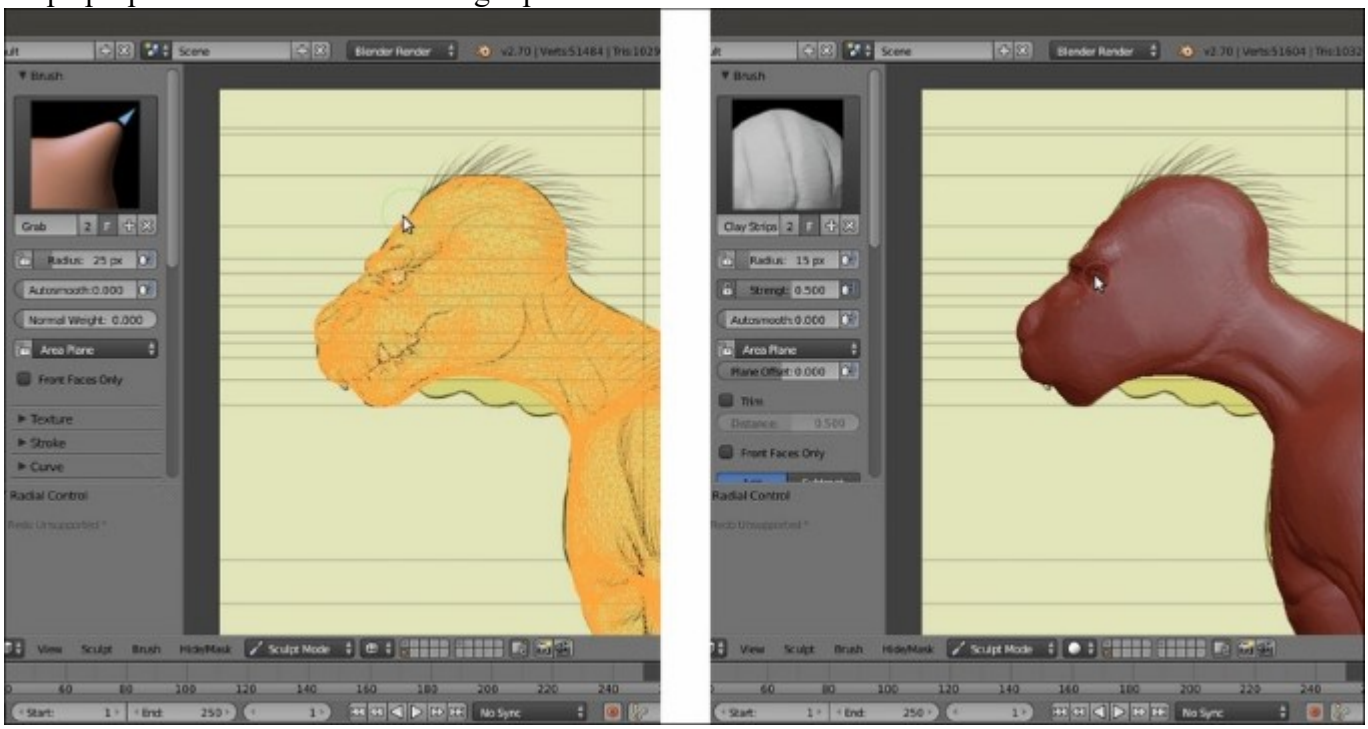


Outlining the major body features

13. When finished with the **body**, exchange the brush for the **Clay Strips** brush (*3* key), start to add stuff (the **nose**, **eyebrows**, and so on), and outline the features of the **head**. Again, press *Ctrl* to subtract clay (for the **eye sockets**, for instance) and *Shift* to soften.

Using the Clay Strips brush to add details and/or carve stuff

14. Always use the templates to check for the proportions and positions of the character's features. Also, use **Wireframe** mode if necessary, by going into **Ortho** view and comparing the sculpted mesh outline with the background template image; use the **Grab** brush to quickly move and shape proportionate features in the right places:

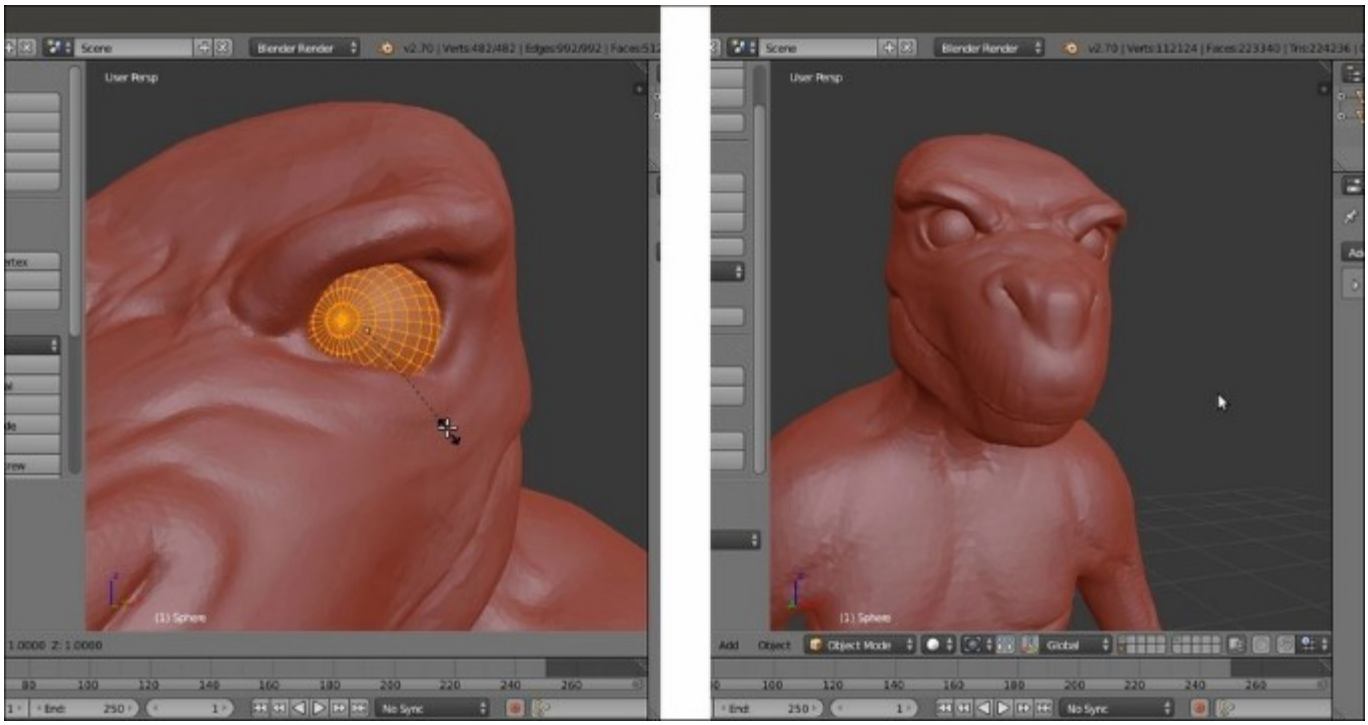


Temporarily switching to the Wireframe viewport shading mode to check the proportions in Side view

15. Using the **Clay Strips** (*3* key), **Smooth** (*S* key), **SculptDraw** (*Shift* + *4* key), **Crease** (*4* key), and **Pinch** (*Shift* + *2* key) brushes, build the **head** of the creature and define as many details as possible such as the **eyebrows**, **mouth rim**, **nostrils**, and **eye sockets**; experiment with all the different brushes:

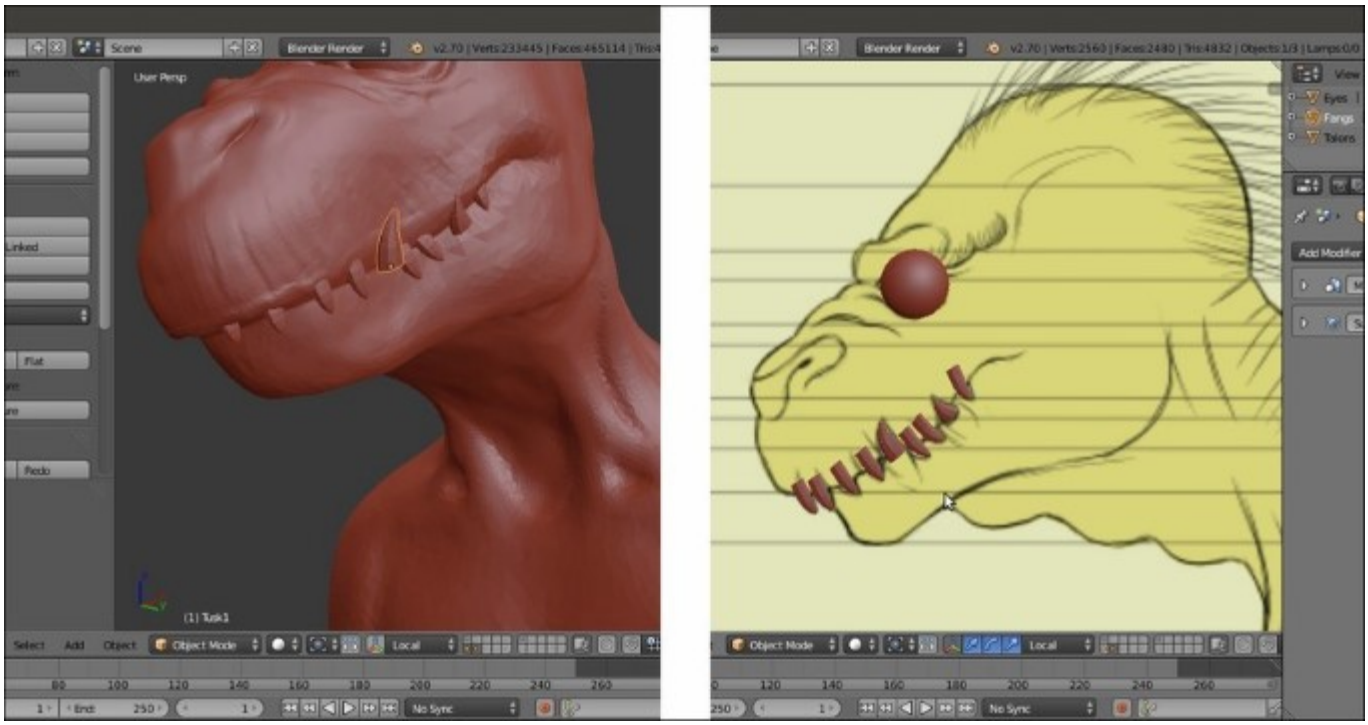
Detailing the head

16. Go out of **Sculpt Mode** and press *N* to make the **Properties** 3D window sidepanel appear; uncheck the **Only Render** item under the **Display** subpanel.
17. Press *Shift* + *A* and add a **UV Sphere** to the scene. Go into **Edit Mode**, if you haven't done so already, select all the vertices and rotate them **90** degrees on the *x* axis; then, scale them to **0.1000**. Finally, scale them again to **0.3600**.
18. Exit **Edit Mode** and move the **UV Sphere** to fit inside the **left eye socket** location.
19. Select the character's mesh and press *Shift* + *S*; then, in the **Snap** pop-up menu, choose **Cursor to Selected**. Select the **UV Sphere** and go to the **Tools** tab under the **Tool Shelf**; click on the **Set Origin** button and choose **Origin to 3D Cursor**. This way we have set the origin of the **UV Sphere** object at the same place as the character's mesh, while the **UV Sphere** mesh itself is located inside the **left eye socket**.
20. Go to the **Object Modifiers** window under the main **Properties** panel and assign a **Mirror** modifier to it.
21. Go to the **Outliner**, press *Ctrl* + left-click on the **UV Sphere** item, and rename it **Eyes**:



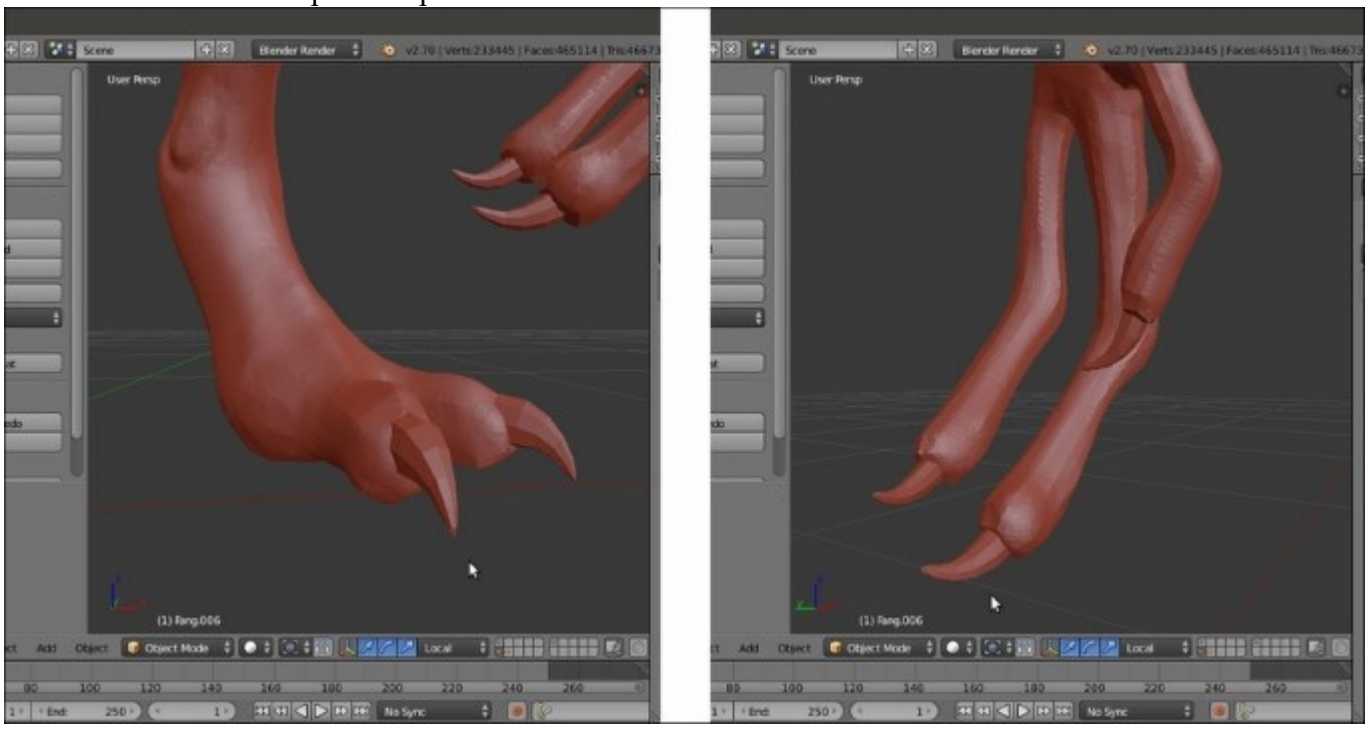
Positioning the eye spheres

22. Press *Shift* + *A* and add a **Cube** primitive. Go into **Edit Mode** and scale it a lot smaller; use the side template as a reference to modify by scaling, extruding, and tweaking the scaled **Cube's** vertices in order to build a low resolution **fang**. Go out of **Edit Mode** and go to the **Object Modifiers** window to assign a **Subdivision Surface** modifier.
23. Duplicate the **fang** and, as always, following the side and front templates as a guide, build all the necessary **teeth** for the **Gidiosaurus**.
24. Select all of them and press *Ctrl* + *J* to join them into one single object; press *Ctrl* + *A* to apply **Rotation & Scale**; then, do the same as in steps 19 and 20.
25. Go to the **Outliner** and rename the new object **Fangs**.



Making the teeth

26. Add a new **Cube** and repeat the process to model the **talons** of the **hands** and **feet**:



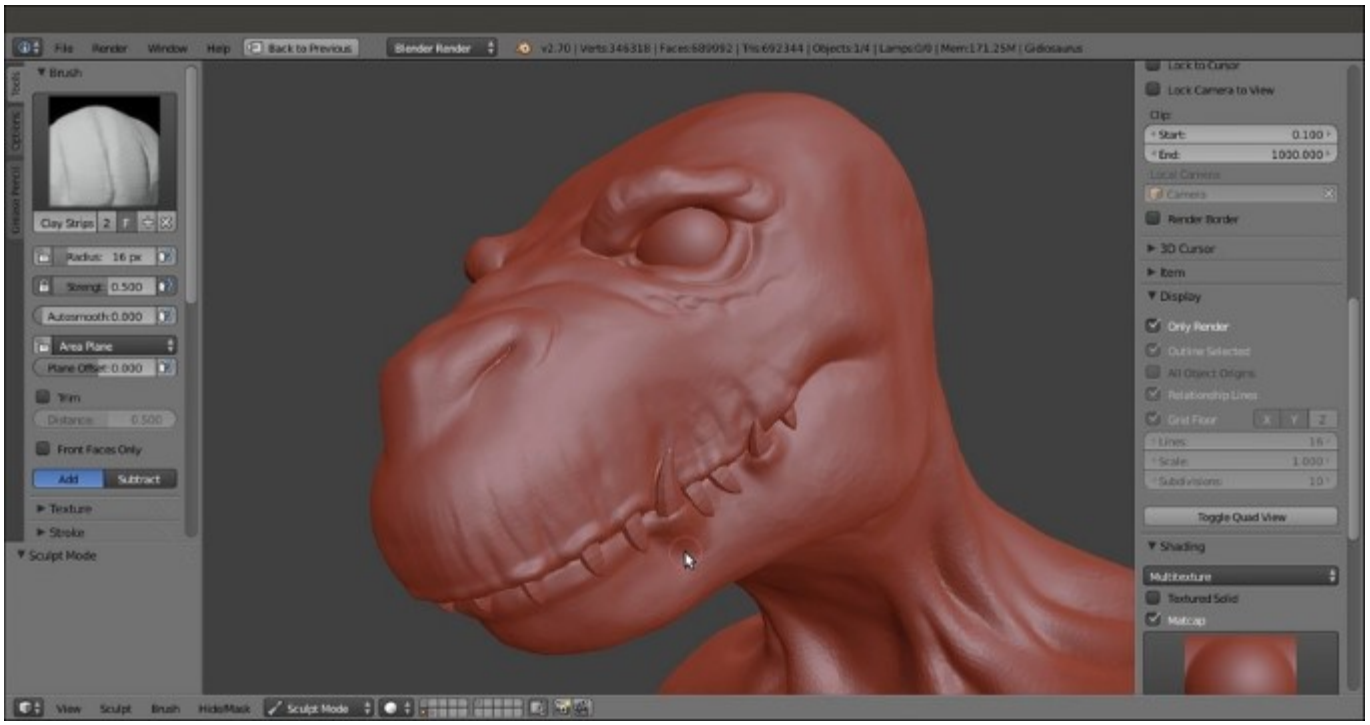
Making the talons

Note that the **Eyes**, **Fangs**, and **Talons** objects are not going to be sculpted, and therefore they are kept as separate objects. Later, we'll start to retopologize the sculpted body of the creature, while the **eyes** will be modeled and detailed in the traditional polygonal way; **fangs** and **talons** are good enough as they are.

27. Reselect the **Gidiosaurus** object and go back into **Sculpt Mode** to keep on refining the creature's shape more and more; don't be afraid to exaggerate the features, we can always smooth them later.

The almost completed sculpted mesh

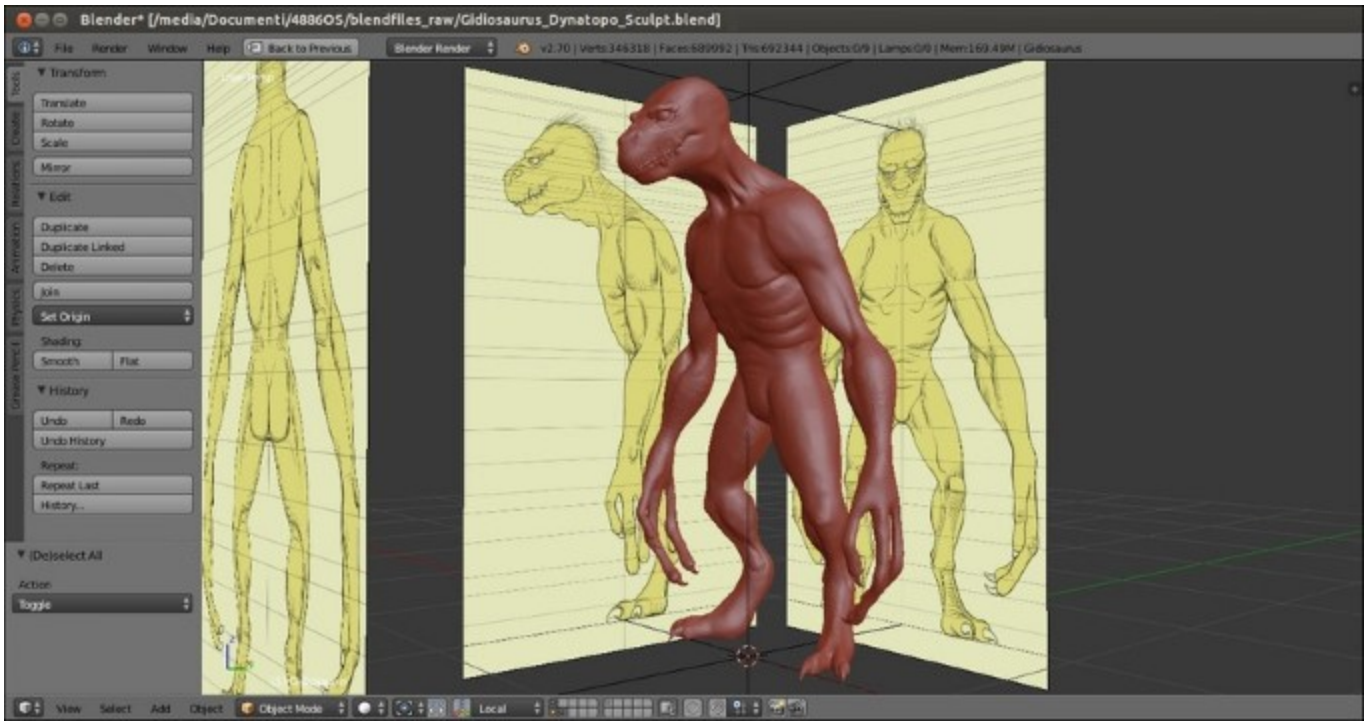
28. Adjust the shape of the **eyebrows** to perfectly fit the **Eyes** object; then, work more on the **mouth rim** to accommodate the **fangs**.



Refining the eyebrows and the mouth rim

29. When you think you have arrived at a good enough point, just go out of **Sculpt Mode** and *remember to save the file!*

Just a quick note: we don't actually need to go out of **Sculpt Mode** to save the file, it's possible to save it periodically (press *Ctrl* + *S* or *Ctrl* + *W* to save the file over itself, and *Ctrl* + *Shift* + *S* to *save as*) without needing to exit the sculpting session each time.



The completed sculpted mesh compared with the reference templates

So, here we are; the character's sculpting is basically done. We can work on it a lot more, tweaking the shapes further and adding details such as **scales**, **wrinkles**, and **veins**, but for this exercise's sake (and for this recipe), this is enough.

There's more…

A nice aspect of the **Dynamic topology** feature is the possibility to actually join different objects into a single mesh; for example, with our **Gidiosaurus**, we can join the **teeth** and the **talons** to the sculpted base mesh and then keep on sculpting the resulting object as a whole.

Actually, there are two ways to do this: simply by joining the objects and by the **Boolean** modifier.

To join the objects in the usual way, we can do the following:

1. Go out of **Sculpt Mode**.
2. Select the first object (that is, the **teeth**), press *Shift* + select the second object (the **talons**), and lastly press *Shift* + select the sculpted base mesh so that it's the active object (the final composited object will retain the active object's characteristics).
3. Press *Ctrl* + *J* and it is done!

This is the way you join objects in Blender in general, and it can actually work quite well. There is only one problem: there will always be a visible seam between the different objects, and although in the case

of **teeth** or **talons** this will not be a problem, in other cases it should be avoided. Let's say you are working on a separated **head** and later you want to join it to a **body**; in this case, you don't want a visible seam between the **head** and **neck**, obviously!

So, the option is to use the **Boolean** modifier:

1. Go out of **Sculpt Mode**.
2. Select the character's base mesh and go to the **Object Modifiers** window under the main **Properties** panel; assign a **Boolean** modifier.
3. Click on the **Object** field of the modifier to select the object you want to join (let's say, the **Talons** object) and then click on the **Operation** button to the left to select **Union**.
4. Click on the **Apply** button to apply the modifier.
5. Hide, move onto a different layer, or delete the original object you joined (the **talons**).

Unlike the previous method, with Booleans, it will be possible to sculpt and smooth the joining of the different objects without leaving visible seams.

Chapter 3. Polygonal Modeling of the Character's Accessories

In this chapter, we will cover the following recipes:

- Preparing the scene for polygonal modeling
- Modeling the eye
- Modeling the armor plates
- Using the Mesh to Curve technique to add details

Introduction

In the previous two chapters, we did the following:

- Quickly modeled a simple base mesh, as close as possible to the shape of the reference templates
- Sculpted this base mesh, refining the shapes and adding details to some extent

We have also quickly modeled very simple teeth and talons, and placed bare UV Spheres as placeholders for the eyes.

It's now time to start some polygonal modeling to complete the eyes, but especially to build the armor that our character is wearing.

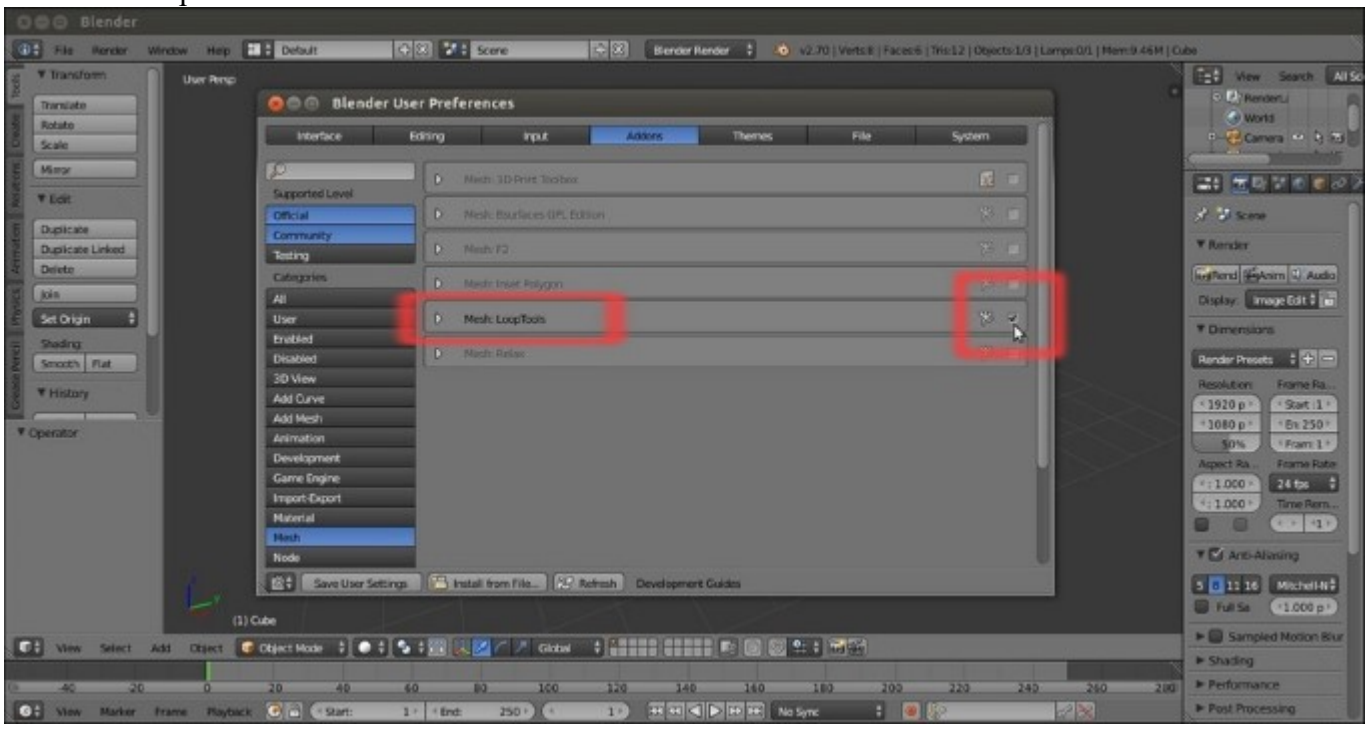
Preparing the scene for polygonal modeling

Coming from a sculpting session, our `.blend` file must first be prepared for the polygonal modeling, verifying that the required add-ons are enabled and all the character's parts are easily visible and recognizable; for this, even though the topic of **Materials** is complex and there will be an entire chapter dedicated to it later in this module, we are going to assign basic materials to these parts so that they have different colors in the 3D viewport.

Getting ready

First, we are going to look for the **LoopTools** add-on, an incredibly useful script by Bartius Crouch that extends the Blender modeling capabilities (and that also has other functionalities, as we'll see in the next chapter about retopology); this add-on is provided with the official Blender release, but still must be enabled. To do this, follow these steps:

1. Start Blender and call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*); go to the **Addons** tab.
2. Under the **Categories** item on the left-hand side of the panel, click on **Mesh**.
3. Check the empty little checkbox on the right-hand side of the **Mesh: LoopTools** add-on to enable it.
4. Click on the **Save User Settings** button at the bottom-left of the panel to save your preferences and close the panel:



The Blender User Preferences panel

5. Open the `Gidiosaurus_Dynatopo_Sculpt.blend` file.

How to do it…

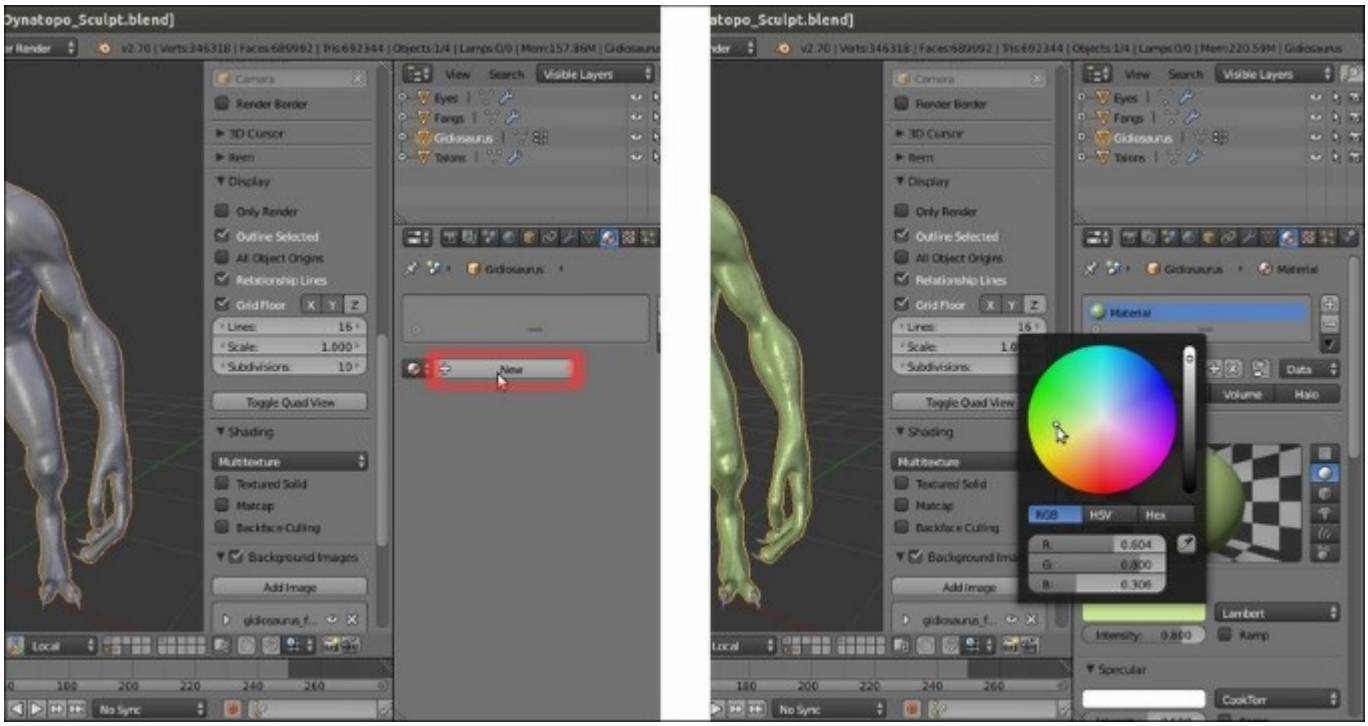
Now, we can start with the scene setup:

1. Click on the **11th** scene layer button (the first one in the second row of the first-left layer block of **Visible Layers** in the toolbar of the 3D window) to make it the only one visible (or else, just put the mouse pointer on the 3D viewport and press the *Alt* + *11* keys; the *Alt* button is to allow for double digits).
2. Press *Shift* + left-click on the **13th** button to multiactivate it (or use the *Shift* + *Alt* + *13* shortcut).
3. Go to the **Outliner** and click on the little grayed arrow icons on the side of the **Eyes**, **Fangs** and **Talons** items to make them selectable again.
4. If not already present, show the **Properties** 3D window sidepanel (*N* key) and go to the **Shading** subpanel; uncheck the **Matcap** item:

Disabling the Matcap item

5. Select the **Gidiosaurus** mesh; go to the **Material** window under the main **Properties** panel to the right and click on the **New** button to assign a material (note that, at least at the moment, we are using the default **Blender Internal** engine); click on the **Diffuse** button and change the color to **RGB 0.604**, **0.800**, **0.306** (a greenish hue, but in this case you can obviously choose any color you wish). Double left-click on the material name inside the data block slot to rename it as `Body`.
6. Select the **Eyes** object and again in the **Material** window under the main **Properties** panel to the right, click on the **New** button to assign a new material; click on the **Diffuse** button and this time change the color to **RGB 0.800**, **0.466**, **0.000**. Rename the material as `Eyes`.

7. Select the **Fangs** object and repeat the process; change the diffuse color to **RGB 0.800**, **0.697**, **0.415**. Rename the material as `Enamel`.
8. Select the **Talons** object and go to the **Material** window under the **Properties** panel to the right; click on the little arrows on the left-side of the **New** button and from the pop-up menu, select the `Enamel` material:



Assigning a material and choosing a color

9. Go to the **UV/Image_Editor** window on the left-hand side of the screen and press *Shift* + left-click on the **X** icon on the right-hand side of the data block name to get rid of the `gidiosaurus_trequarters.png` image. Then, click on the **Open** button, browse to the `templates` folder, and load the `gidiosaurus_armor1.png` image.
10. Save the file as `Gidiosaurus_modeling.blend`.

The armoured character's image loaded in the UV/Image Editor for reference

How it works…

We have deselected the **Matcap** view, assigning also differently colored basic materials to the four parts making up the character's mesh (**body**, **eyes**, **fangs**, and **talons**) to have a clearer way of differentiating the different pieces of the mesh. Then, we have replaced the template we used as reference for the sculpting of the **Gidiosaurus** body with a new one showing the armor as well (in the `templates` folder there are actually two slightly different versions of the armor; we chose the first one).

We have also activated the **13th** scene layer to be ready for the modeling of the **armor** (in the **11th** we have the character's mesh and in the **12th** we have the **fangs**, **talons**, and **eyes**).

Note that, in this cookbook, I will always specify scene layers to indicate the **20** 3D layers accessible from the buttons on the viewport toolbar and distinguish them from other types of layer systems present in Blender, such as for the **bones** or the **Grease Pencil** tool and so on.

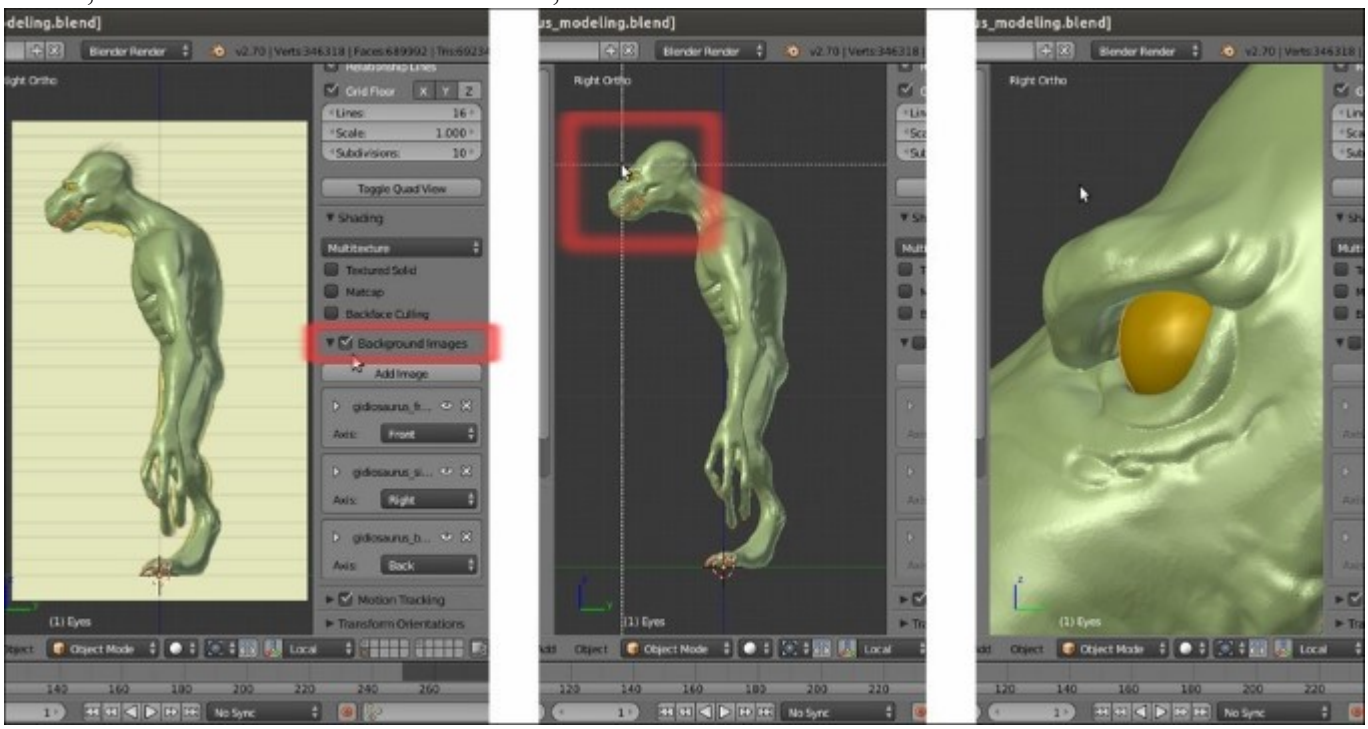
Modeling the eye

It's now time to start to define the creature's **eyes**. We already had **UV Sphere** placeholders, but we're going to refine this mesh to deliver a more convincing eye. By the way, keep in mind that a good portion of the expressiveness of the eye will be due to the use of appropriate textures; for more information, see Chapter 12, *Creating the Materials in Cycles*, and Chapter 13, *Creating the Materials in Blender Internal*.

Getting ready

Following the previous recipe, there is nothing particular to be prepared before starting, except for the following:

1. Go to the **Properties** 3D view sidepanel (*N* key if not already present) and uncheck the **Background Images** item.
2. Press *3* on the numpad to go in **Side** view and zoom to the **UV Sphere** location, by pressing *Shift* + *B* and drawing a box around the point you want to zoom at; as you release the mouse button, the selected area will be zoomed in;



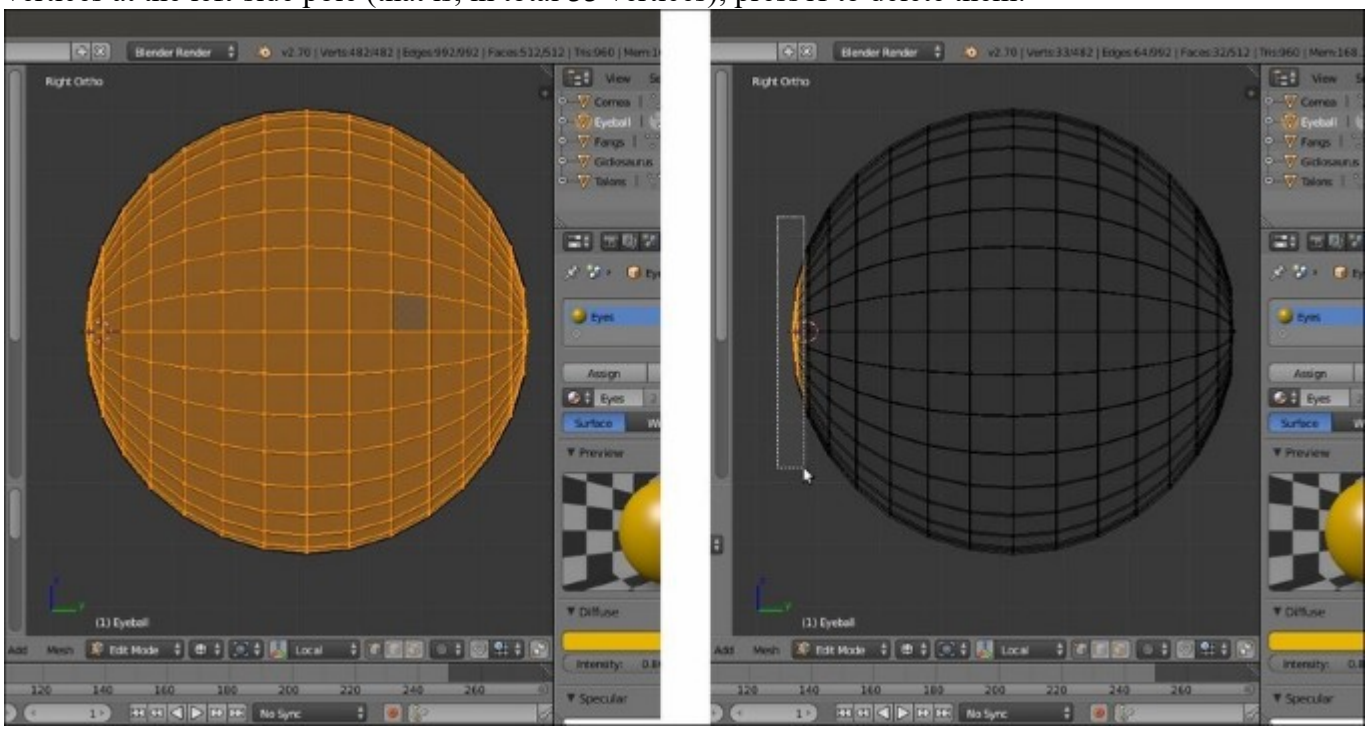
Disabling the background images and zooming to the eyes area

3. Go to the **Outliner** and click on the eye icon on the right-hand side of the **Gidiosaurus** item to hide it; or else, select the mesh in the 3D viewport and press the *H* key. Alternatively, you can also press the slash (*/*) key in the numpad to go in **Local** view, a particular view mode where only the selected objects are still visible (press the slash (*/*) again to go back to the normal view mode).

How to do it…

Without further ado, let us begin to build the eye:

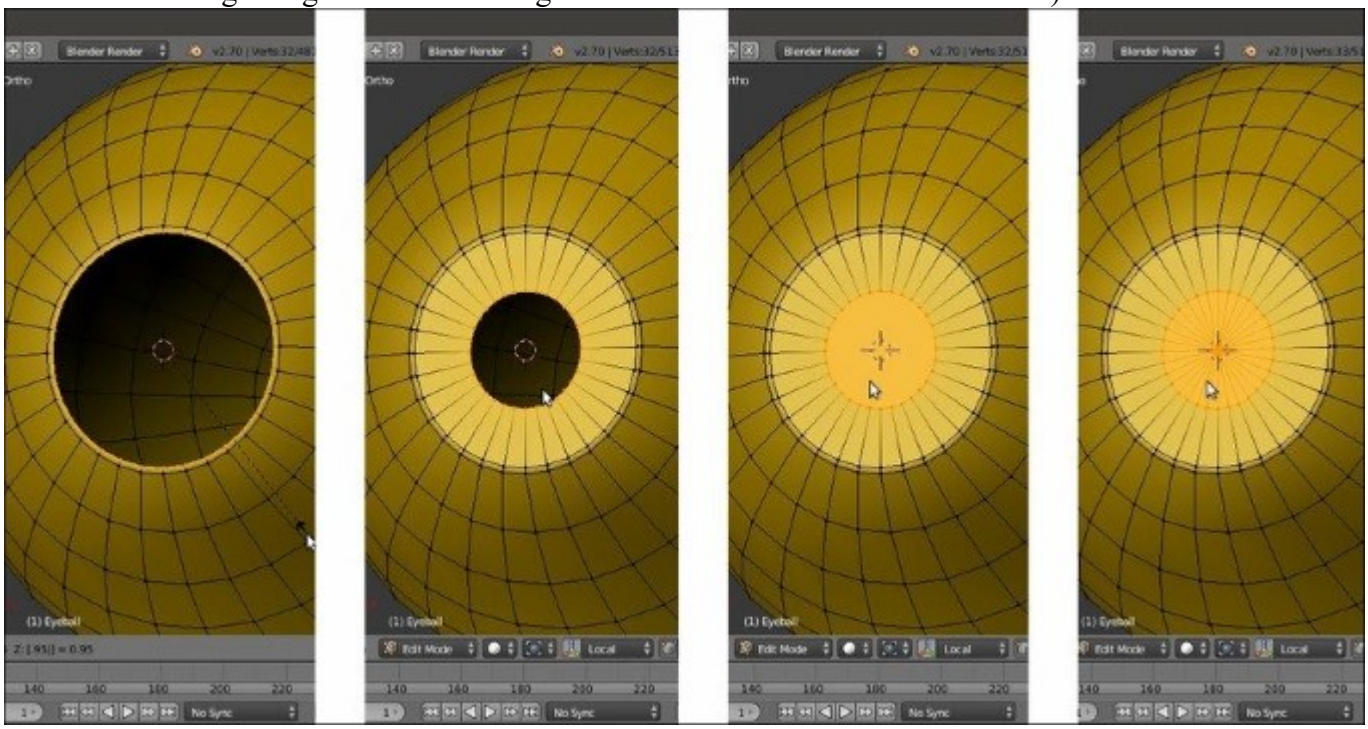
1. Press *Z* to go in the **Wireframe** viewport shading mode.
2. In the **Outliner**, select the **Eyes** item (or else, if you wish, in the 3D viewport, select the **UV Sphere** object) and rename it as **Cornea**.
3. Press *Shift* + *D* and then immediately press the *Esc* key or right-click to cancel the *Grab/ Translate* function, obtaining a duplicated object that now shows as **Cornea.001**; in the **Outliner**, rename the new object as **Eyeball**.
4. Press *Tab* to go in **Edit** Mode; if necessary, press *A* to select all the vertices and scale them to **0.990** (*S* | **.99** | *Enter*).
5. Press *A* to deselect all the vertices. Then, box-select (*B* key) the pole vertex and the first row of vertices at the left-side pole (that is, in total **33** vertices); press *X* to delete them:



Box-selecting the vertices at the UV Sphere pole

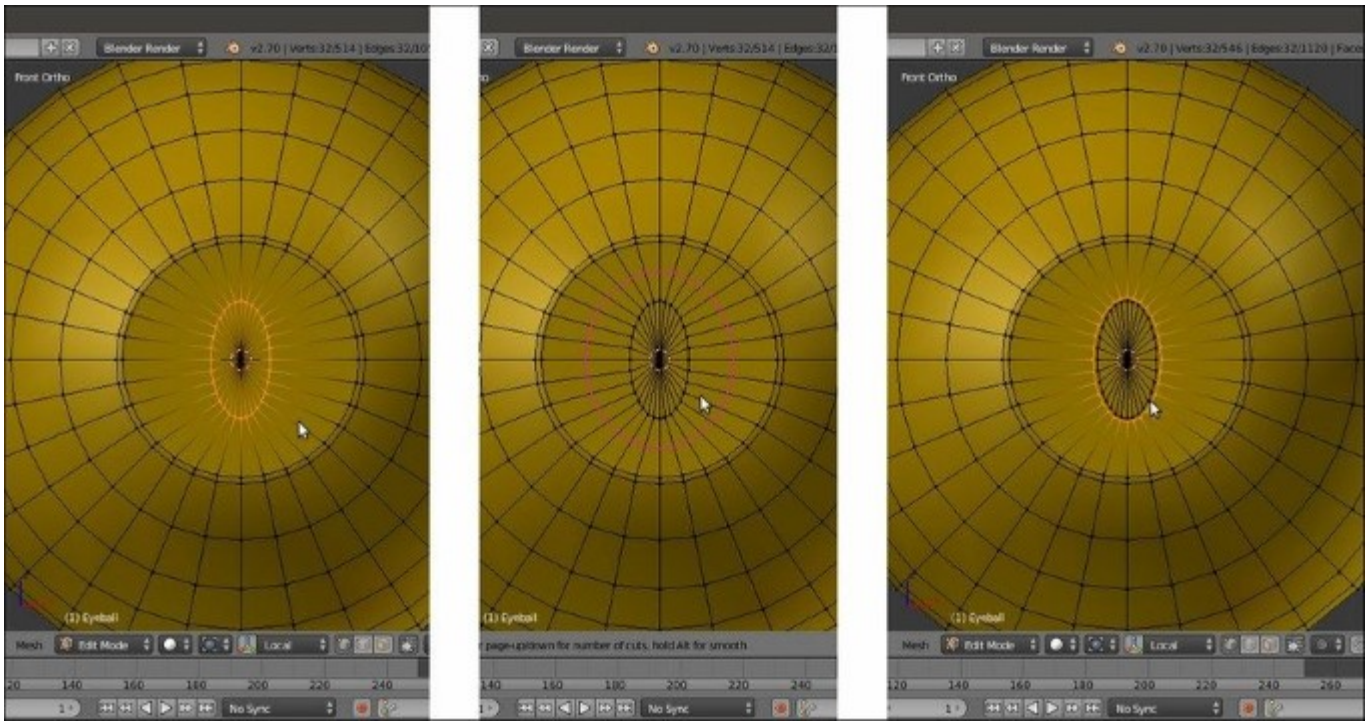
6. Reselect all the remaining vertices; then, press the period (.) key on the numpad to center the view on the selection.
7. Go to the **Outliner** and click on the eye icon on the left-hand side of the **Cornea** item to hide it.
8. Rotate the view to align it with the hole in the **UV Sphere** and, if necessary, press the *5* key on the numpad to go in **Ortho** mode.
9. Press *Z* to go in the **Solid** viewport shading mode and press *A* to deselect everything.
10. Select the first row of vertices around the hole (*Alt* + right-click on the edge-loop). Press *E* to extrude them and then *S* to scale them; keep *Ctrl* + *Shift* pressed and scale to **0.9500** (or else, press *S* | **.95** | *Enter*).

11. Press *E* and *S* again to extrude and scale the vertices to **0.500**.
12. Press *F* to fill the selection and *Alt* + *P* to poke the created N-gon face (that is, to automatically subdivide the single N-gon face into triangular faces connected to a central vertex).



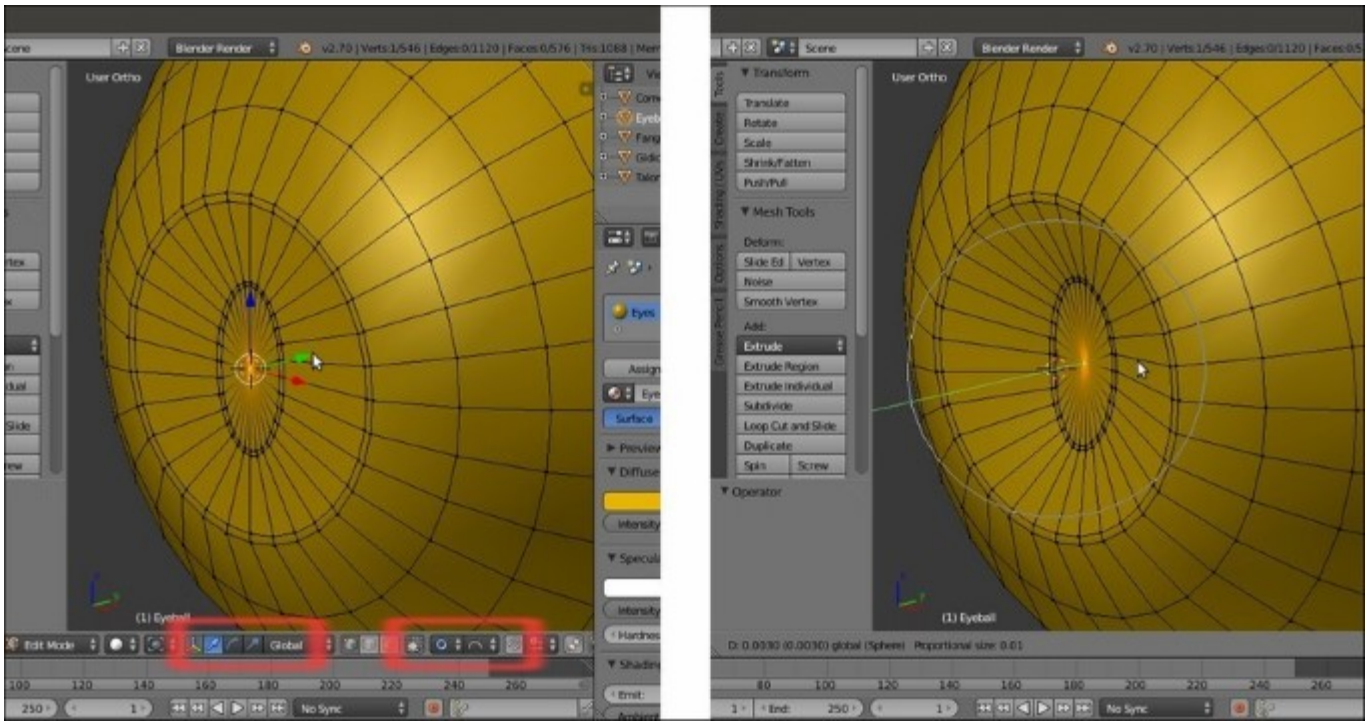
Extruding and closing the eye

13. Press *1* on the numpad to go in **Front** view. Scale the selected vertices to **0.500** on the *x* axis (*S* | *X* | **.5** | *Enter*).
14. Press *Ctrl* + *R* and add an edge-loop outside of the iris; keep *Ctrl* pressed and move the mouse to edge-slide it to **-0.900**.



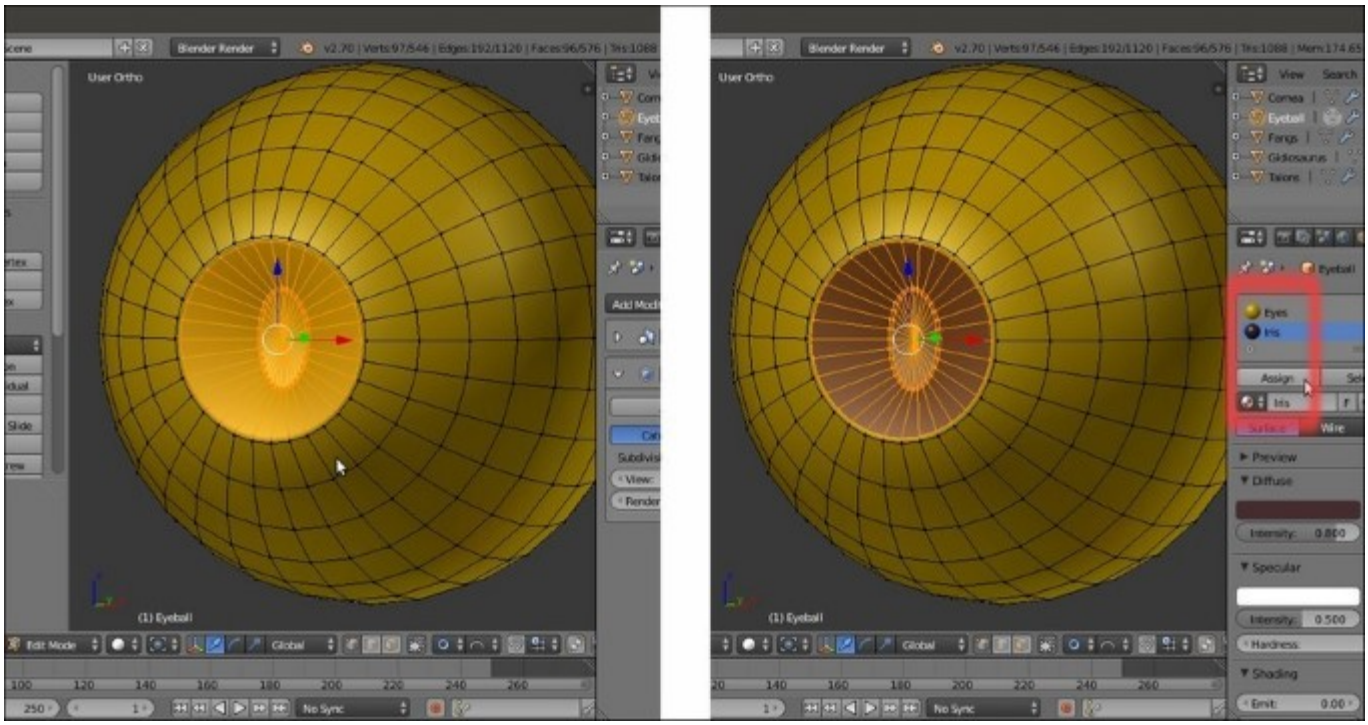
Making the pupil

15. In the toolbar of the 3D window, enable the **PET** (the **Proportional Editing** tool); set it to **Connected** and the **Proportional Editing Falloff** option to **Sphere**.
16. Enable the widget, set it to *Translate* (the second icon from the left, the one with the arrow), set **Transform Orientation** to **Global**, and select the central vertex of the pole. By using the widget, move it on the *y* (green) axis to **0.0030** (click on the green arrow and hold *Shift* for a finer control as you move the mouse on the *y* axis), while with the middle mouse wheel, set the **Proportional size** value of the **PET** to a quite small radius, or **0.01** to be precise:



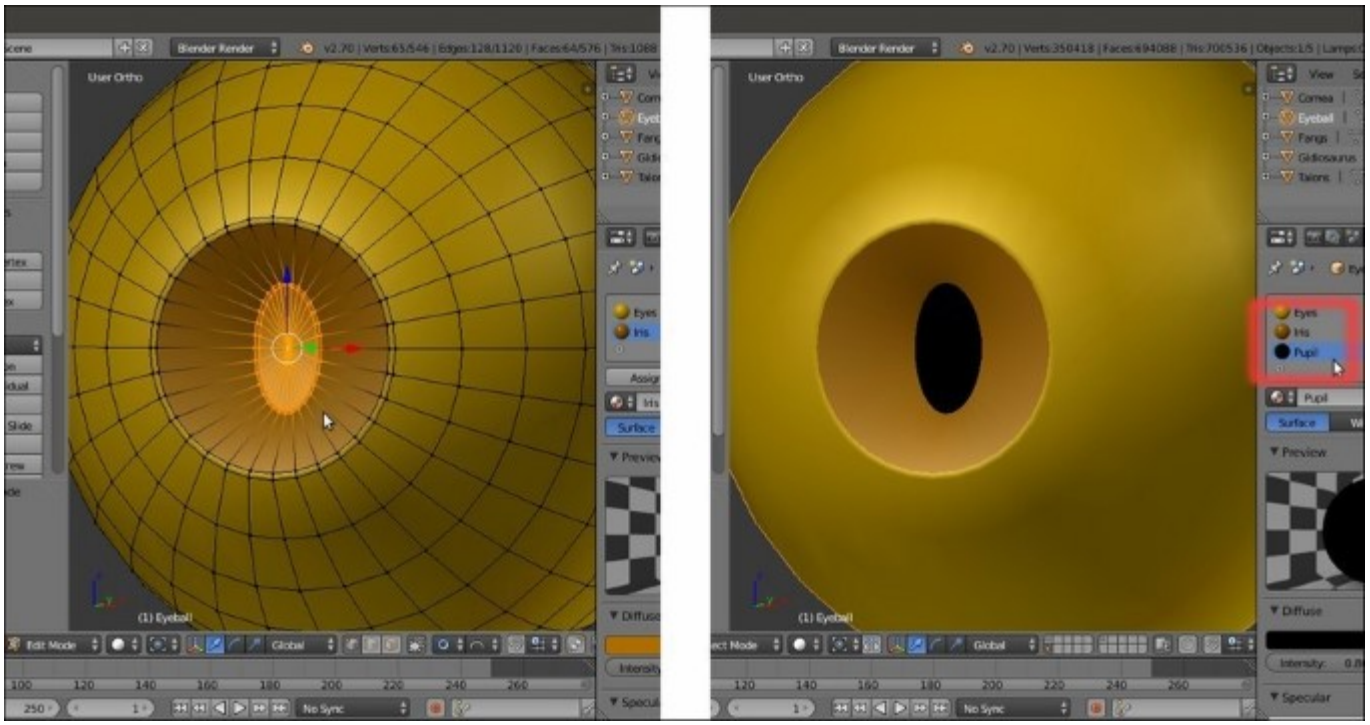
Creating the iris concave shape

17. Press *Ctrl* and the + key on the numpad **3** times, in order to grow the selection starting from the single selected vertex at the center of the iris.
18. Go to the **Material** window, create a new material, and rename it as `Iris`; change its diffuse color to something like **RGB 0.061**, **0.025**, **0.028** and then click on the **Assign** button:



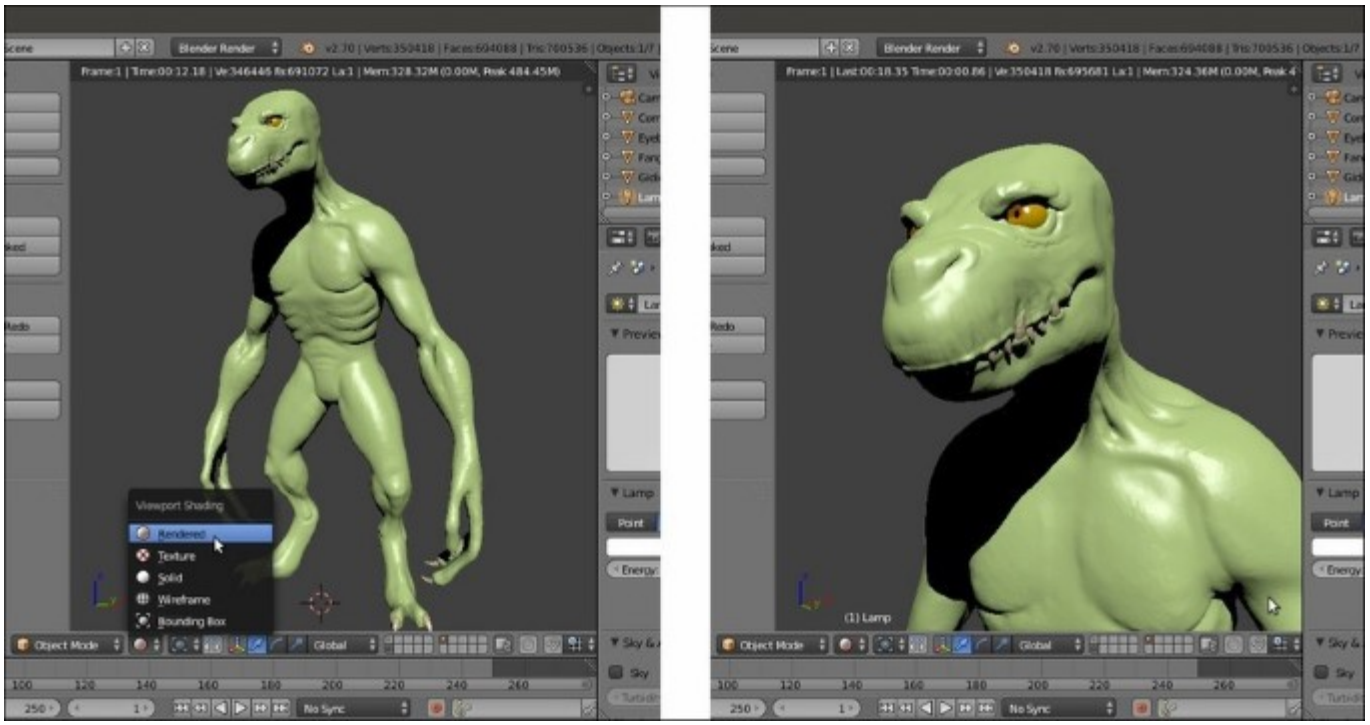
Assigning a material to the iris

19. Press *Ctrl* and the - key on the numpad just **1** time, in order to reduce the selection to the pupil. Go to the **Material** window, create a new material, and rename it as `Pupil`; change its diffuse color to plain black and then click on the **Assign** button.
20. Press *Tab* to go out of **Edit Mode**.



The almost completed eye

21. Go to the **Object Modifiers** window under the main **Properties** panel on the right-hand side of the UI and assign a **Subdivision Surface** modifier; check the **Optimal Display** item.
22. In the **Outliner**, unhide the **Cornea** object and assign a **Subdivision Surface** modifier as well; check the **Optimal Display** item and then hide it again (you can also use the *H* and *Alt* + *H* keys to do this).
23. Select the **Eyeball** object and go to the **Material** window; select the `Pupil` material and go to the **Specular** subpanel to set the **Intensity** value to **0.000**. Set the **Specular Shader Model** option of both the `Eyes` and `Iris` materials to **WardIso** and the **Slope** value to **0.070**. Set the `Iris` material's **Emit** value (under the **Shading** subpanel) to **0.050**.
24. In the **Outliner**, select the **Cornea** object and in the **Material** window, click on the little icon reporting **2** on the right-hand side of the material name (it's the display of the number of users for that material). The name `Eyes` automatically changes to `Eyes.001`: rename it `Cornea`; then, go to the **Transparency** subpanel and enable it. Set the **Fresnel** value to **1.400** and the **Blend** factor to **2.000**. Go to the **Options** subpanel further down and uncheck the **Traceable** item.
25. Unhide the **Gidiosaurus** mesh (*Alt* + *H*) and enable the **6th** scene layer (the one with the **Camera** and the **Lamp**). Select the **Lamp** and in the **Object Data** window, change the type to **Sun** and then rotate it to: **X = 55.788948°**, **Y = 16.162031°**, and **Z = 19.84318°**; you can press *N* and then type these values in the slots of the **Rotation** panel at the top of the **Properties** 3D window sidepanel.
26. Press *N* to hide again the **Properties** 3D window sidepanel and in the toolbar of the 3D window, go to the **Viewport Shading** button and select **Rendered** (or directly press the *Shift* + *Z* shortcut) to have a nice preview of the effect:



The Rendered preview of our character so far

27. Save the file.

How it works…

Actually, the **eyes** of the character are composed of two distinct objects: the **Eyeball** and the **Cornea** object.

The **Cornea** object is the transparent layer covering the **Eyeball** object, and by clicking on the eye icon in the **Outliner**, it has been made invisible in the 3D viewport but still renderable. With the **Cornea** object visible in the 3D views, **irises** and **pupils** would have been hidden behind, making the work of animating the **eyes** quite hard; animators always need to know what the character is looking at.

Both the **Cornea** and **Eyeball** objects, at the moment, are mirrored to the right by the **Mirror** modifier; this will be changed when we skin the mesh to the **Armature**.

If you can't find the **Rendered** view in the **Viewport Shading** mode button on the 3D viewport's toolbar, you may want to make sure you have the latest version of Blender; only versions after **2.6** have this feature for the Blender Render engine.

Modeling the armor plates

In the previous recipe, we modeled the character's **eye** and we had already modeled the **teeth** in Chapter 2, *Sculpting the Character's Base Mesh*, because we needed them, at that moment, to go on with the sculpting; they had been made with simple **Cube** primitives quickly scaled and tweaked in **Edit Mode**.

It is now time to model the **armor** for our warrior. Let's begin by creating the hard metal plates. We are going to use an approach similar to the modeling of the **fangs**, which is by starting with a **Cube** primitive and subdividing it to have more geometry to be edited in the proper shape, and we'll also use the **LoopTools** add-on to simplify some processes.

Getting ready

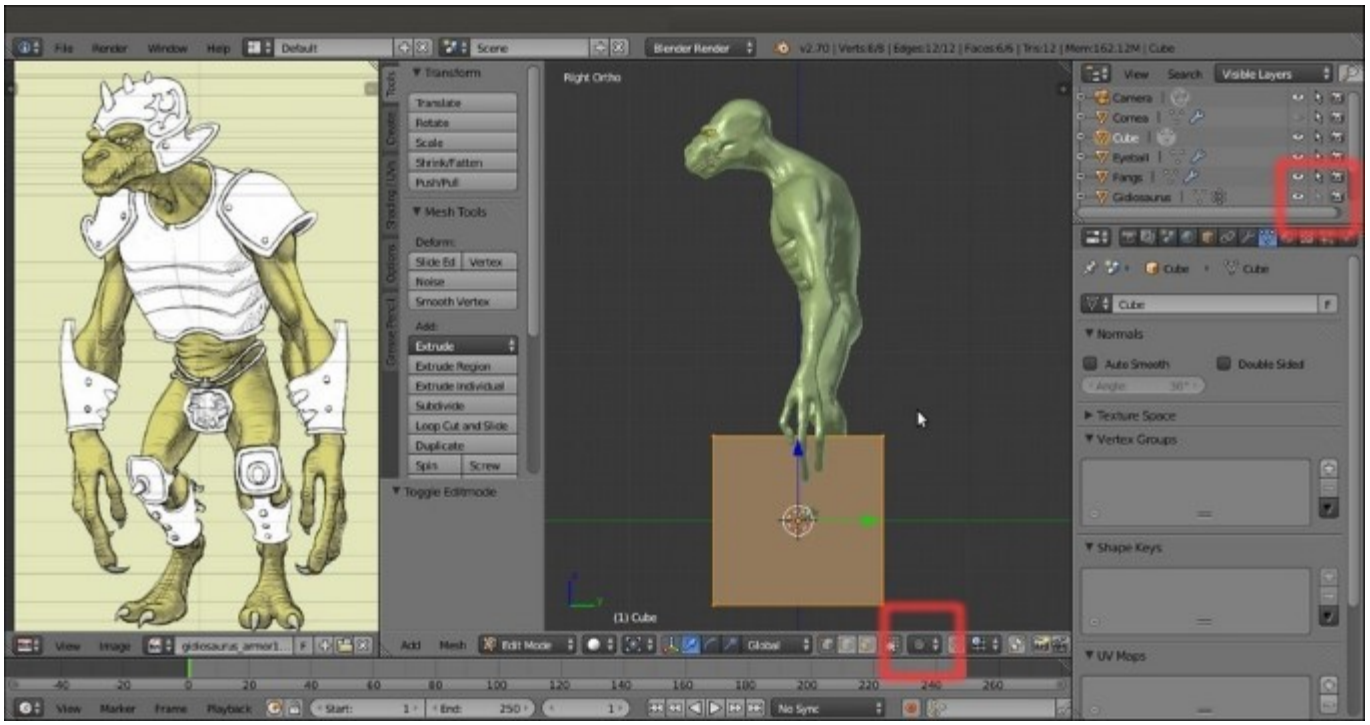
We will carry on with the `Gidiosaurus_modeling.blend` file:

1. Press *3* on the numpad to go in **Side** view.
2. By scrolling the middle mouse wheel, zoom back to frame the **Gidiosaurus** mesh in the 3D window.
3. In the **Outliner**, click on the arrow icon on the right-hand side of the **Gidiosaurus** item to make it unselectable.

How to do it…

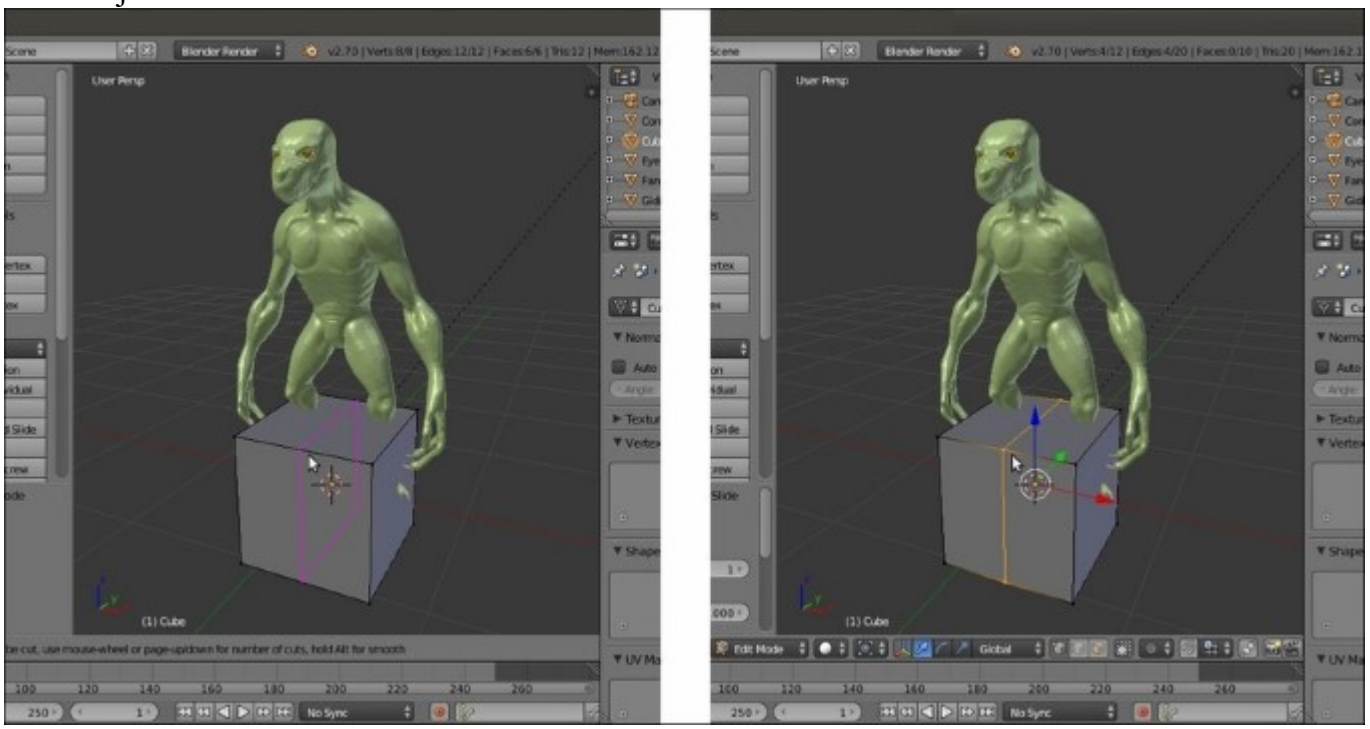
Now, we can start to build the **armor**; let's go with the **chest** piece:

1. Note that the **3D Cursor** is in the middle of the scene, at the character's pivot location (*Shift* + *S* | *Cursor to Selected* or also *Cursor to Active*, just in case).
2. Press *O* to disable the **Proportional Editing** tool; go to the 3D viewport toolbar to verify that the tool button is grayed.
3. Press *Shift* + *A* and add a **Cube** primitive to the scene.
4. Press *Tab* to go in **Edit Mode** and scale all the vertices to **0.500** (or press *S* | **.5** | *Enter*).



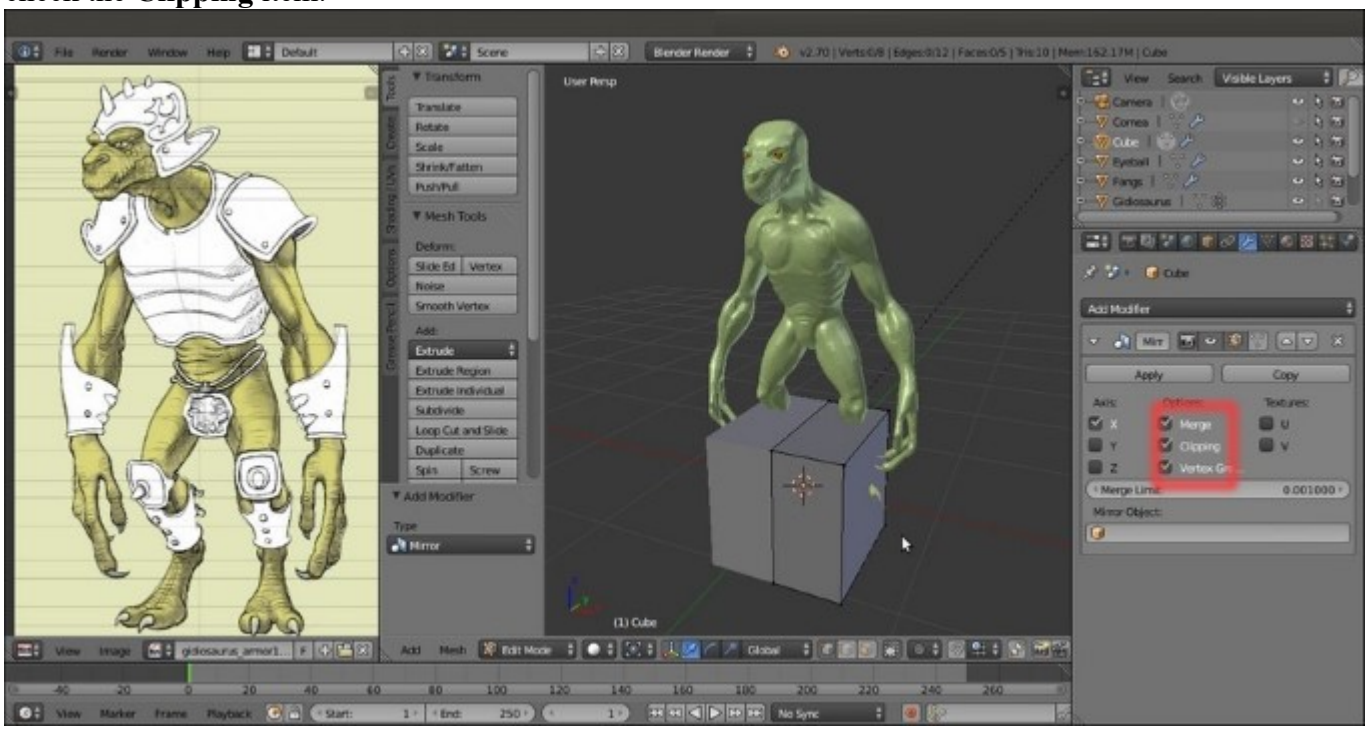
Adding the Cube primitive to the scene

5. Press *Ctrl* + *R* to add a loop along the *y* axis and then left-click twice to confirm it at the middle of the object:



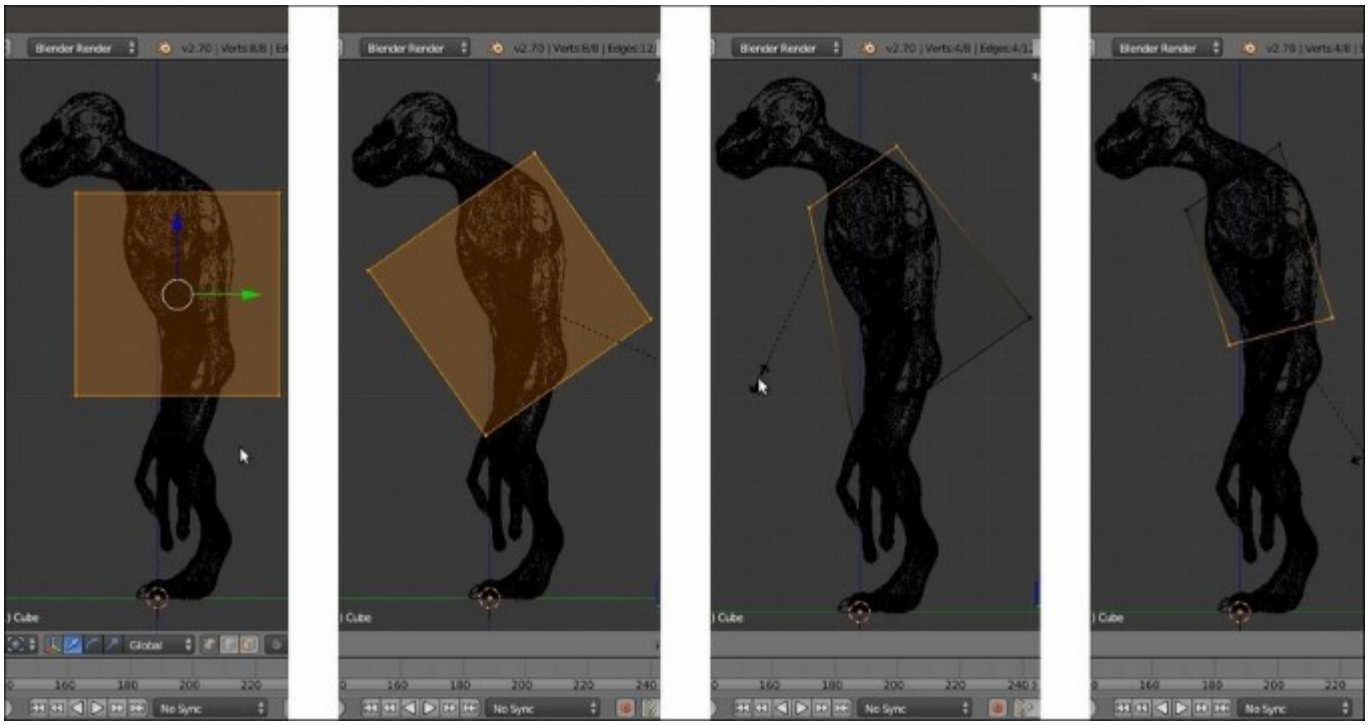
Adding a central vertical edge-loop to the Cube

6. Select the right-side vertices of the **Cube** and delete them; then, assign a **Mirror** modifier and check the **Clipping** item:



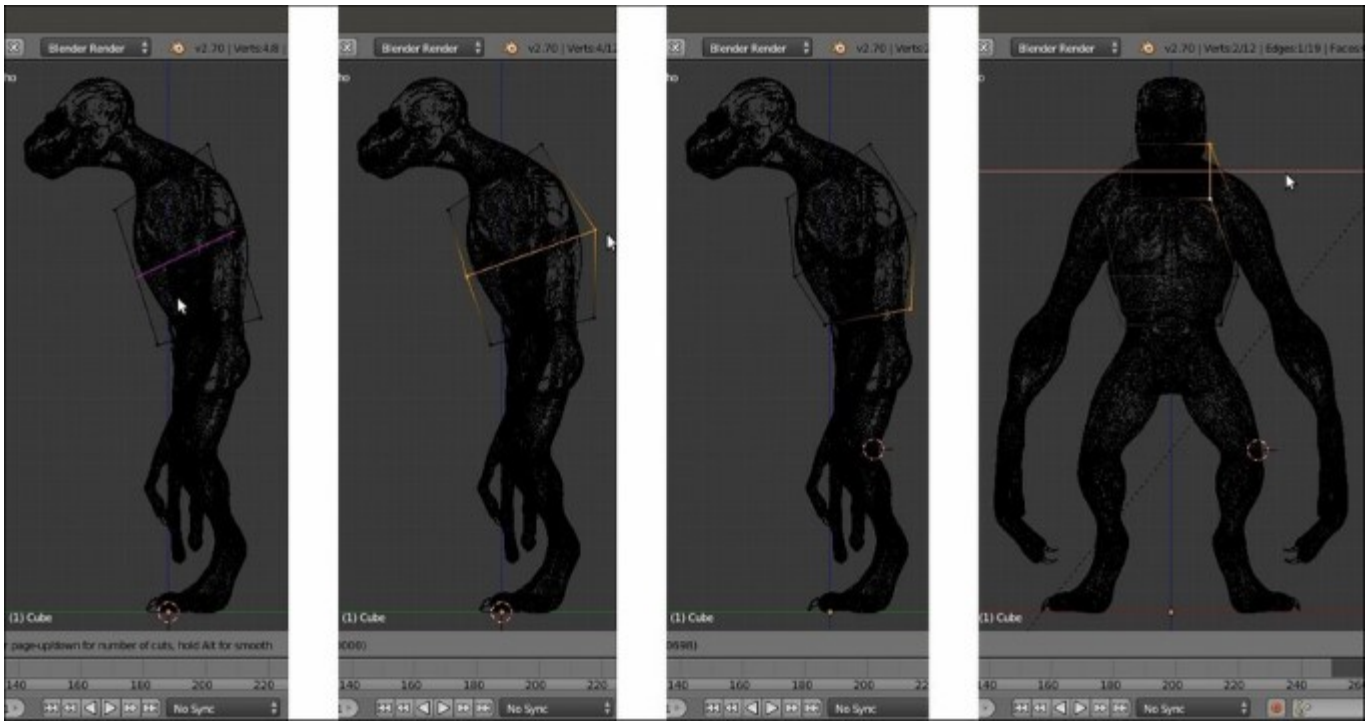
The Cube with the Mirror modifier

7. Go again in **Side** view and press *Z* to go in the **Wireframe** viewport shading mode; select all the vertices and move them upward.
8. Rotate the vertices to reflect the angle of the character's **chest**.
9. Select the upper vertices and scale and rotate them to fit the creature's **neck** area.
10. Select the bottom vertices and scale and rotate them to fit the base of the **chest**:



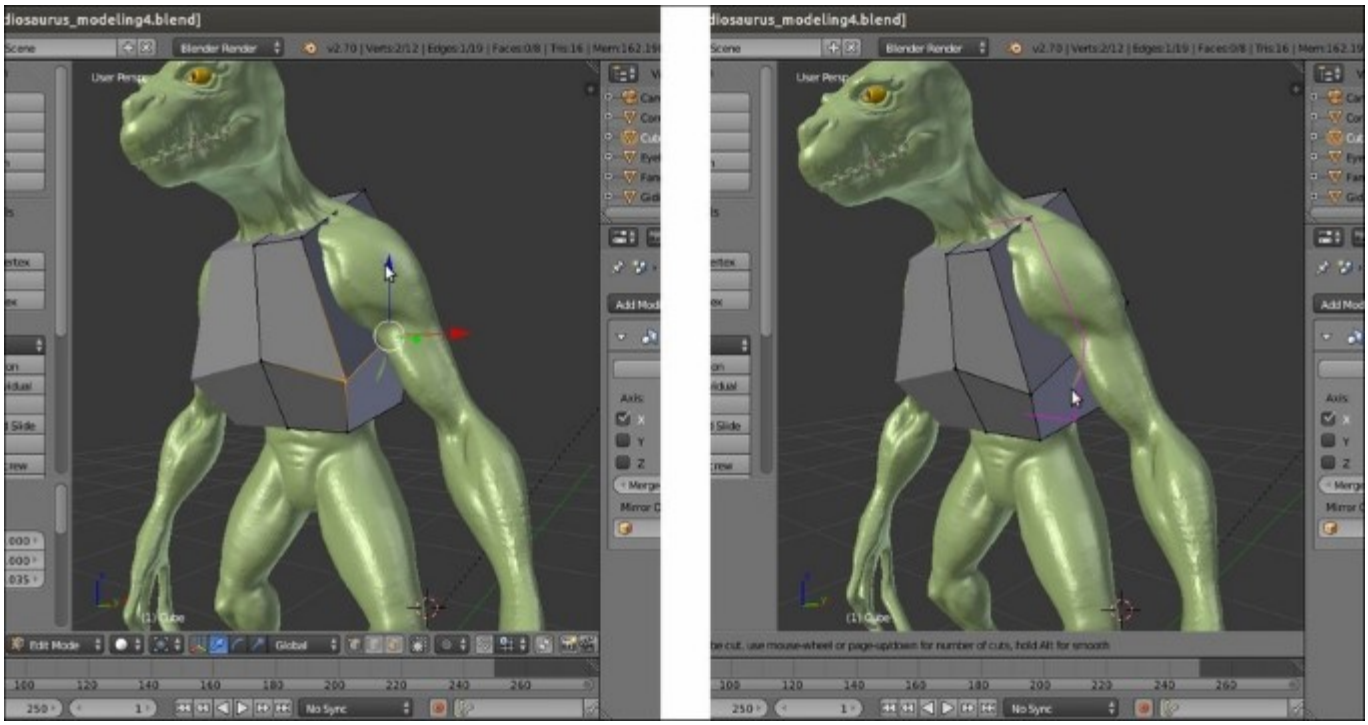
Starting to model the armor from the Cube primitive

11. Press *Ctrl* + *R* to add a new horizontal edge-loop at the middle of the **Cube**; scale it bigger to fit the shape of the creature's **chest**.
12. While still in **Side** view, grab and move the vertices to conform them to the **chest** shape.
13. Press *1* to go in **Front** view and again move the vertices to adjust them consistently to the character's **chest** shape:



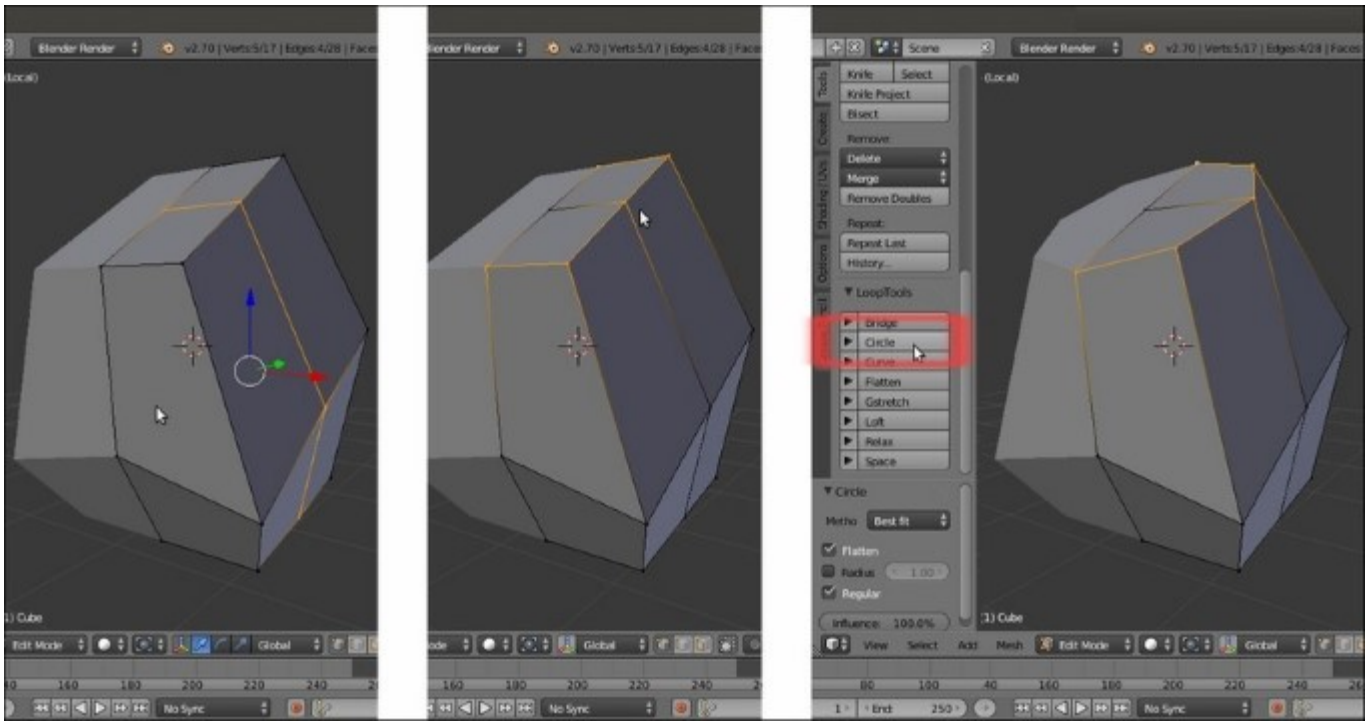
Adding more geometry and shape to the Cube

14. Select the **2** middle outer vertices and move them down, in order to place the edge connecting them just below the character's **armpit**.
15. Press *Ctrl* + *R* to add a loop along the *x* axis; click twice to confirm it at the middle of the lateral side:



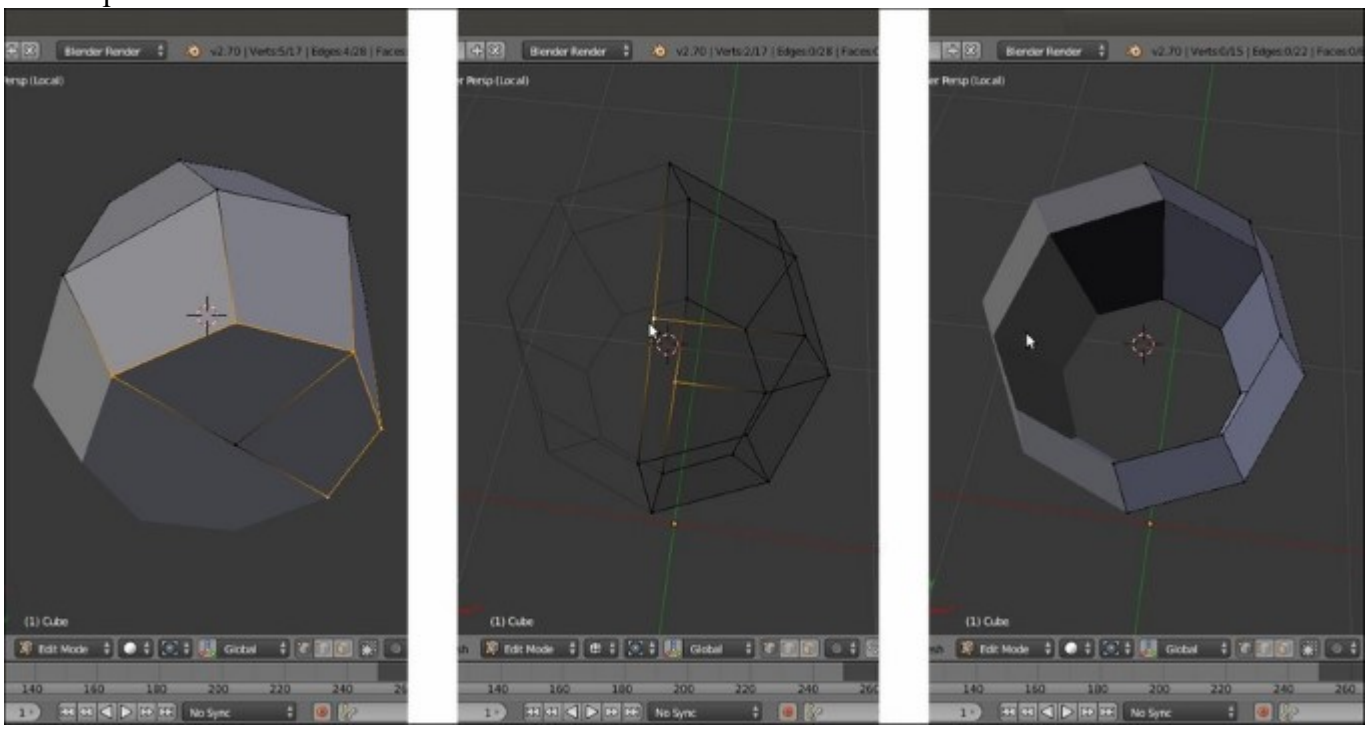
Adding more geometry again

16. Press the slash key (/) on the numpad to go in **Local** view with the selected object (in this case, even if still in **Edit Mode**, it is the **Cube**) and select the upper outer edge-loop.
17. Go to the **Tool Shelf** panel and scroll down the **Tools** tab to find the **LoopTools** subpanel (the **LoopTools** items are available also in the **Specials** menu that we can call by pressing the *W* key in **Edit Mode**); click on the **Circle** button to make the selection on a circular path:



Using the LoopTools add-on

18. Do the same also with the middle and the bottom edge-loop; then, select the central upper and bottom pole's vertices and delete them:

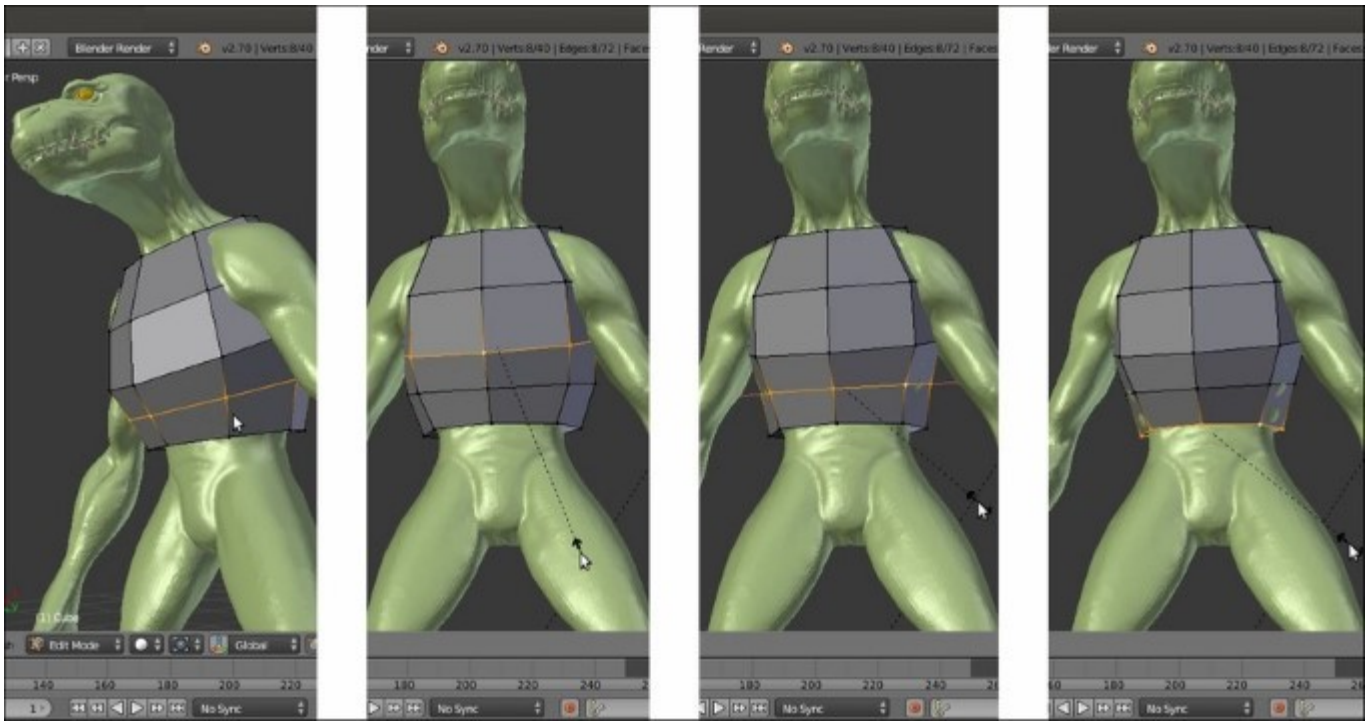


Going on with the modeling

19. Press the slash key (/) on the numpad to go out of **Local** view.
20. Press *Tab* to go out of **Edit Mode** and go to the **Object Modifiers** window under the main **Properties** panel; click on the **Apply** button to apply the **Mirror** modifier.
21. Go back in **Edit Mode** and press *Ctrl* + *R* to add a horizontal edge-loop to the upper half of the mesh.
22. Scale the new edge-loop to **1.100**:

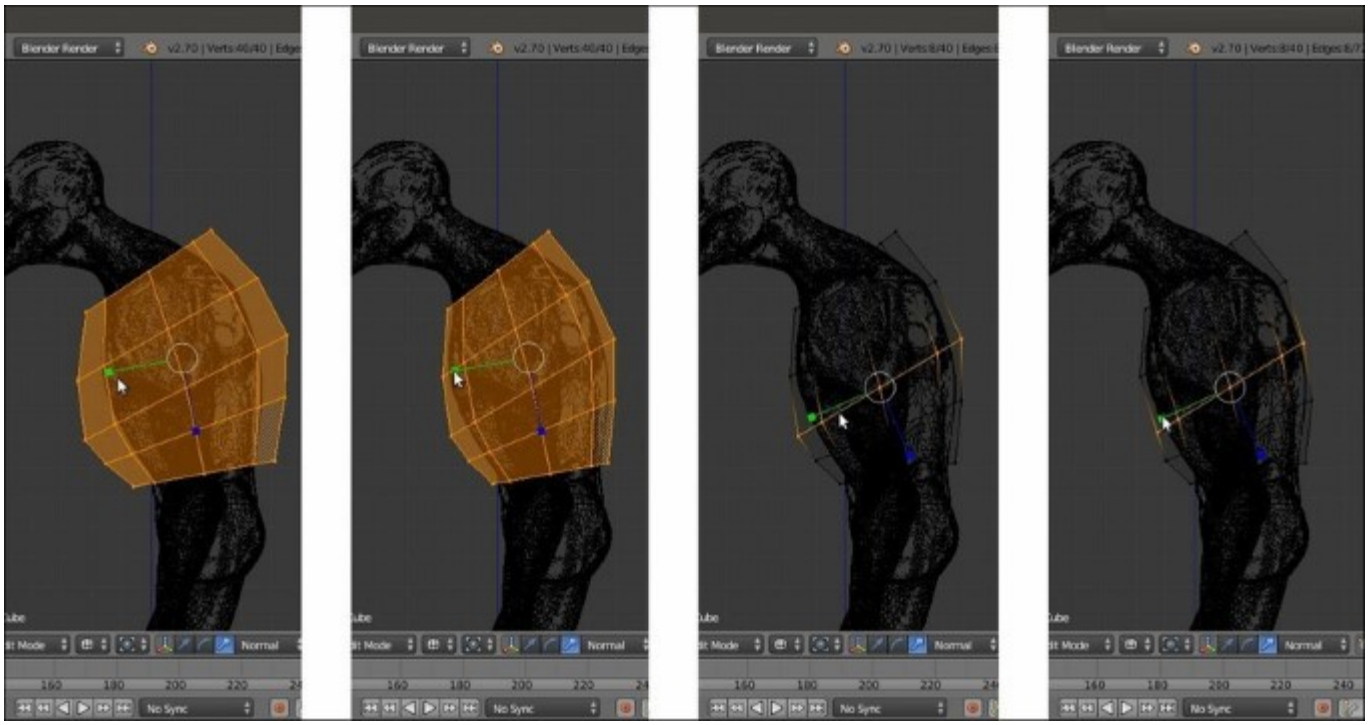
Adapting the shape of the armor to the chest by adding more geometry as edge-loops

23. Add a new horizontal edge-loop also to the lower half of the mesh.
24. Select the middle edge-loop and scale it smaller on the *x* axis, to **0.900**.
25. Select the bottom edge-loop and scale it smaller on the *x* axis as well.
26. Select the last edge-loop and repeat the operation.



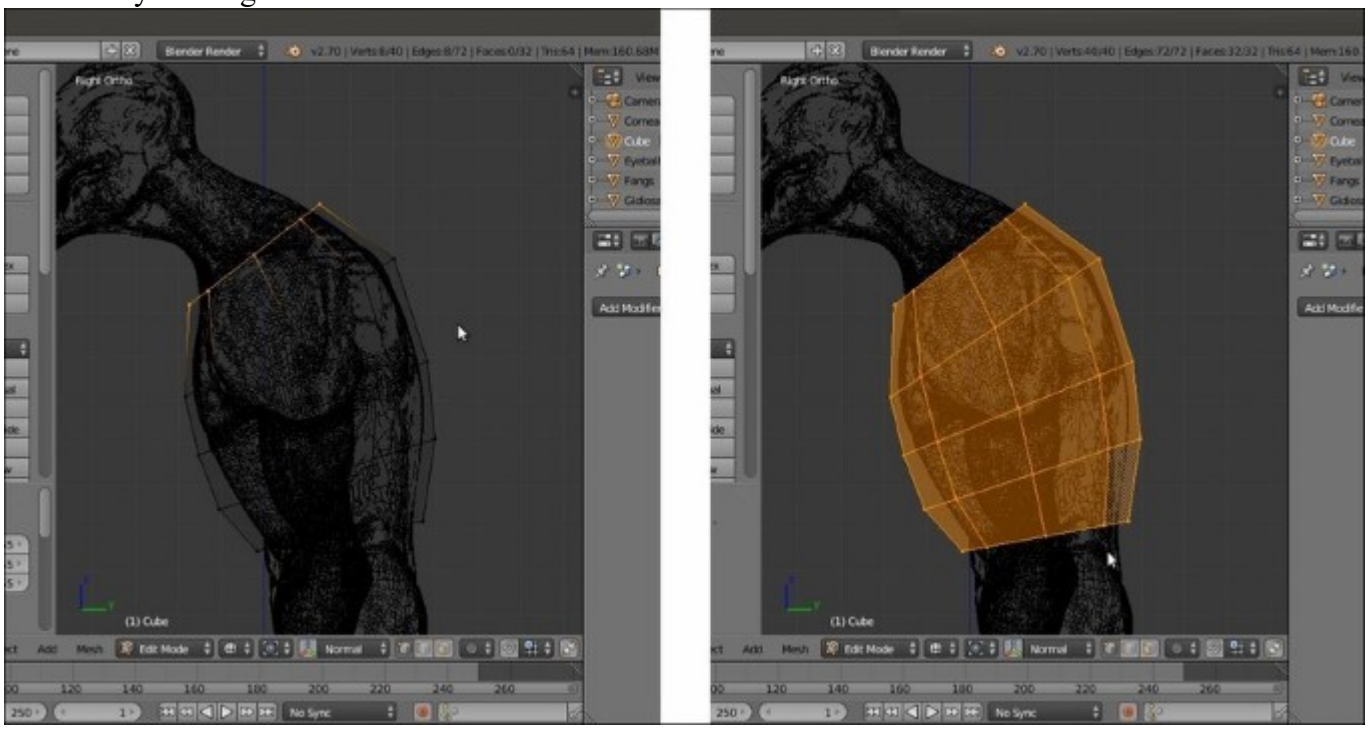
Going on with the modeling by adding edge-loops

27. Press *3* on the numpad to go in **Side** view and *Z* to go in the **Wireframe** viewport shading mode.
28. If not already, enable the widget in the toolbar of the 3D window; set the **Transformation manipulators** to **scaling** (the last icon to the right) and the **Transform Orientation** option to **Normal**.
29. Select all the vertices and by moving the green scaling manipulator of the widget, scale smaller all the edge-loops on the normal *y* axis; small enough to almost reach the character's **back** and **chest** surfaces.
30. Deselect everything and then select the middle edge-loop (press *Alt* + right-click); scale it again by using the widget to get close to the **torso** shape:



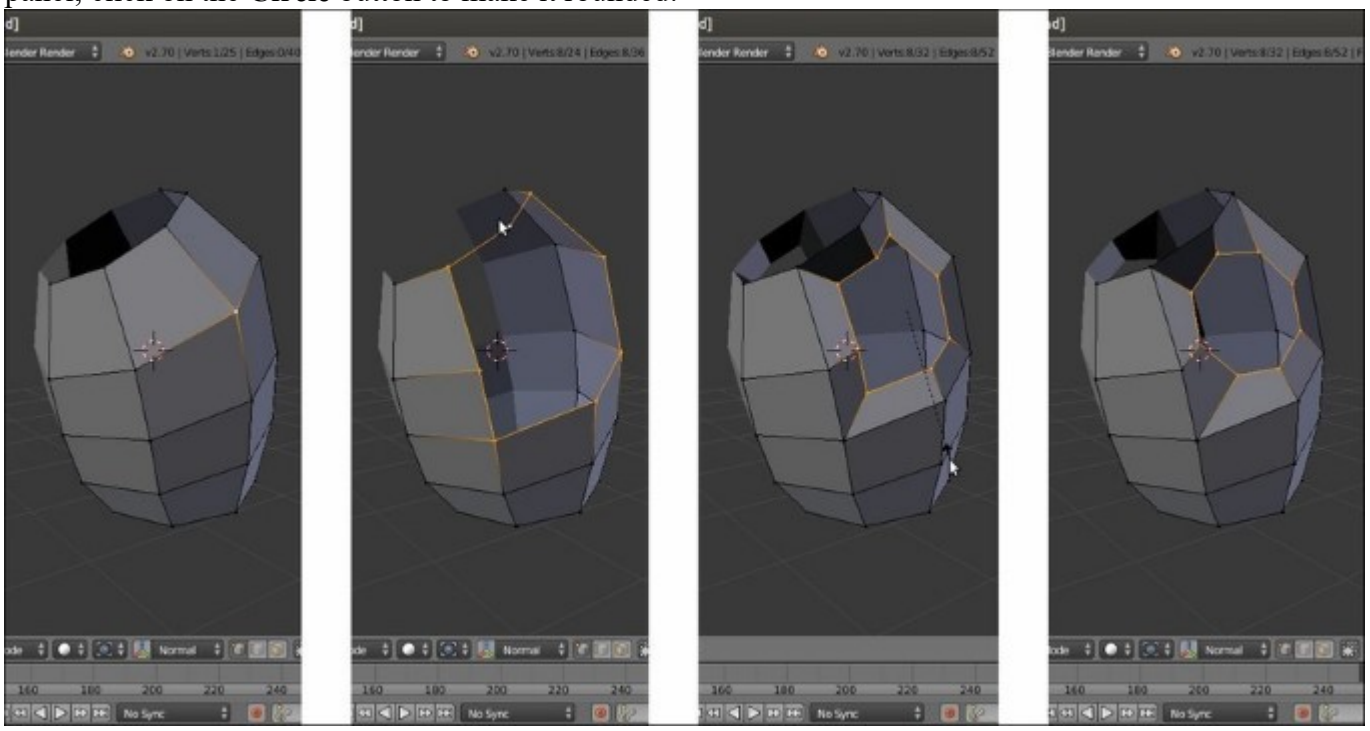
Adjusting the chest armor depth

31. Do the same with the other edge-loops by selecting them individually, rotating and scaling them, and also by moving the vertices.



Refining the lateral profile of the armor

32. Press the slash (*/*) key on the numpad to go again in **Local** view.
33. Select the right-side vertices of the **Cube** and delete them.
34. Go to the **Object Modifiers** panel and assign a new **Mirror** modifier; as usual, check the **Clipping** item.
35. Select the central vertex on the upper-side part and delete it.
36. Select the resulting loop of edges around the resulting hole (you can press *Alt* + right-click and then *Shift* + right-click to add the remaining unselected top vertex to the selection; it doesn't get selected with the edge-loop because there are no faces connecting it to the other vertices, but only edges).
37. Press *E* and then *S* to extrude new faces and scale them (about **0.600**).
38. Select the new edge-loop and go to the **LoopTools** panel under the **Tools** tab of the **Tool Shelf** panel; click on the **Circle** button to make it rounded:

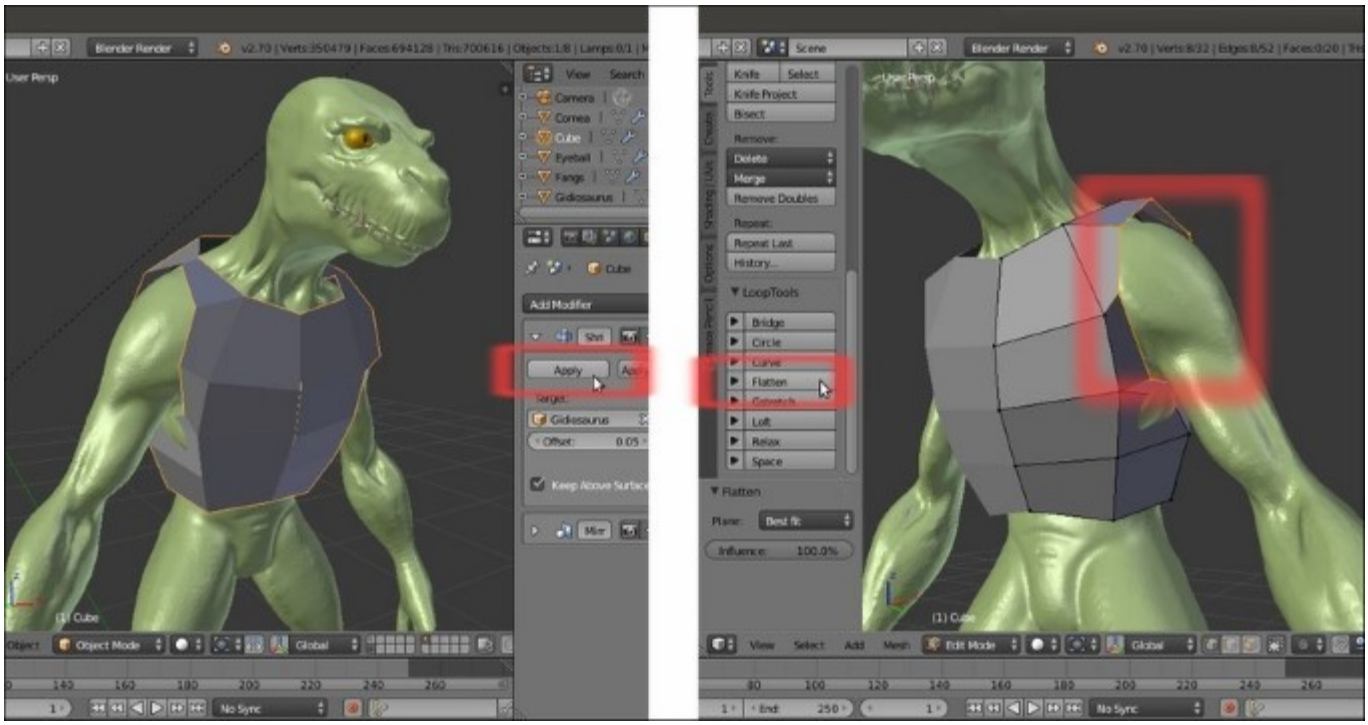


Adding the arm's holes

39. Go in **Front** view and move the edge-loop outward.
40. Go out of **Edit Mode** and press the slash (*/*) key on the numpad to go out of **Local** view.
41. Press *1* on the numpad to go again in **Front** view. Go to the **Object Modifiers** panel and assign a **Shrinkwrap** modifier to the **Cube**; check the **Keep Above Surface** item and in the **Target** field, select the **Gidiosaurus** name:

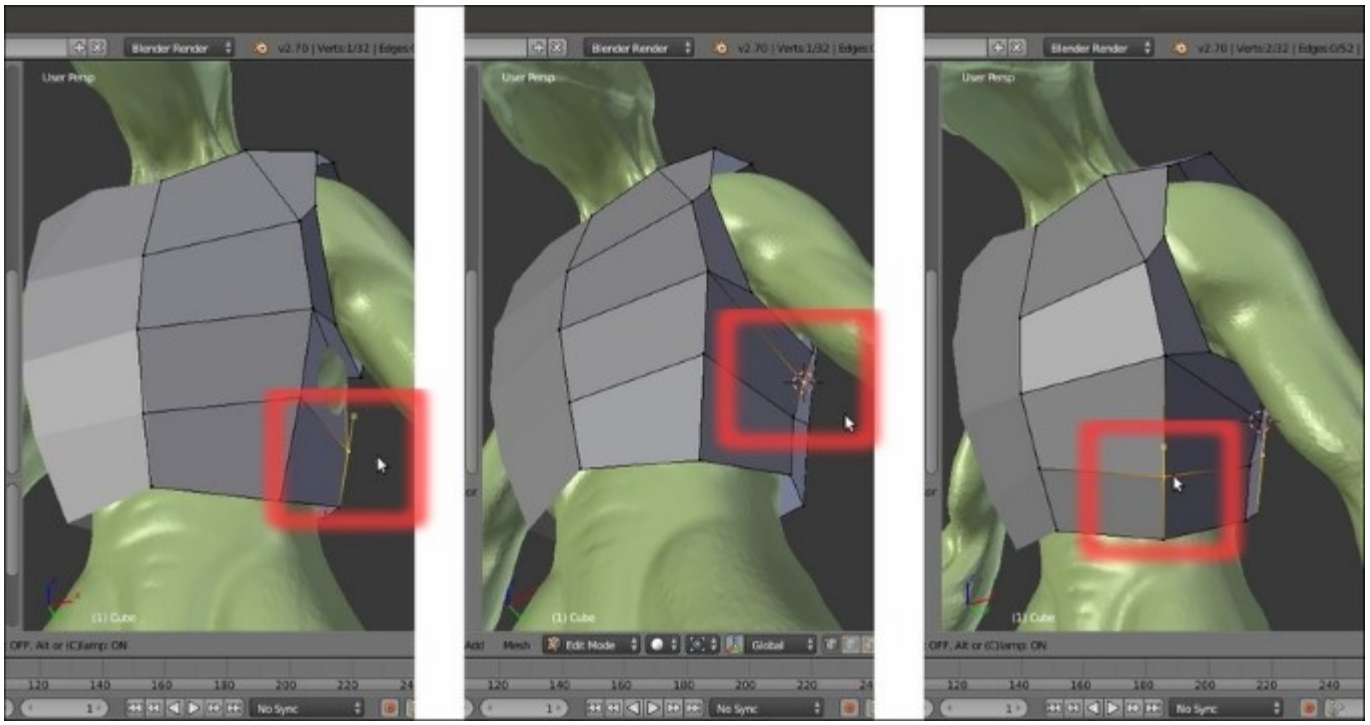
Assigning the Shrinkwrap modifier to the chest armor

42. Set the **Offset** value to **0.05**.
43. Move the **Shrinkwrap** modifier to the top of the modifier stack and click on the **Apply** button.
44. Go in **Edit Mode** and select the **shoulder** edge-loop; go to the **LoopTools** panel and click on the **Flatten** button:



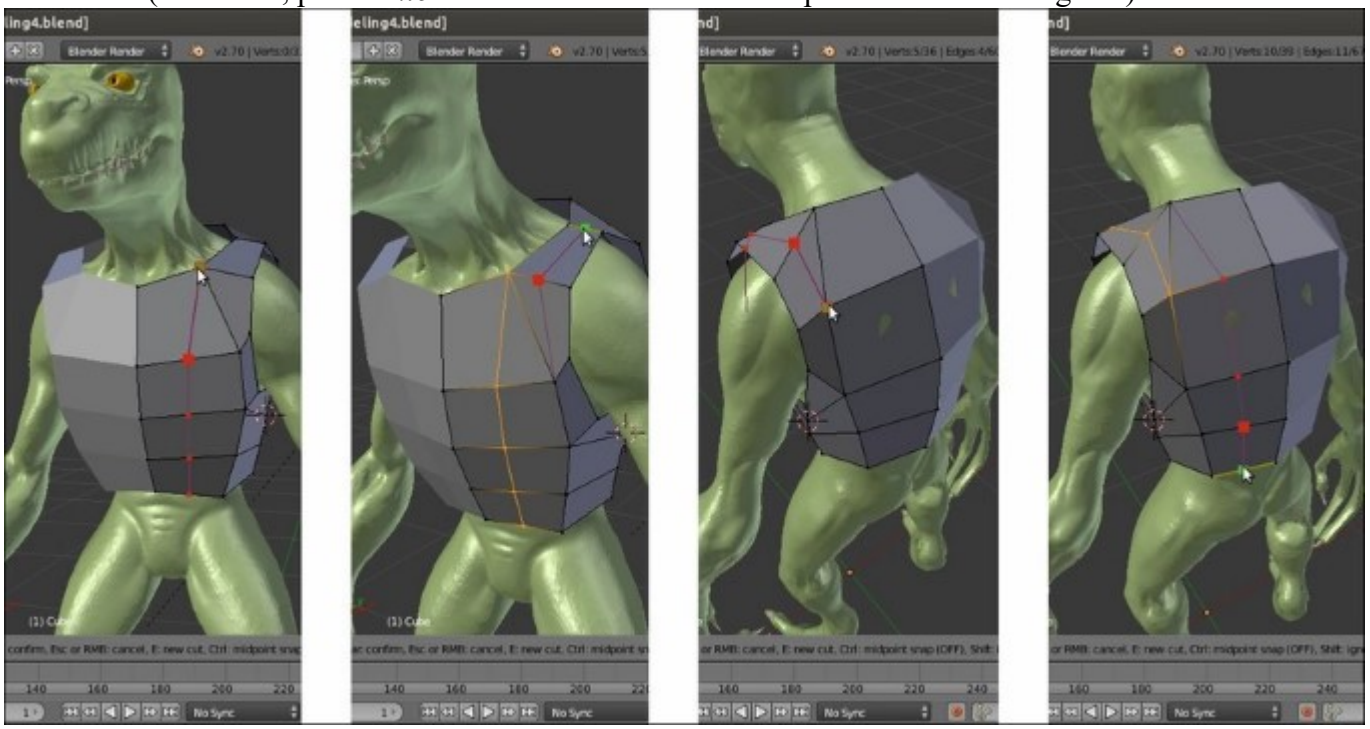
Refining the modeling through the LoopTools add-on

45. Fix, below the **armpit**, the lateral vertices that are curved inwards, by using the *Alt* + *S* shortcut to move them outward along their normal, the **3D Cursor** and the **Snap** pop-up menu (*Shift* + *S*) to place them midway from other vertices, and the *Shift* + *V* shortcut to slide them along the edges:



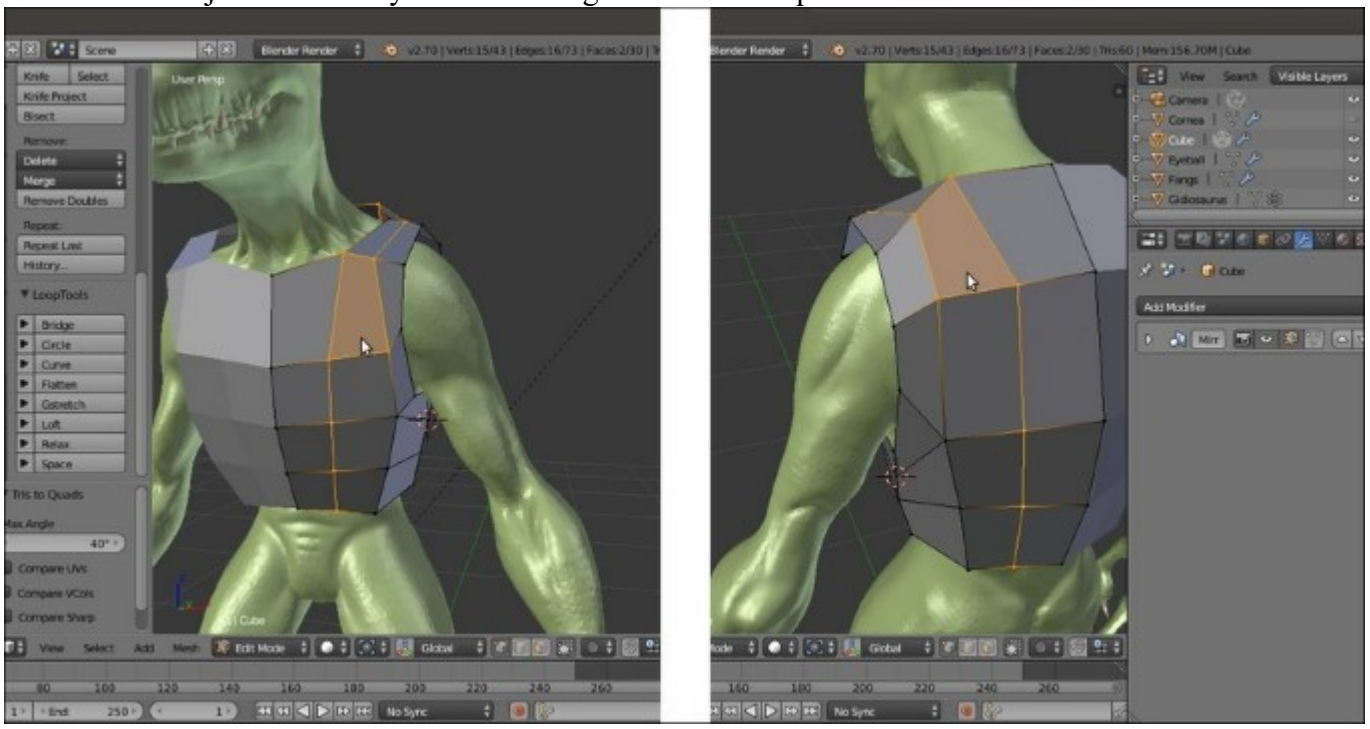
Tweaking vertices

46. Press the *K* key to activate the **Knife Topology Tool**; by keeping the *Ctrl* key pressed to constrain the cuts to the middle of the edges, cut a new edge-loop as shown in the following screenshot (each time, press *Enter* to confirm the cut and then pass to the following one):



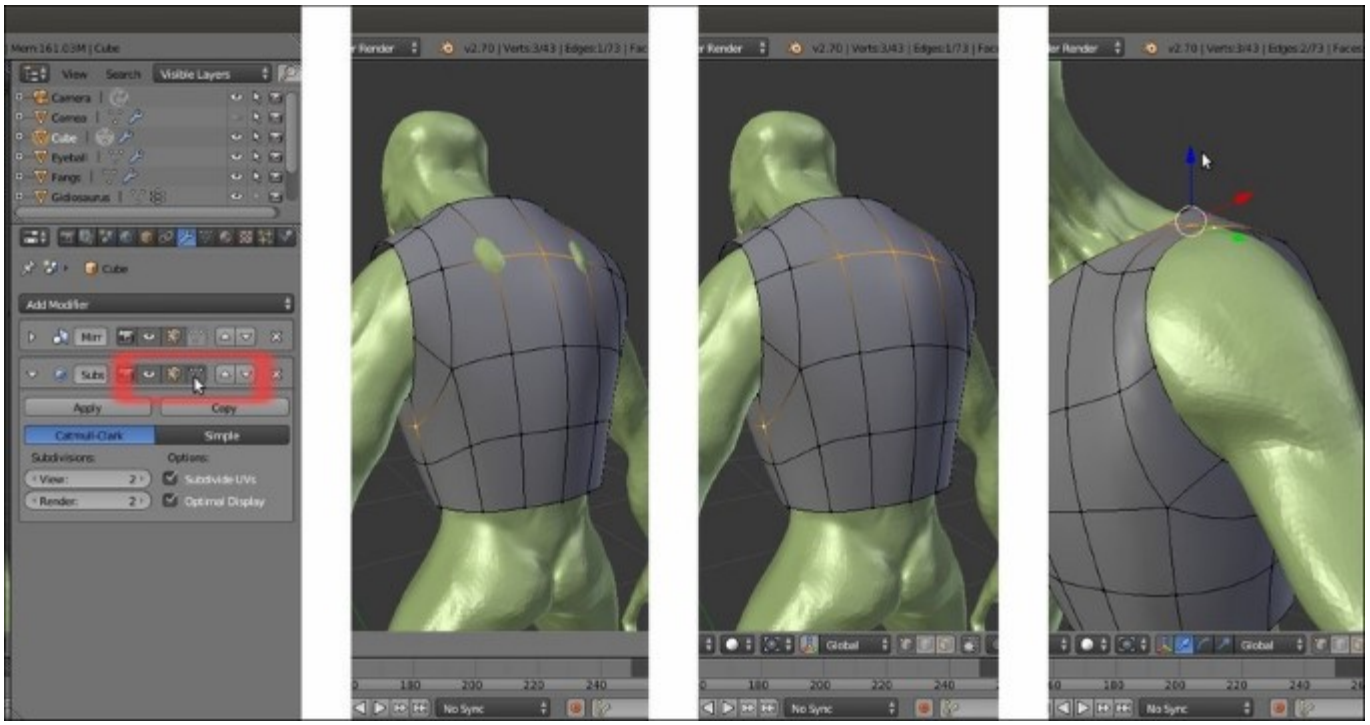
Using the Knife tool

47. Press *Alt* + *J* to join the already selected triangular faces into quads:



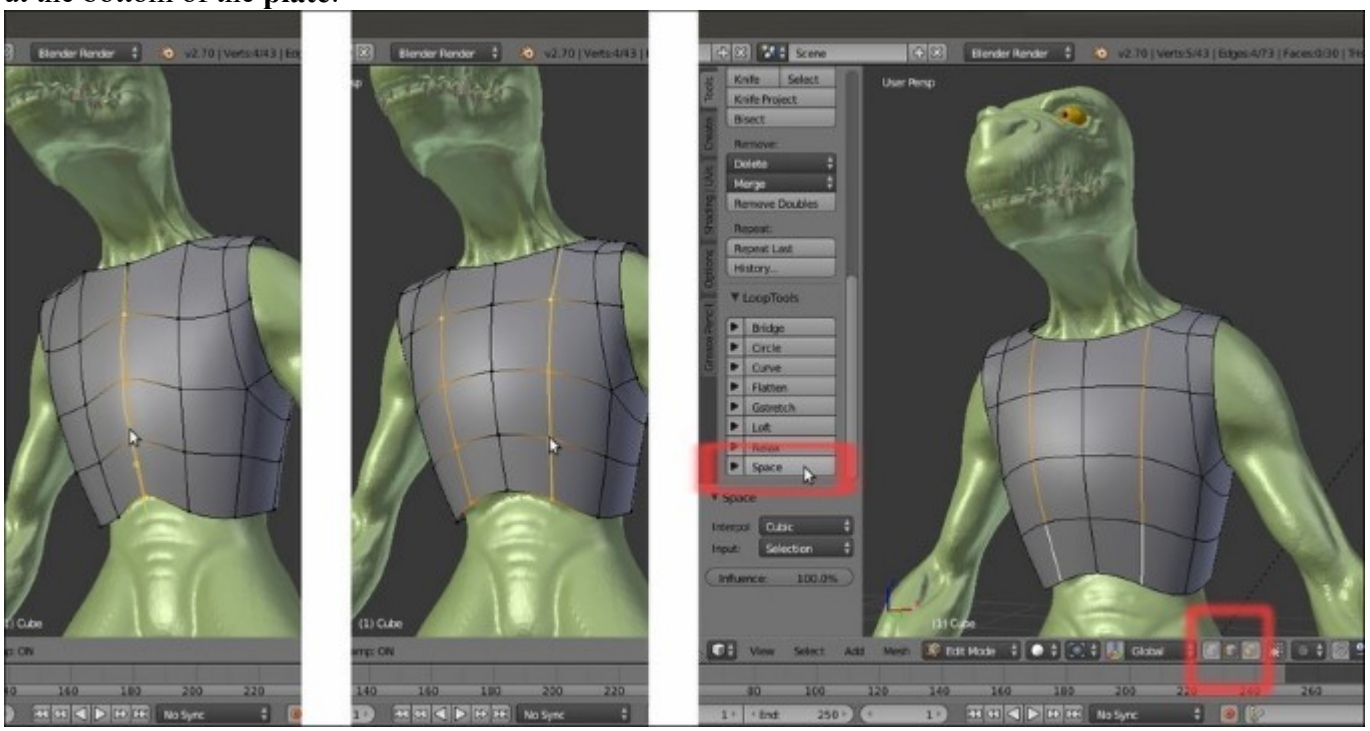
Joining two triangular faces into one quad face

48. Select all the vertices and press *Ctrl* + *N* to recalculate the normals.
49. Deselect all the vertices and go to the **Object Modifiers** window; assign a **Subdivision Surface** modifier, set the **Subdivisions** level for **View** to **2**, and check the **Optimal Display** item. Click on the *Adjust edit cage to modifier result* icon, the last one to the right with the editing triangle, in order to see the effect of the modifier in **Edit Mode**.
50. Go out of **Edit Mode** and then go to the **Tools** tab under the **Tool Shelf**; under the **Edit** subpanel, select the **Smooth** shading.
51. Go back in **Edit Mode** and select the vertices (in our case, mainly on the side and back) corresponding to areas where the sculpted mesh is overlapping the **armor**. Press *Alt* + *S* to scale their position along their normals and so fix the overlapping; then, select the upper vertices of the **shoulder** and move them closer to the character's **shoulder** surface:



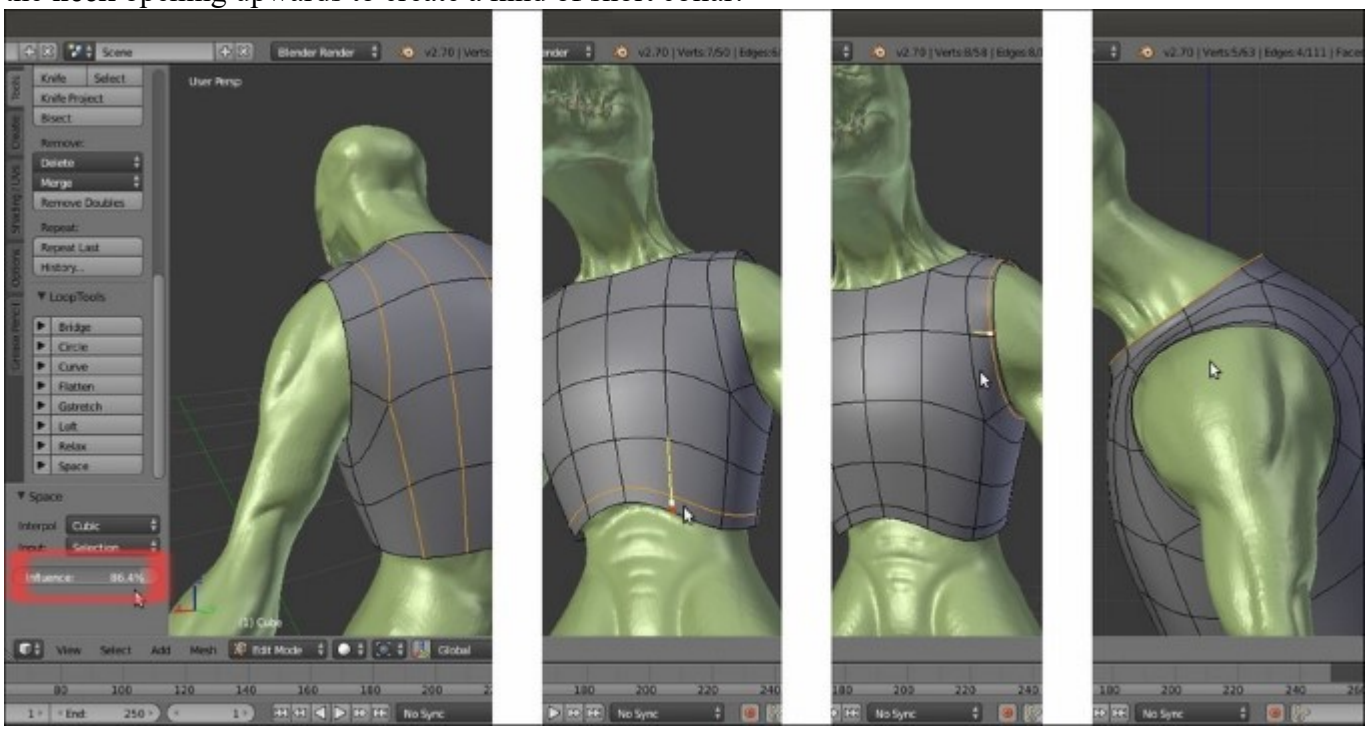
Tweaking vertices with the visible cage of the Subdivision Surface modifier

52. Repeat the operations of the previous step on all the vertices that need it; select the vertices on the **belly** and press *Shift + V* to move them upward, but along the edges to model the arc shape at the bottom of the **plate**:



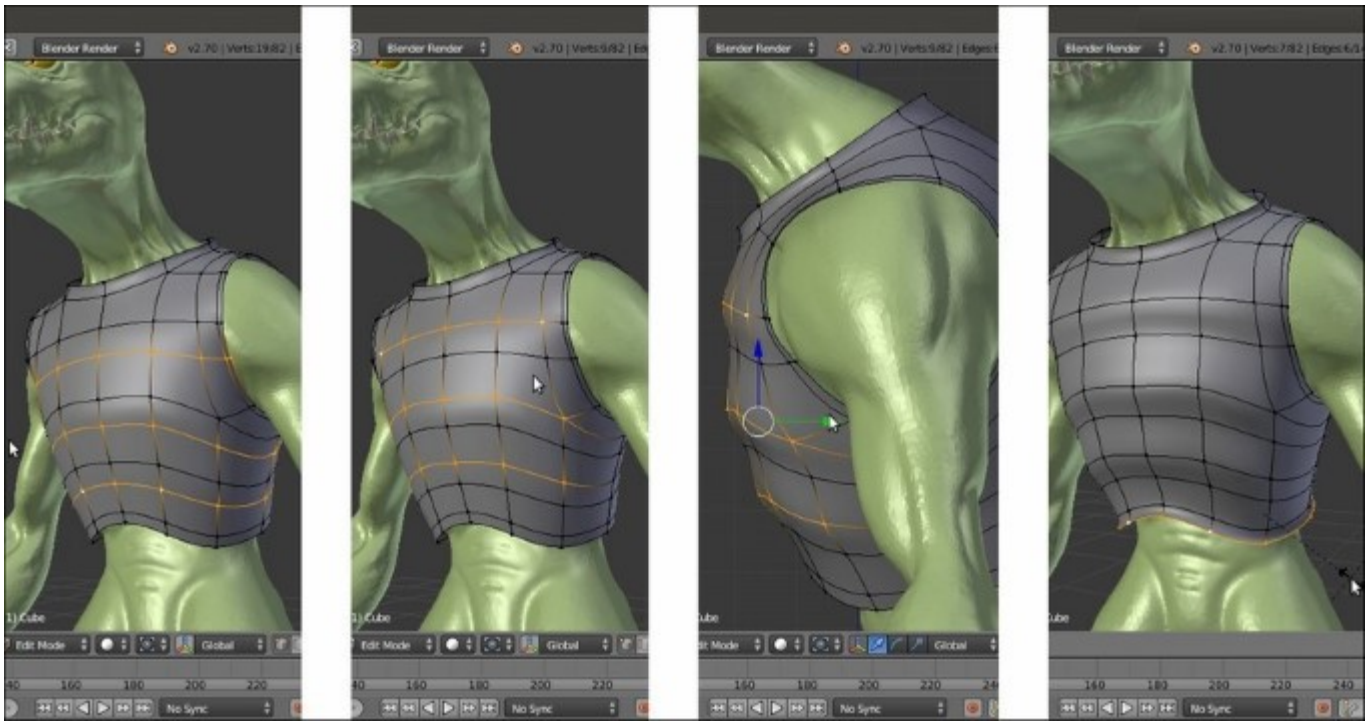
Sliding the vertices and adjusting the polygonal flow through the LoopTools add-on

53. Select the edges of the front and back and click on the **Space** button in the **LoopTools** add-on panel; if needed, tweak the value of the **Influence** slider at the bottom of the **Tools** tab to set the amount for the operation.
54. Add edge-loops at the bottom of the **armor** and at the **shoulder** opening to create a rim; extrude the **neck** opening upwards to create a kind of short collar:



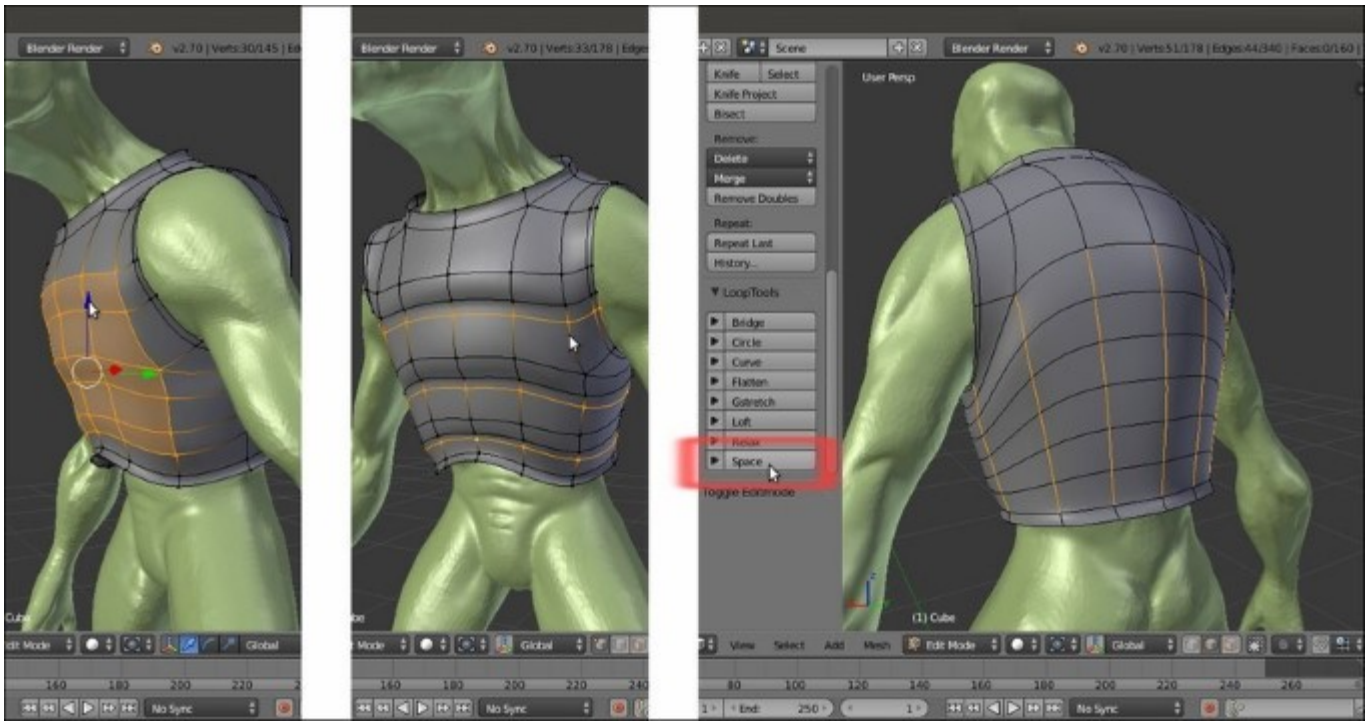
Extruding geometry

55. Add edge-loops on the front of the **chest** plate as shown in the following screenshot (*Ctrl* + *R* and then slide it to **0.500**) and then select the front vertices of the alternate edge-loops and in **Side** view, move them forward.
56. Select the last bottom edge-loop and scale it bigger (to **1.100**):



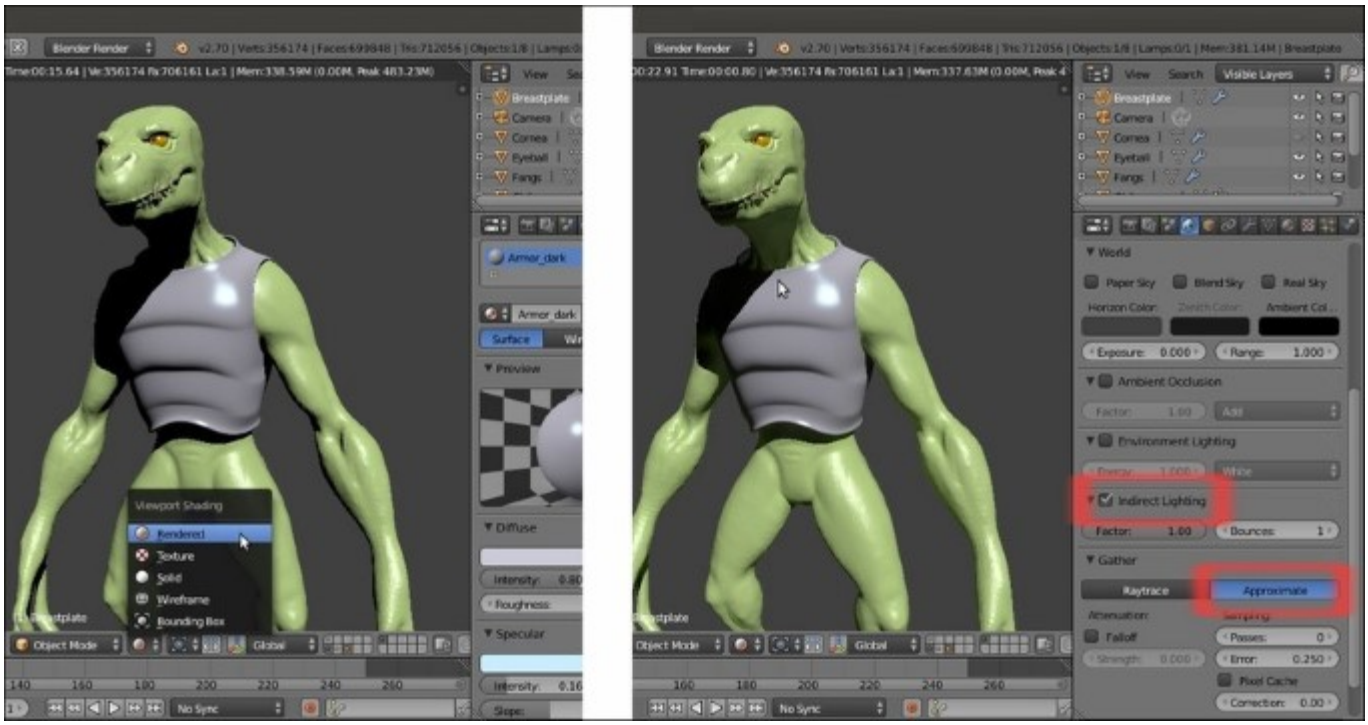
Adding edge-loops to add detailing to the armor

57. Move the front vertices of the **breast** and **belly** downward, using the image loaded in **UV/ Image Editor** as reference. Add more edge-loops to add definition to the front of the **chest** plate (in the following screenshot, the three added edge-loops are selected at the same time only to highlight them; in Blender, they must be added one at a time). Then, smooth the resulting oddly spaced back vertices by using the **Space** button of the **LoopTools** add-on:



Making the edges' length even through the LoopTools add-on

58. In the **Outliner**, rename the **Cube** item as **Breastplate** (by either double-left-clicking or by pressing *Ctrl* + left-click on the item).
59. Then, go to the **Material** window under the main **Properties** panel and assign a new material to the **Breastplate** object; rename the material as `Armor_dark`. Set the diffuse color to **RGB 0.605**, **0.596**, **0.686** and the **Diffuse Shader Model** option to **Oren-Nayar**; set the specular color to **RGB 0.599**, **0.857**, **1.000** and the **Specular Shader Model** option to **WardIso**; set the **Intensity** value to **0.164** and the **Slope** value to **0.100**. Under the **Shading** subpanel, check the **Cubic Interpolation** item.
60. Go to the **Object Modifiers** window and assign a **Solidify** modifier; move it up in the stack, before the **Subdivision Surface** modifier. Set the **Thickness** value to **0.0150** and check the **Even Thickness** item.
61. Set **Viewport Shading** to **Rendered** to have a quick preview (be sure to have the proper scene layers activated, that is, the **6th** for the lighting and the **11th** and **13th** for the **character** and **armor**). Then, go to the **World** window under the main **Properties** panel and activate the **Indirect Lighting** tab; then, click on the **Approximate** button under the **Gather** subpanel. For the moment, leave the rest as it is:



The Rendered result so far, with some World Lighting setting

62. Save the file.

How it works…

This is the usual polygonal modeling process that is common to most aspects of 3D packages. Starting from a **Cube** primitive, we moved and arranged the vertices to model the **chest** armor plate, extruding and also adding new edge-loops by using the **Knife Topology Tool** and the *Ctrl* + *R* shortcut.

We used the **Mirror** modifier to work only on half of the mesh and to have the other half automatically updated. In some cases, we had to temporarily apply the **Mirror** modifier to better scale the edges as complete circles (otherwise, they would have been half circles with odd scaling pivot points); then, we had to delete the vertices from one side and assign the **Mirror** modifier again.

At a certain point, as the **armor's** shape got more defined, we started to tweak the vertices in **Edit Mode**, but with the **Subdivision Surface** modifier applied to the editing cage in order to have the right feedback while conforming the **armor's** shape to the **character's** shape.

We also used a few of the options available in the **LoopTools** add-on that has been revealed to be an incredibly handy aid in the modeling process.

See also

- https://sites.google.com/site/bartiuscrouch/looptools
- http://www.blender.org/manual/modeling/index.html

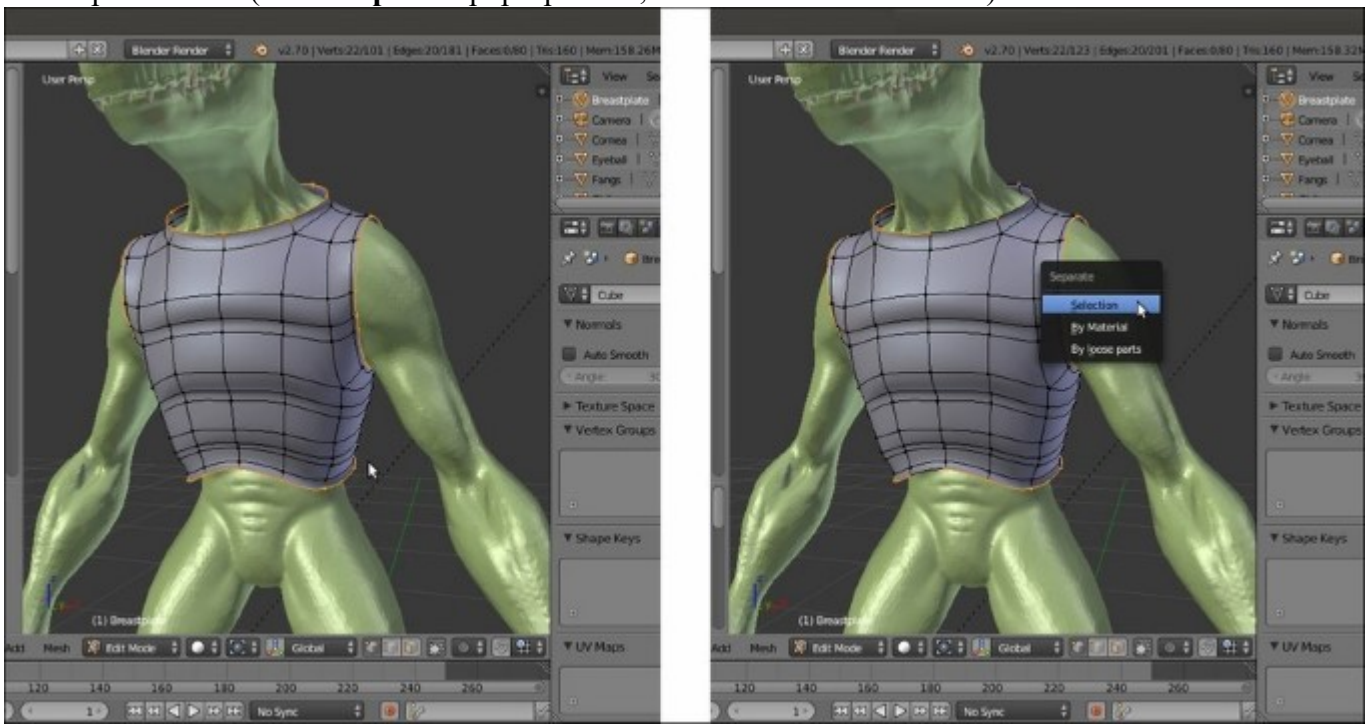
Using the Mesh to Curve technique to add details

In the previous recipe, we modeled the basic bulk of the **Breastplate**. We are now going to see a simple but effective technique to add detailing to the borders of the **armor** plate.

How to do it…

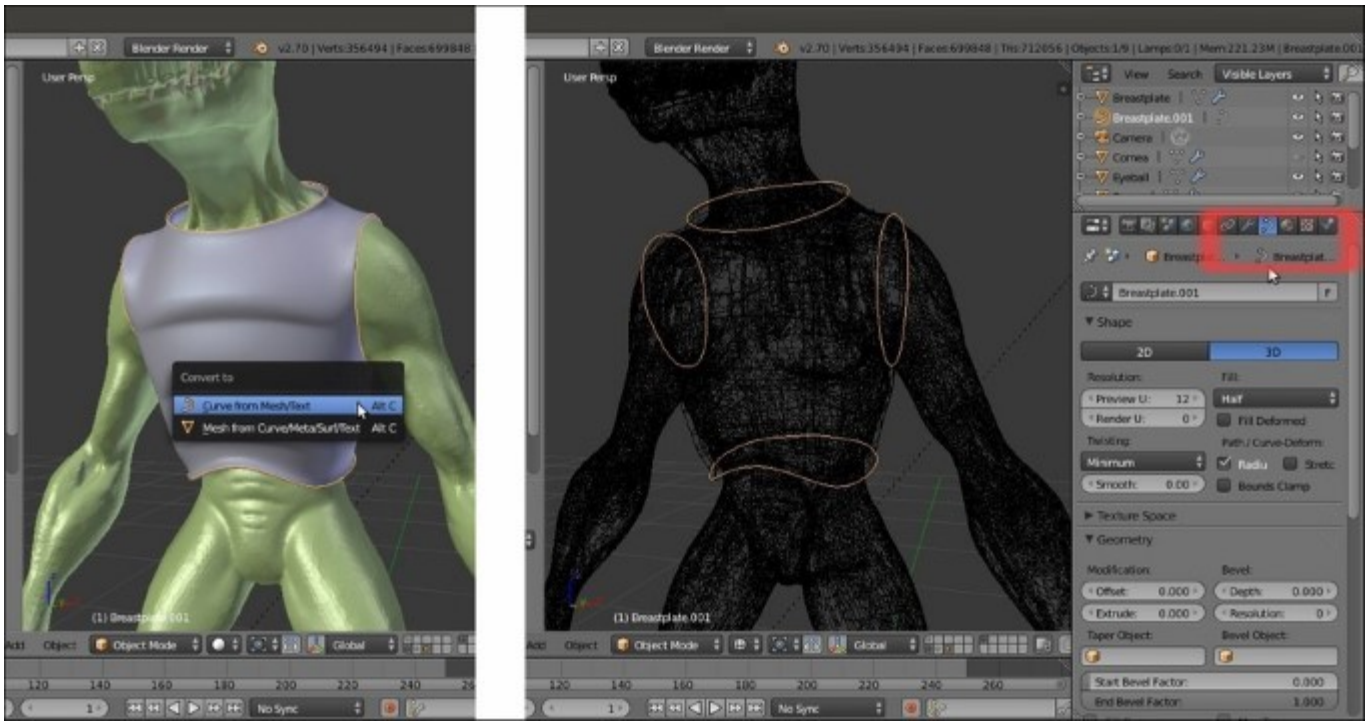
Assuming we have gone out of **Edit Mode** and then saved the file, reopen the `Gidiosaurus_modeling.blend` file and proceed with the following:

1. Go back in **Edit Mode** and select the edge-loop around the **neck** (*Alt* + right-click), the edge-loop around the **shoulder** hole (*Alt* + *Shift* + right-click), and the last one at the base of the **Breastplate** (*Alt* + *Shift* + right-click again).
2. Press *Shift* + *D* and soon after, the right-mouse button to duplicate without moving them; press *P* to separate them (in the **Separate** pop-up menu, choose the **Selection** item):



Separating geometry by selection

3. Go out of **Edit Mode** to select the **Breastplate.001** object (the duplicated edge-loops).
4. Press *Alt* + *C* and in the **Convert to** pop-up menu, select the first item: **Curve from Mesh/Text**.
5. The mesh edge-loops actually get converted into **Curve** objects, as you can see in the **Object Data** window under the main **Properties** panel on the right-hand side of the UI:



Converting the geometry in Curves

6. In the **Object Data** window, under the **Geometry** tab, set the **Extrude** value to **0.002** and the **Depth** value to **0.010**; then, under the **Shape** tab, set the **Fill** mode to **Full**:

"Modeling" the Curves by the settings

7. Press *Alt* + *C* and this time, in the **Convert to** pop-up menu, select the second item: **Mesh from Curve/Meta/Surf/Text**.
8. Press *Tab* to go in **Edit Mode**, press *A* to select all the vertices, and in the **Mesh Tools** tab under the **Tools** tab in the **Tool Shelf** panel, click on the **Remove Doubles** button (note that in the top main header, a message appears: **Removed 2240 vertices**; so always remember to remove the doubles after a conversion!).
9. Go out of **Edit Mode** and click on the **Smooth** button in the **Edit** subpanel; in the **Outliner**, rename it as **Breastplate_decorations**.
10. Assign a **Subdivision Surface** modifier, with the **Subdivision** level as **2** and **Optimal Display** enabled.
11. Go to the **Material** window and assign a new material; rename it as `Armor_light` and copy all the settings and options from the `Armor_dark` material, except for the diffuse and the specular colors—set them to **RGB 1.000** (pure white; a faster way is to assign the `Armor_dark` material, make it a single user, change the colors to white, and rename the material as `Armor_light`).

Assigning a new material

12. As always, remember to save the file.

How it works…

Even if at first sight this seems a complex process, actually it's one of the easiest and fastest ways to model a mesh. We have just duplicated the edge-loops that are located where we had the intention of adding the modeled borders. With a simple shortcut, we have converted them to a **curve** object that can

be beveled both by other curve objects or simply by values to be inserted in the fields under the **Geometry** tab. Then, once we obtained the shape we wanted, we converted the curve back to a mesh object.

We could have kept the armor decorations as **curves**, but by converting them to meshes, we have the opportunity to unwrap them for the mapping of the textures according to the rest of the **armor**.

Note that the **Preview U** value under the **Resolution** item in the **Shape** subpanel for the **curve** objects should be kept low if you don't want a resulting mesh with a lot of vertices; you can set it quite lower than the default **12**. Just experiment before the final conversion, while keeping in mind that once converted to mesh, the **decorations** will probably be smoothed by a **Subdivision Surface** modifier with the rest of the **armor**; in any case, the obtained **decorations** mesh can also be simplified at a successive stage.

In this chapter, we saw the process that can be used to model the **armor** meshes. We will not demonstrate the rest of the **armor** modeling, as the same techniques can be used over again. However, feel free to model the rest of the **armor** on your own or have a look at the provided `Gidiosaurus_modeling_02.blend` file:

The completed armor as it appears in the rendering

Chapter 4. Re-topology of the High Resolution Sculpted Character's Mesh

In this chapter, we will cover the following recipes:

- Using the Grease Pencil tool to plan the edge-loops flow
- Using the Snap tool to re-topologize the mesh
- Using the Shrinkwrap modifier to re-topologize the mesh
- Using the LoopTools add-on to re-topologize the mesh
- Concluding the re-topologized mesh

Introduction

The re-topology of a mesh, as the name itself explains, is simply the reconstruction of that mesh with a different topology; usually, the re-topology is used to obtain a low resolution mesh from a high resolution one.

In our case, this is obviously needed because we are later going to rig and animate our **Gidiosaurus**, and these tasks would be almost impossible with a mesh as dense as the high resolution sculpted one; we not only need to reconstruct the shape of the mesh with a lower number of vertices, but also with the edge-loops properly placed and flowing for the best render and deformation of the character's features.

In Blender, we have several tools to accomplish this task, both hardcoded into the software or as add-ons to be enabled, and in this chapter, we are going to see them.

Using the Grease Pencil tool to plan the edge-loops flow

It would be perfectly possible to start immediately to re-topologize the high resolution mesh, at least for an expert modeler; by the way, it's usually a good practice to have a guide to be followed in the process, to solve a priori any issue (or at least most of them) that we would come across.

So, let's start this chapter by planning what the right topology can be for a low resolution mesh of our **Gidiosaurus** character; we are going to use the **Grease Pencil** tool to draw the paths of the edge-loops and polygons flow, straight onto the sculpted mesh.

Getting ready

First, let's prepare the screen:

1. Open the `Gidiosaurus_modeling_02.blend` file.
2. Go to the **UV/Image Editor** window to the left and *Shift* + left-click on the **X** icon on the toolbar to get rid of the template image (to be more technically precise, to unlink the template image data block; the *Shift* key is to set the users to **0** and definitely eliminate the image from the file).
3. Put the mouse pointer on the border between the two windows and right-click; in the little **Area Options** pop-up panel, left-click on the **Join Area** item and then slightly move the mouse pointer to the left and left-click again, to join the two windows and obtain a single big 3D viewport window:

Joining the two windows into one

4. Click on the **11th** scene layer button to show only the sculpted **Gidiosaurus** mesh and parts such as the **teeth**, **eyes**, and so on.
5. Go to the **Outliner** and click on the icons showing an eye image placed to the right side of the **Eyeballs**, **Fangs**, and **Talons** items to hide them.
6. Press the *N* key to make the **Properties** 3D view sidepanel appear to the right of the 3D window and scroll it to find the **Grease Pencil** subpanel (already enabled by default); go to the **Tool Shelf** panel to the left of the 3D window and click on the **Grease Pencil** tab:

The Grease Pencil panels and the screen layout in current state

7. Check to enable the **Continuous Drawing** item just below the four buttons at the top of the **Grease Pencil** tab on the **Tool Shelf**.
8. Go to the **Grease Pencil** subpanel under the **Properties** 3D view sidepanel to the right and click on the **New** button; then, click on the + icon button to the left side to add a new **Grease Pencil** layer, which is by default labeled **GP_Layer**; set the **Stroke** color to **RGB 1.000**, **0.000**, **0.350** and **Thickness** of the strokes to **4** pixels.
9. Double-click on the **GP_Layer** name to rename it as **Head**.
10. Go to the **Tool Shelf** and, under **Stroke Placement**, click on the **Surface** button:

Starting to use the Grease Pencil tool

11. Save the file as `Gidiosaurus_retopology.blend`.

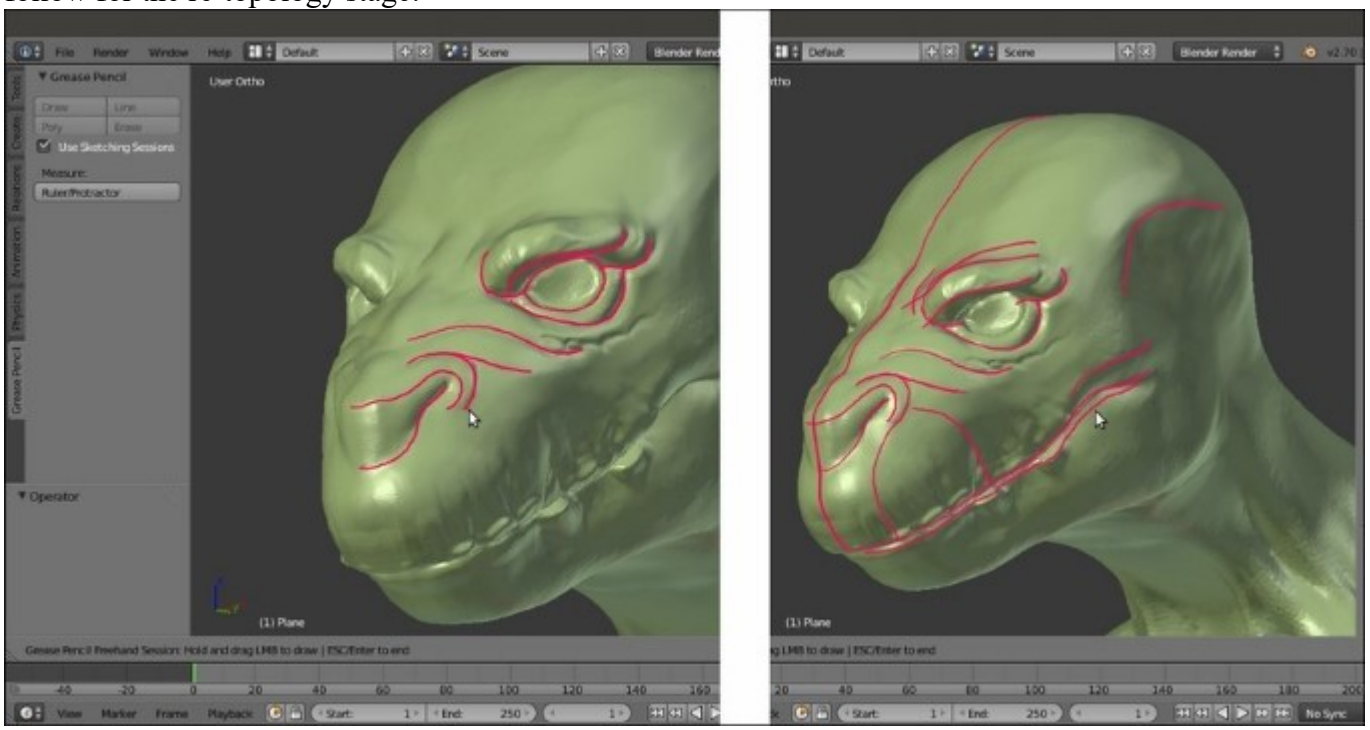
How to do it…

We are now going to start to draw on the character's **head**:

1. Press *Shift* + *B* and draw a box around the head of the **Gidiosaurus** to zoom to it; then, press the *5* key on the numpad to go into **Ortho** view.
2. Click on the **Draw**, **Line** or **Poly** buttons at the top of the **Grease Pencil** tab in the **Tool Shelf**; alternatively, keep the **D** key pressed (along with left-click) to start to draw the first stroke on the mesh (*Ctrl* + *D* + left-click and *Ctrl* + *D* + right-click, respectively for **Line** and **Poly**).

   Because we enabled the **Continuous Drawing** item in the **Tool Shelf**, we can continue to draw without the need to reactivate the drawing mode at each stroke. To quit the sketching session (for example, to change the brush), we can press the *Esc* or the *Enter* keys, so *confirming* the sketching session itself at the same time; otherwise, without the **Continuous Drawing** item enabled, the sketching is confirmed right after each stroke.
3. Start to draw (one half side of the mesh is enough) the strokes; try to make the strokes follow the main, basic, and more remarkable features of the sculpted mesh such as the main **skin folders** going from the **snout** to the **eye sockets** and the bags under the **eyes**, **nostril**, **mouth rim**, and so on.
4. Don't worry too much about the quality or the precision of the strokes; also, don't be afraid to erase (*D* + right-click or the **Erase** button) and/or correct the strokes, if necessary. The **Grease**

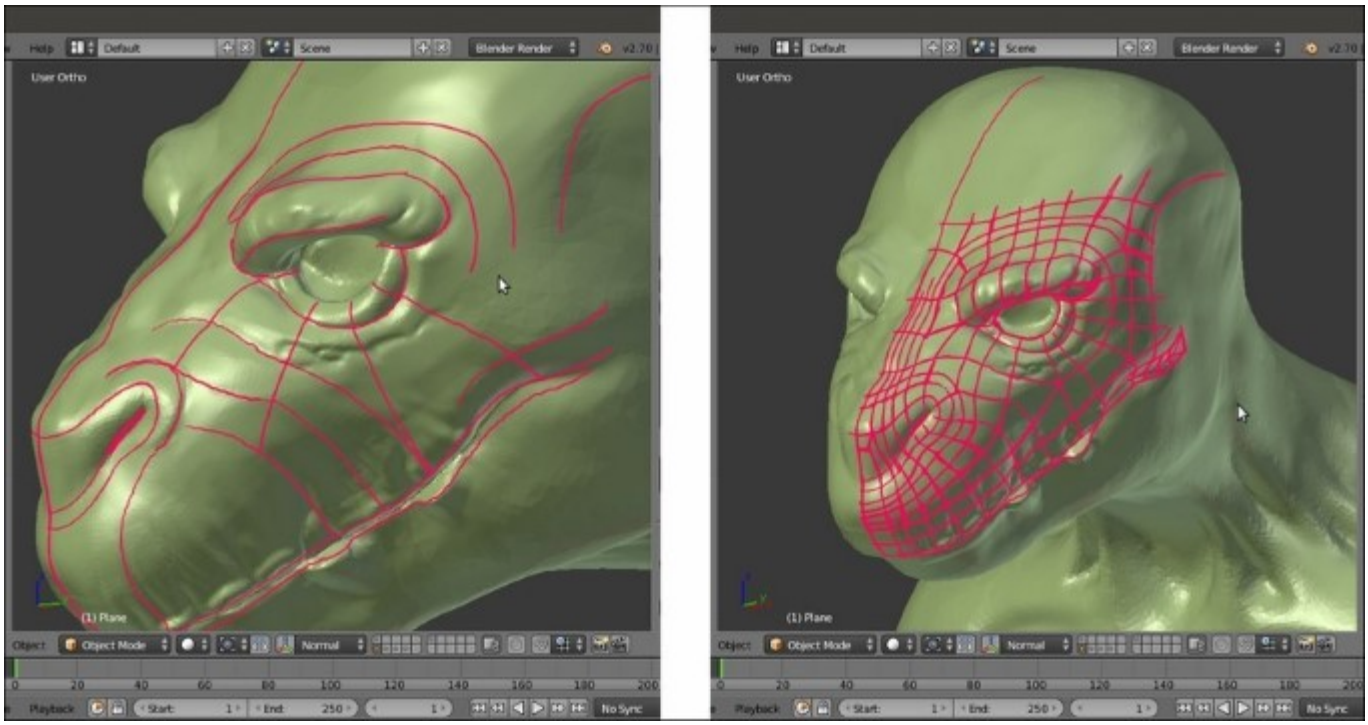
Pencil, in this case, is just a tool to sketch directly on the mesh the guidelines we will later follow for the re-topology stage:



Drawing the head's main features topology

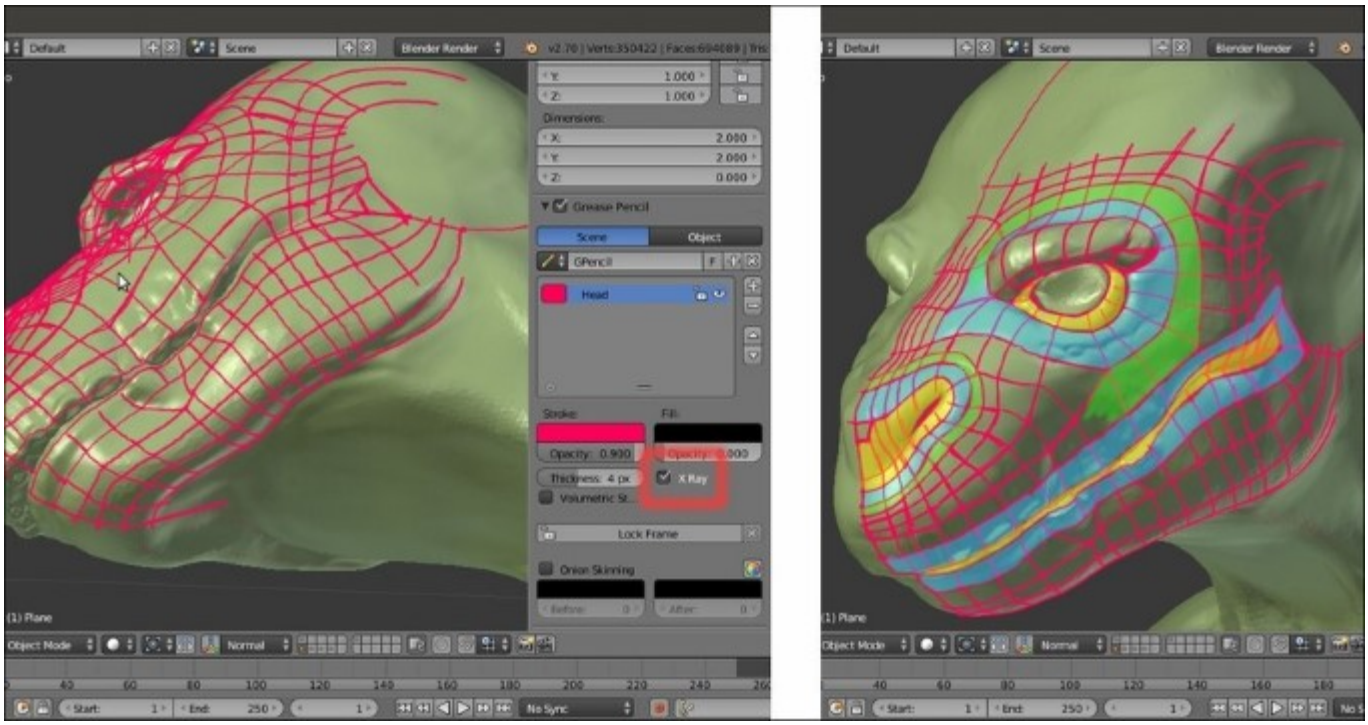
In the case of our **Gidiosaurus**, the topology for a correct deformation is similar to the topology we would use for a human face, but a lot simpler: we just need edge-loops around the **eyes** and in the **eyebrows** area, to give them mobility for expressiveness; a few edge-loops around the **mouth** that, however, in our case, remains quite rigid; and edge-loops following the folders on the top of the **snout**, which can also be important for the *growl* expression.

5. Once the strokes for the main features have been posed, try to join them into a web of edges, as balanced and efficient as possible:



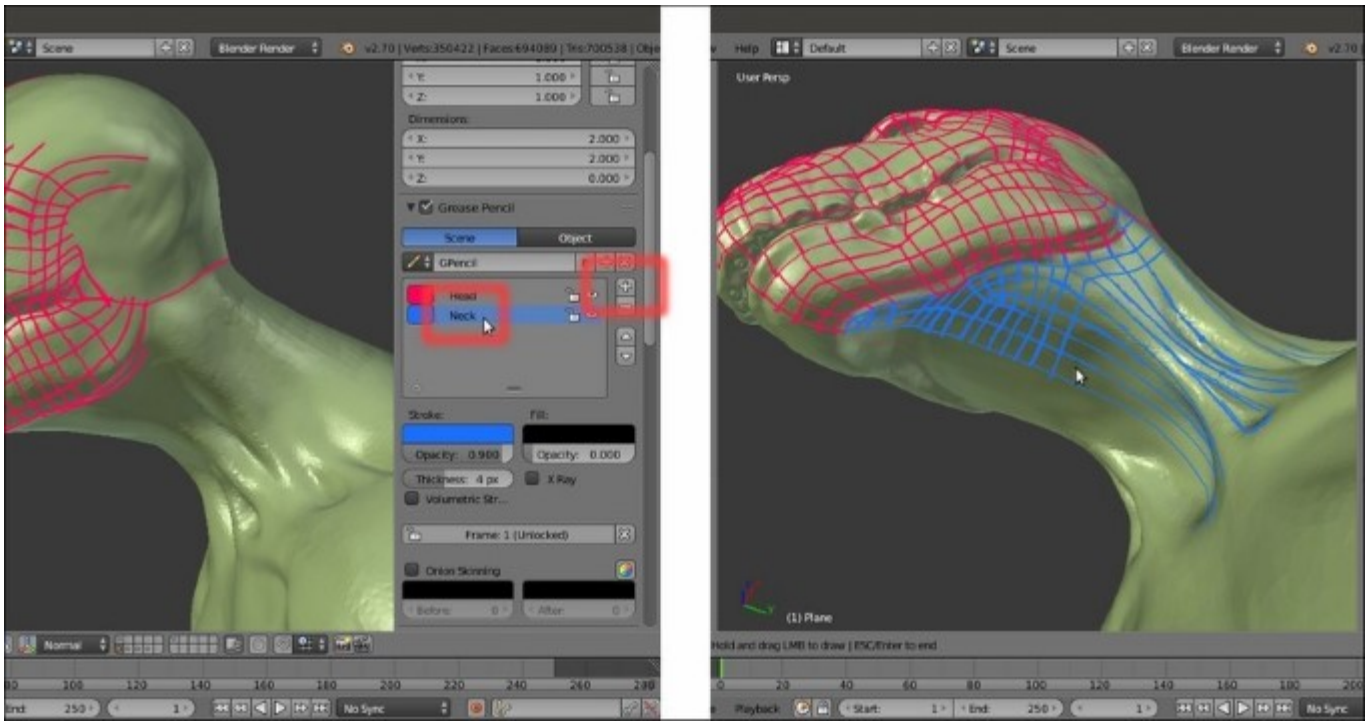
Connecting the strokes

6. At a certain point, when and if the overlapping of the strokes starts to become confusing, you can uncheck the **X Ray** item, which is located to the right side of the **Thickness** slot in the **Grease Pencil** layer subpanel, to disable the visibility of the strokes behind the mesh surface.
7. Forget about the edge-loops of stiff parts such as the cranium; it's enough to plan the position and the flow of the deforming ones. In the screenshot at the bottom right, I have highlighted (in **Gimp**) the main facial edge-loops for the **Gidiosaurus** with different colors:



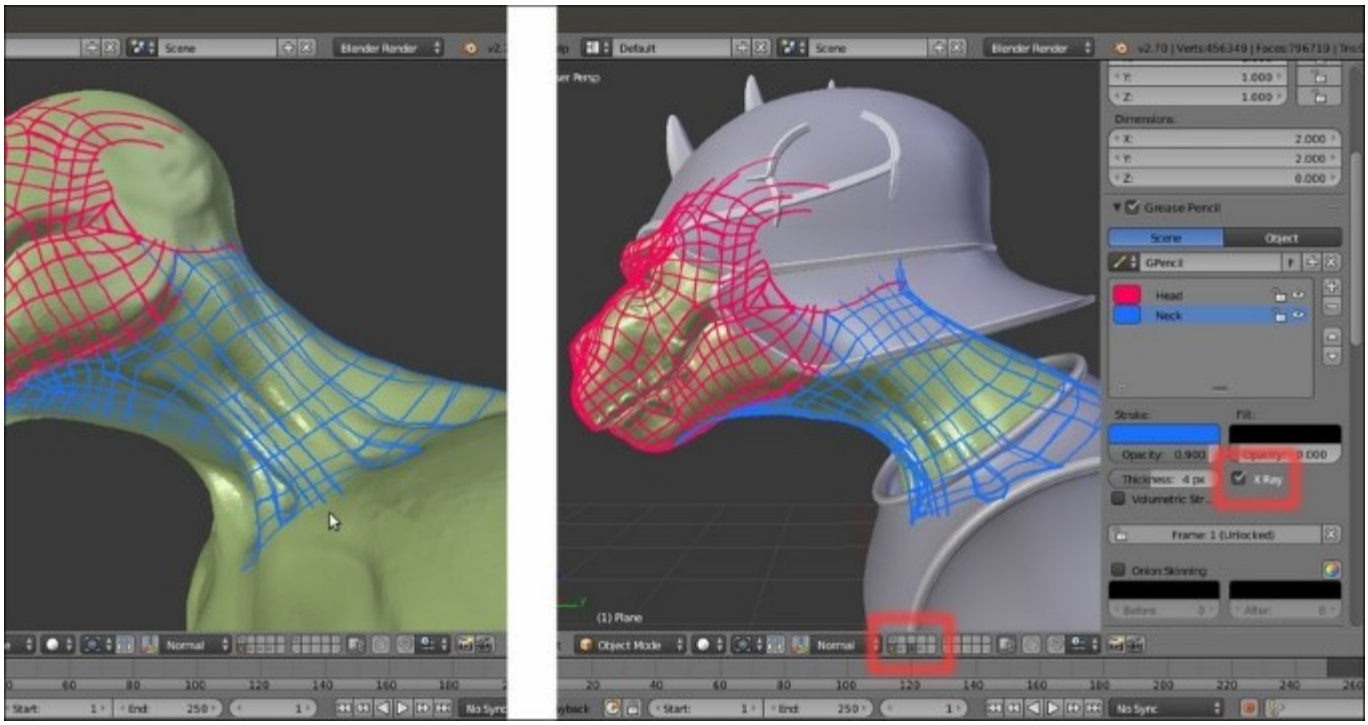
The X Ray button and the highlighted main edge-loops

8. When you think you are done with the **head**, click on the + icon button to add a new layer and rename it as **Neck**. Set the values the same you did for the **Head** layer, just change the color of the strokes; I set mine to **RGB 0.106**, **0.435**, **0.993**, but whatever color you choose, be sure that it stands out in the viewport against the mesh color.
9. In the case of the **neck**, the important thing is to find the correct joining with the **head's** edge-loops under the lower **jaw**, as you can see in the bottom-right screenshot:



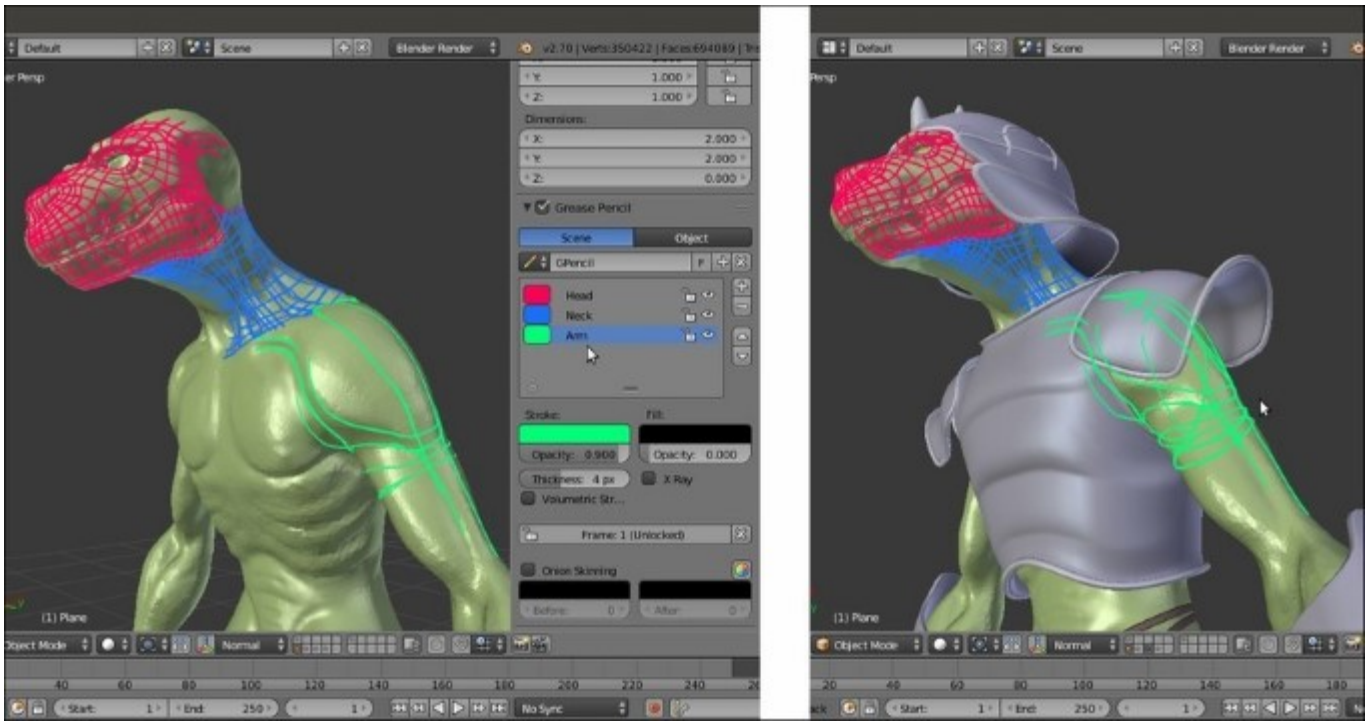
The Neck layer

10. Continue to stroke on the **neck** by drawing parallel horizontal loops along its length and use the vertical strokes to outline the **neck's** muscles (don't look for a **sternocleidomastoid** muscle here; the **Gidiosaurus**, anatomy, although similar in some ways, is not human at all!).
11. Remember that because our character is wearing an **armor**, it is not necessary to re-topologize the whole body, but only the exposed parts; so we can stop the planning just a little beyond the plates outside edges. To verify the correct extension of the strokes, just be sure to have the **X Ray** item enabled in the **Grease Pencil** layers and also the **13th** scene layer enabled to show the **armor**:



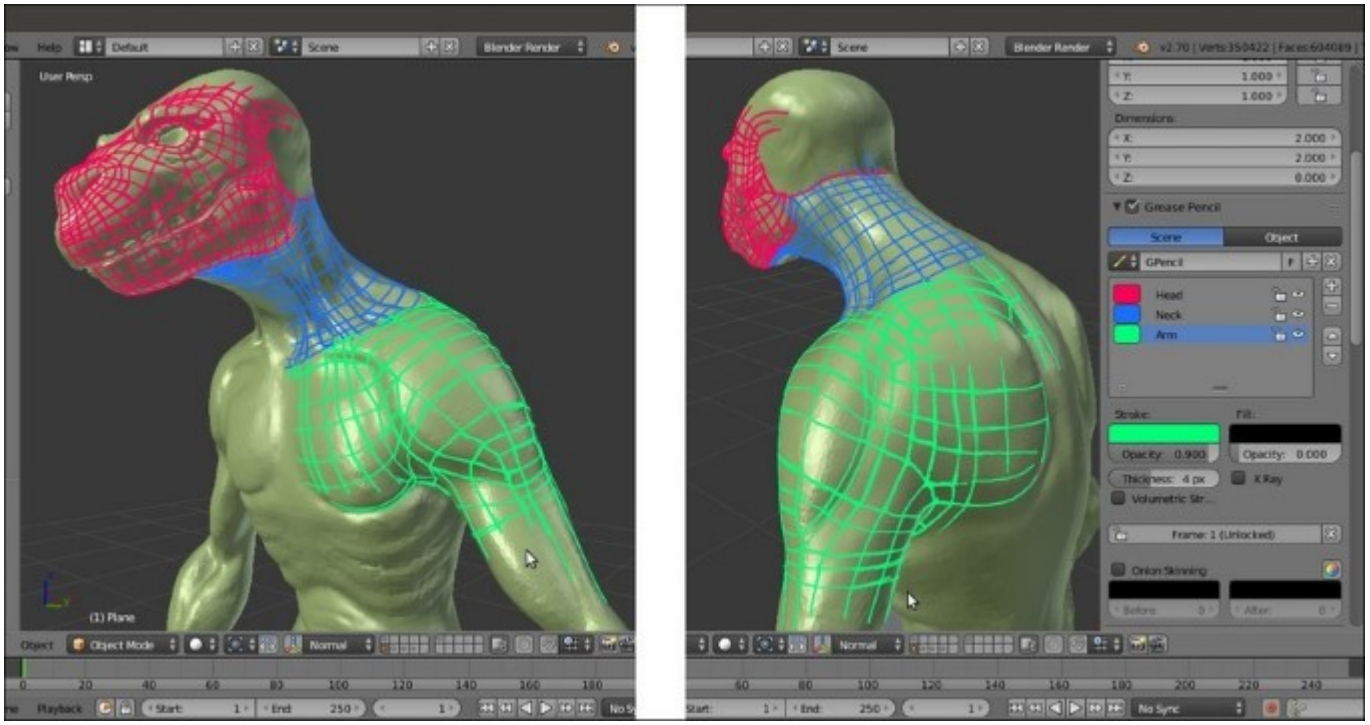
Verifying the extension of the strokes under the armor

12. Click again on the + icon button in the **Grease Pencil** subpanel under the **Properties** 3D view sidepanel and rename the new layer as **Arm**. Set the values the same as you did for the **Head** and **Neck** layers, but change the color once more (**R 0.000**, **G 1.000**, **B 0.476**); this time, we have to plan the joining of the cylindrical shape of the **arm** with the **shoulder** and the **collar bones** areas:



Sketching the guidelines on the arm

13. As before, also in this case, it is not necessary to go beyond the boundaries of the **armor chest** plate, but including also the muscles of the **chest** and **back** in the topology planning can give a more natural result:



The completed guidelines for the shoulder and the arm joining

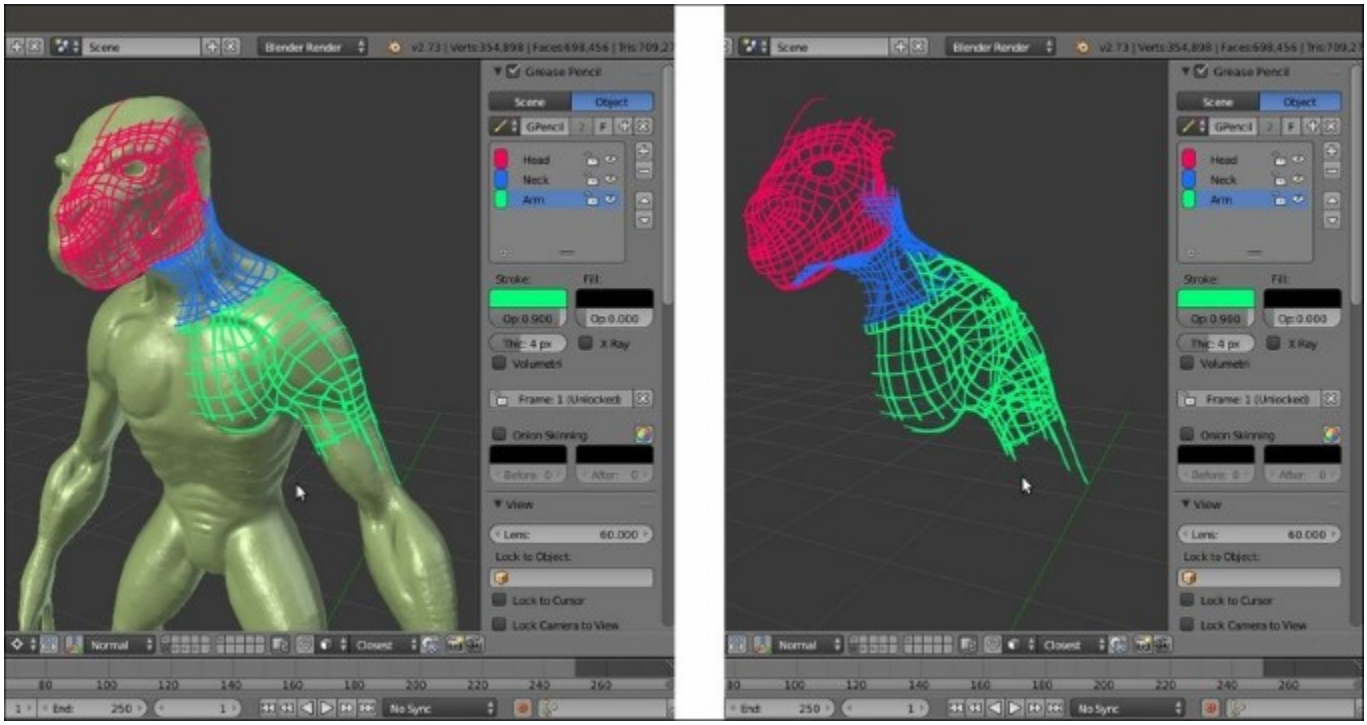
14. When you are done, save the file.

At this point, we can stop with the **Grease Pencil** sketching of the topology; the remaining parts of the exposed body are a lot simpler and will be quickly resolved in the successive recipe of this chapter.

There's more…

We can load any already existing **Grease Pencil** layer data blocks even into an empty scene, by clicking on the little arrows on the left-hand side of the **Gpencil** slot (*Freehand annotation sketchbook*) at the top of the **Grease Pencil** subpanel on the **Properties** 3D view sidepanel, and indifferently for **Scene** or **Object**. Actually, the **Grease Pencil** tool can be used as a sketchbook tool, to write quick notes and/or corrections inside the **Node Editor** window or the **UV/Image Editor** window, and even as an animation tool, by drawing inside an empty scene or on the surface of other objects to be used as *templates*.

In the following screenshot, you can see the sketching sessions previously made on the **Gidiosaurus** object's surface, showing *a solo* and keeping the volumes of the character in the 3D space:



The Grease Pencil layers in the 3D space

See also

- http://www.blender.org/manual/grease_pencil/introduction.html

Using the Snap tool to re-topologize the mesh

In this recipe, we'll use the **Snap** tool to start to re-topologize the sculpted high resolution mesh.

Getting ready

First, let's prepare both, the mesh to be *traced*, which is the high resolution mesh, and the tool itself:

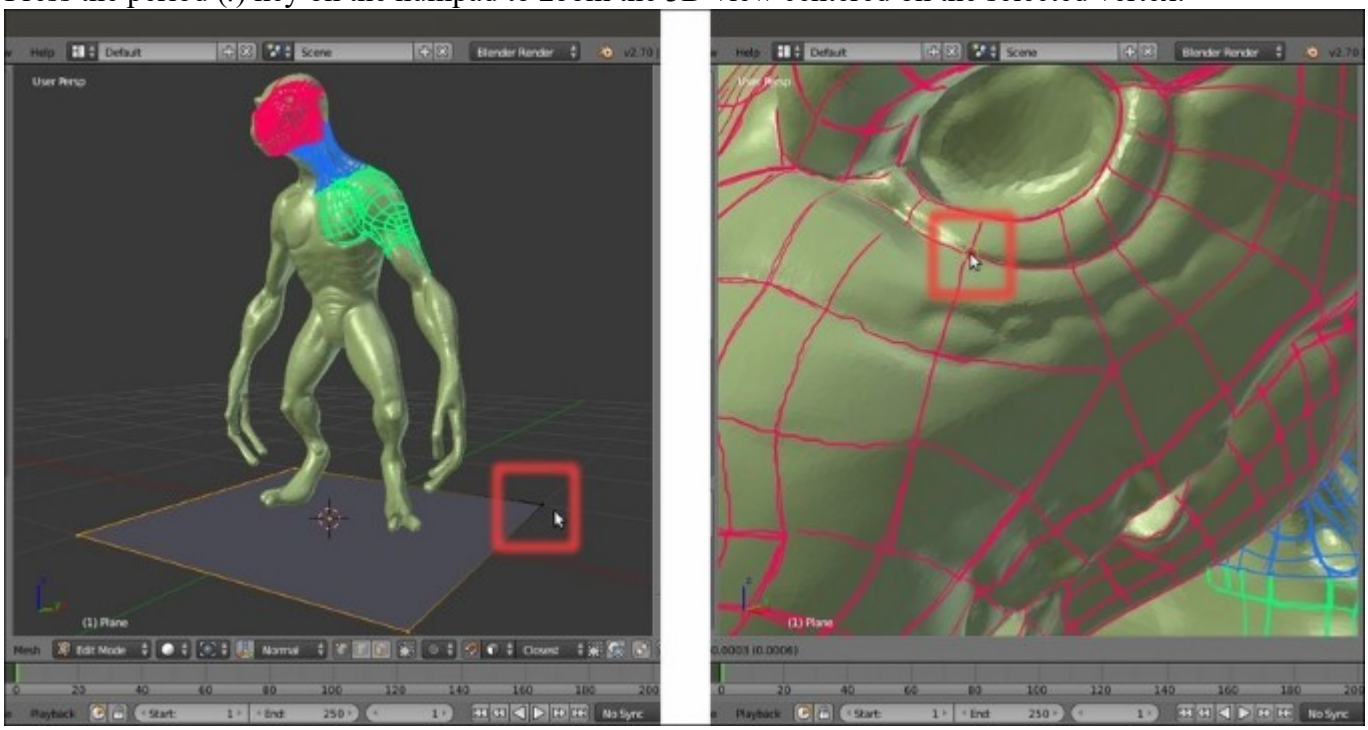
1. Go to the **Outliner** and click on the *Restrict viewport selection* icon, which is the arrow one, to the side of the **Gidiosaurus** item to make it not selectable.
2. Be sure that the **3D Cursor** is at the center of the scene (*Shift* + *C*) and add a **Plane** primitive.
3. Click on the *Snap during transform* button, the little icon with the magnet, on the 3D view toolbar, or else press *Shift* + *Tab* to activate the tool.
4. Click on the **Snap Element** button (*Type of element to snap to*) on the close right to select the **Face** item, or else press **Shift** + **Ctrl** + **Tab** to make the **Snap Element** pop-up menu appear in order to select the item from:

The Snap Element menu

How to do it…

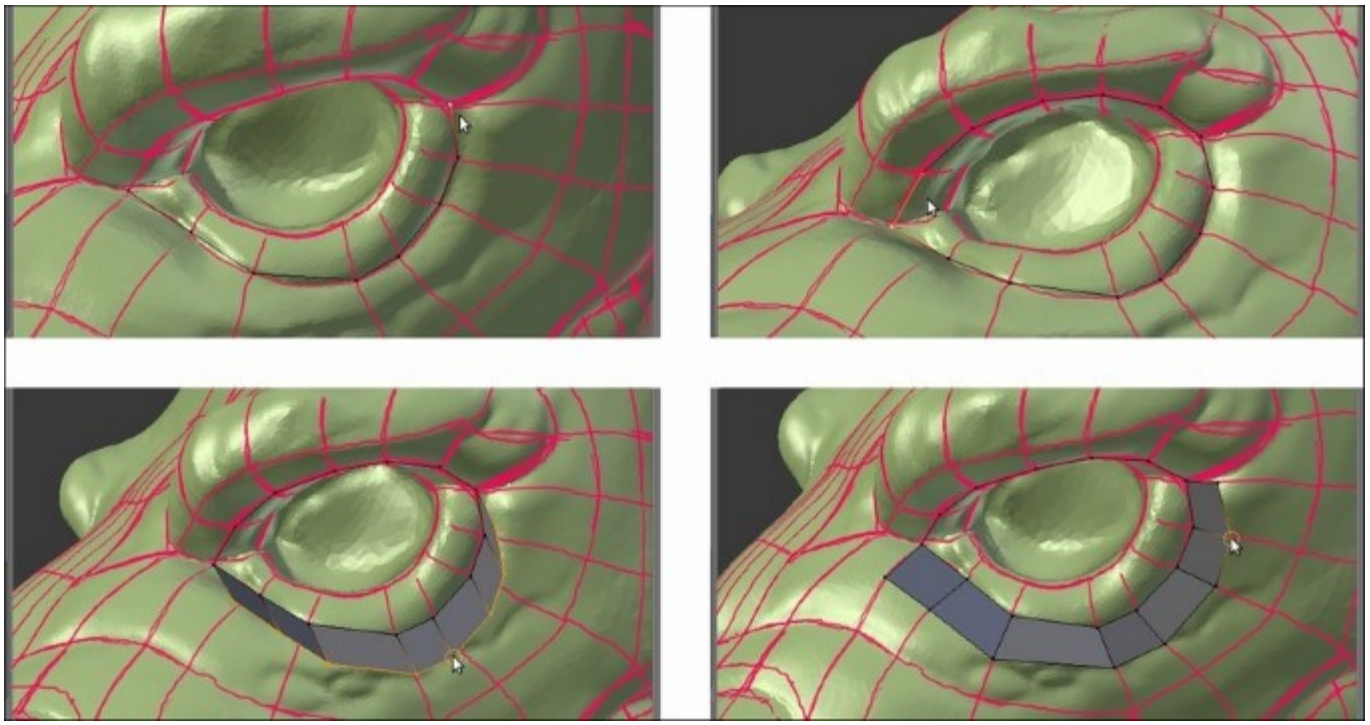
Now, we are going to start the re-topology:

1. With the **Plane** object selected, press *Tab* to go into **Edit Mode**; with all the vertices already selected by default, by pressing *Shift* + right-click, deselect just **one** vertex (anyone of them, it doesn't matter which one).
2. Press *X* to delete the other **three** vertices that are still selected.
3. Select the single remaining vertex and move it onto the **head** of the sculpted mesh, close to the **left eye socket**; as the **Snap** tool is enabled, the vertex stays on the mesh surface.
4. Press the period (.) key on the numpad to zoom the 3D view centered on the selected vertex:



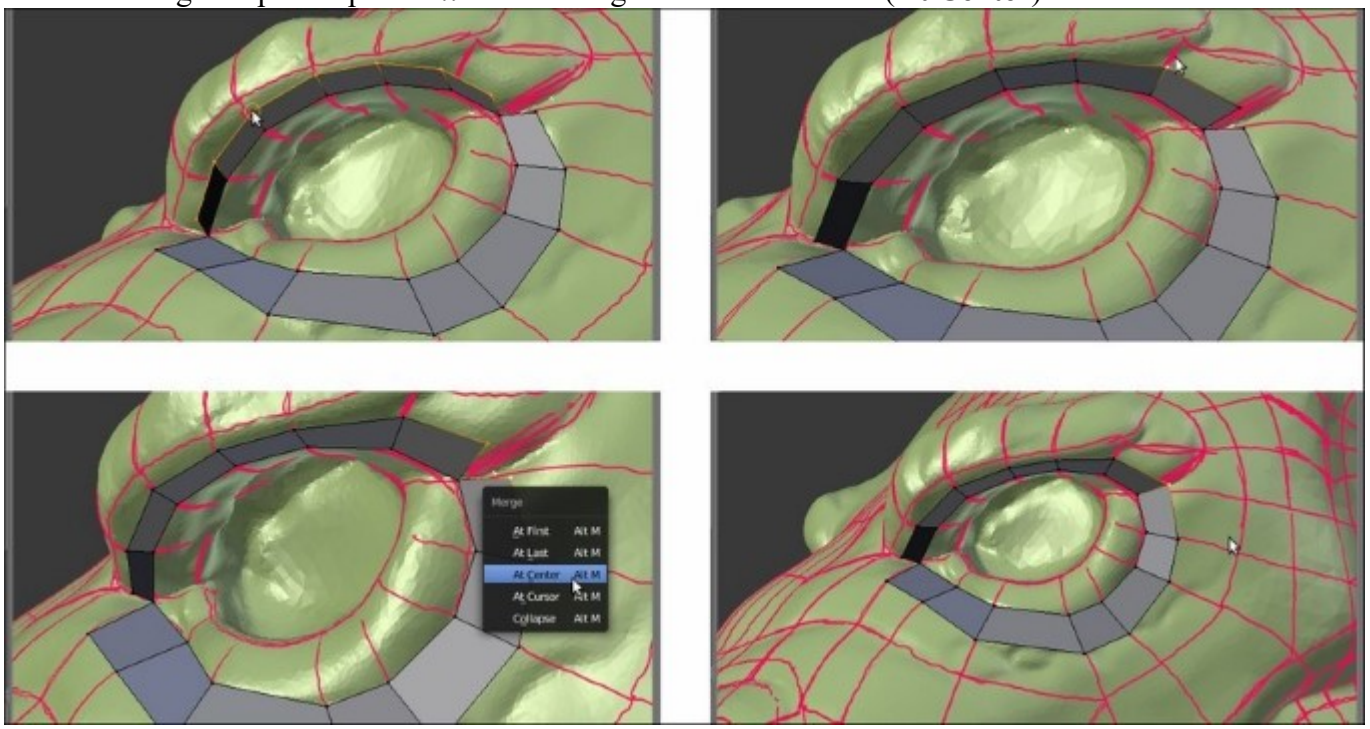
Starting the re-topology process

5. Go to the **Object Data** window and check the **X-Ray** item under the **Display** subpanel in the main **Properties** panel to the right of the screen.
6. Start to extrude the vertex, building an edge-loop around the **eye socket** and following the **Grease Pencil** guideline, both by pressing the *E* key or *Ctrl* + left-click to add vertices; if needed, press *G* to move them at the right location (that is, at the intersections of the guidelines).
7. When you have almost completed the edge-loop around the **eye socket**, select the last and the first vertices and press the *F* key to close it.
8. Select the bottom row of vertices of the edge-loop and extrude them; adjust the position of each vertex on the ground of the strokes guideline:



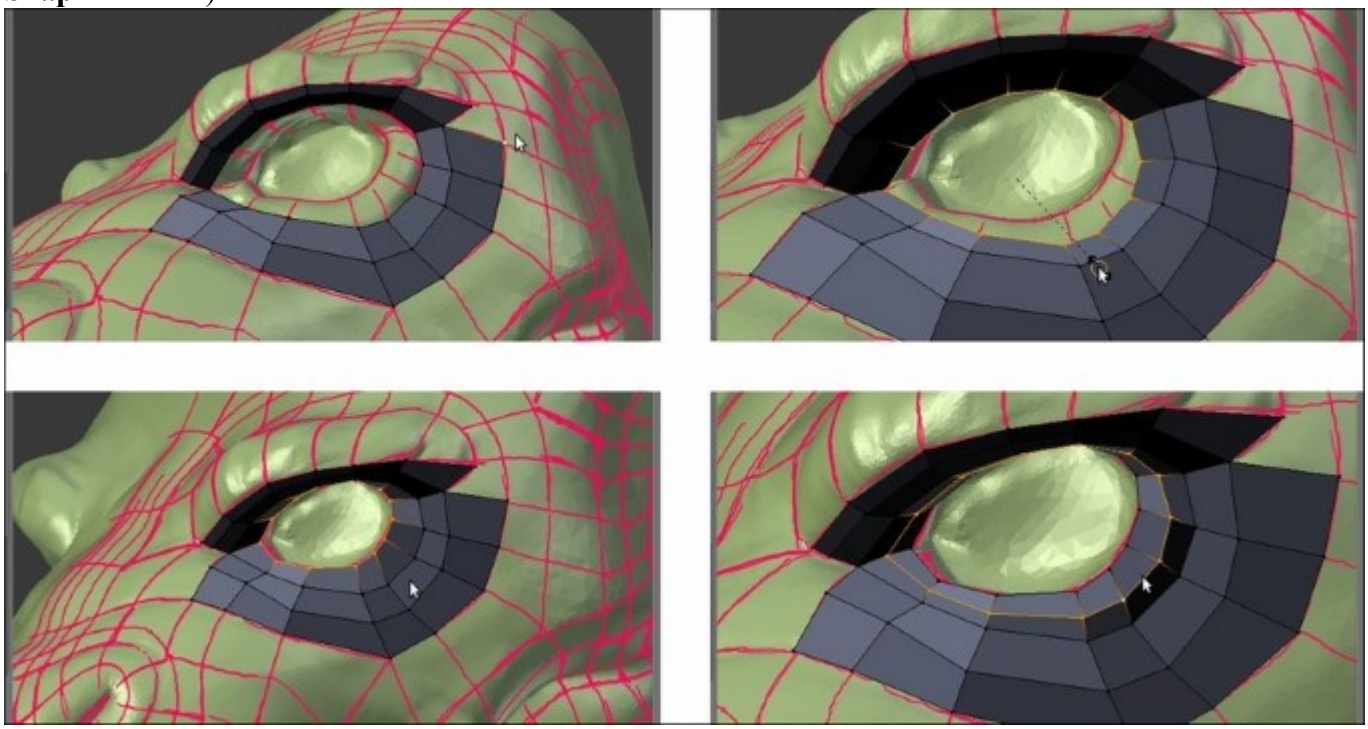
The first re-topology around the eye socket

9. Do the same with the upper row of vertices and then select the free vertices on the right-hand side of the edge-loops and press *Alt* + *M* to merge them at the center (**At Center**):



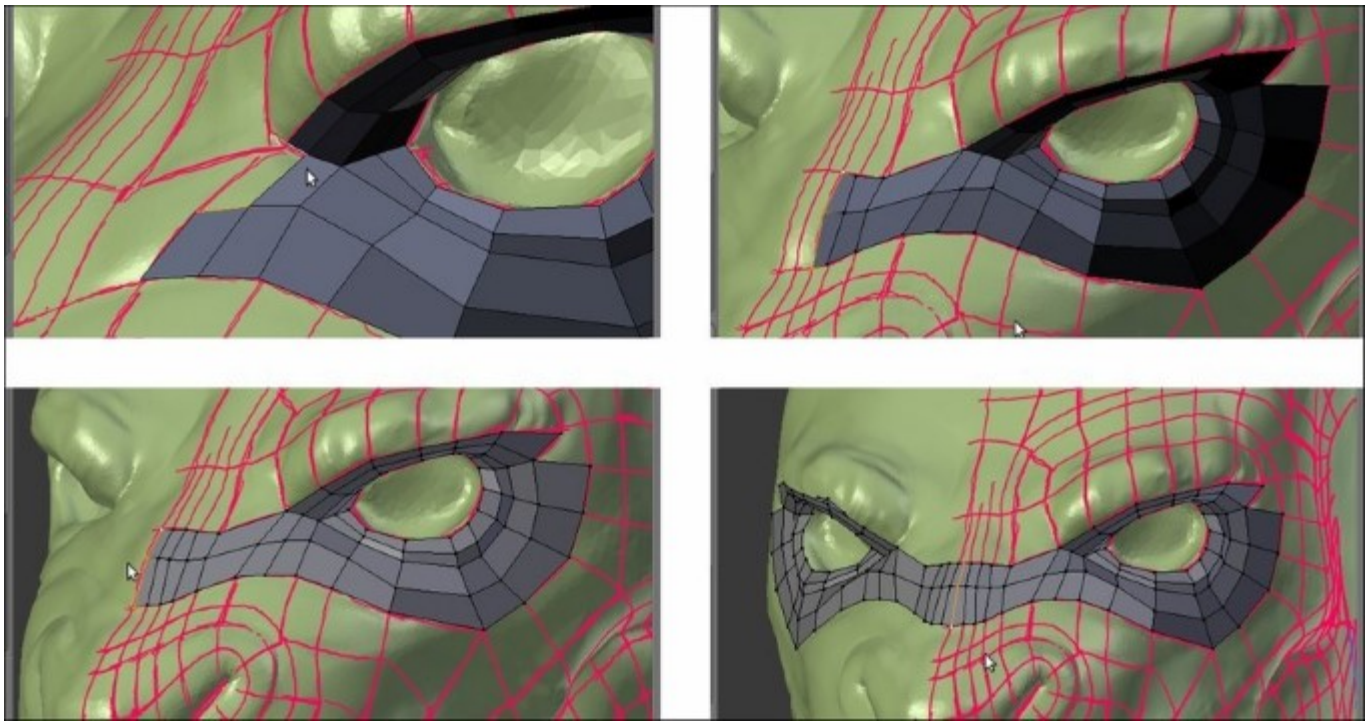
Building the eye edge-loop

10. While still in **Edit Mode**, select all the vertices and press *Ctrl* + *N* to recalculate the normals.
11. Keep on extruding the edge-loops to build the faces around the **eye socket**. Select the inner edge-loop and extrude it; then, scale it inside and adjust the vertices position as usual.
12. Cut a new edge-loop in the middle of the **eye socket** by pressing *Ctrl* + *R* and then select each vertex; press *G* and, immediately after, click with the left button of the mouse. This way the newly added vertex stays in place, but is snapped to the underlying surface (sadly, it doesn't work automatically as you cut or add vertices; they must be moved in some way to make the **Snap** tool work).



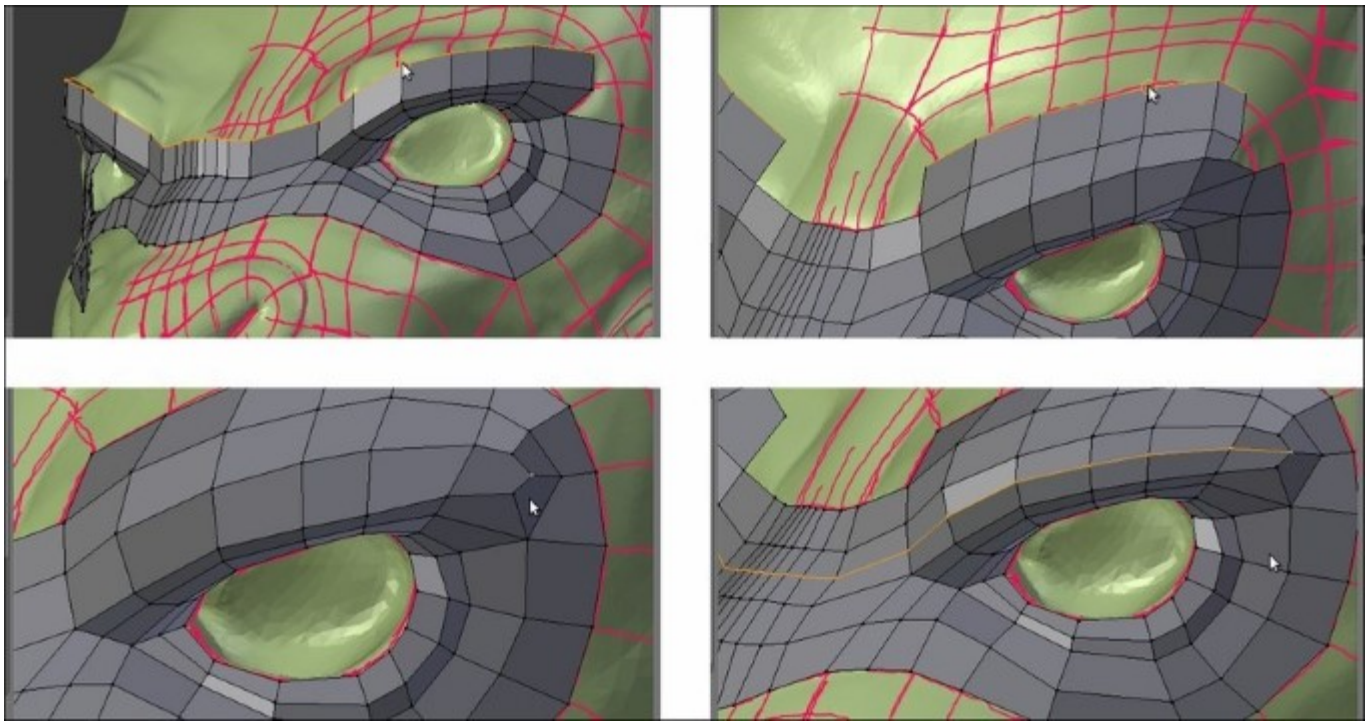
Adding geometry and snapping the vertices to the surface

13. Keep on adding geometry to the mesh, extruding or *Ctrl* + left-clicking, and switch between edges and vertices selection mode to make the workflow faster. Press *5* on the numpad to go into **Ortho** view when necessary. Following the strokes guideline, build faces going towards the median line of the object.
14. As you are arrived to the median line of the object, go to the **Object Modifiers** window under the **Properties** panel and assign a **Mirror** modifier.
15. Click on the *Adjust edit cage to modifier result* icon (the last one in the row to the side of the modifier's name), to activate the modifier during the editing, and check the **Clipping** item.
16. Adjust the vertices you just added to the median line of the mesh to stay on the *y* axis and recalculate the normals.
17. Go to the **Outliner** and rename the **Plane** item as **Gidiosaurus_lowres**.



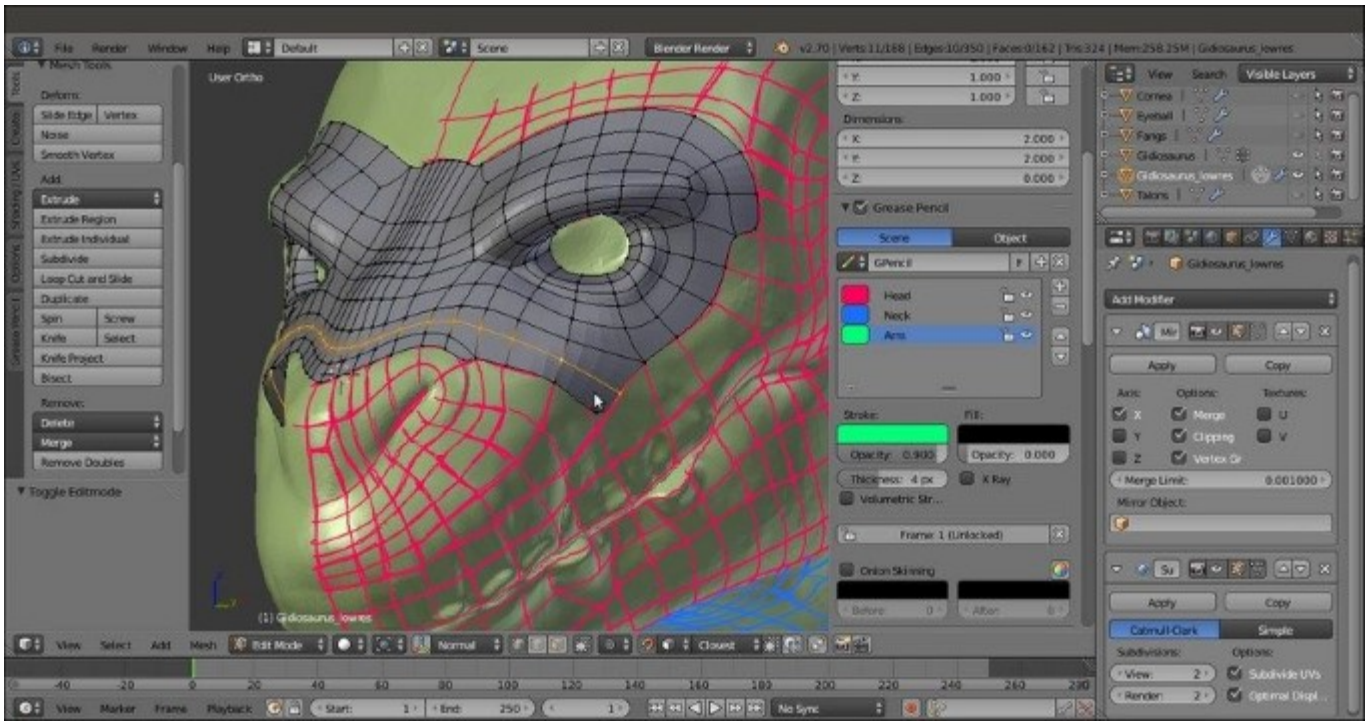
Going towards the median line

18. Build the remainder of the faces the same way, extruding edges or vertices, moving them to react to the **Snap** tool, and adding cuts and edge-loops where needed to keep all quads. **N-gons** faces can be split into quads by dividing an edge to add a vertex in the middle, selecting the new vertex and its opposite one and pressing the *J* key to connect them (see the two screenshots at the bottom row):



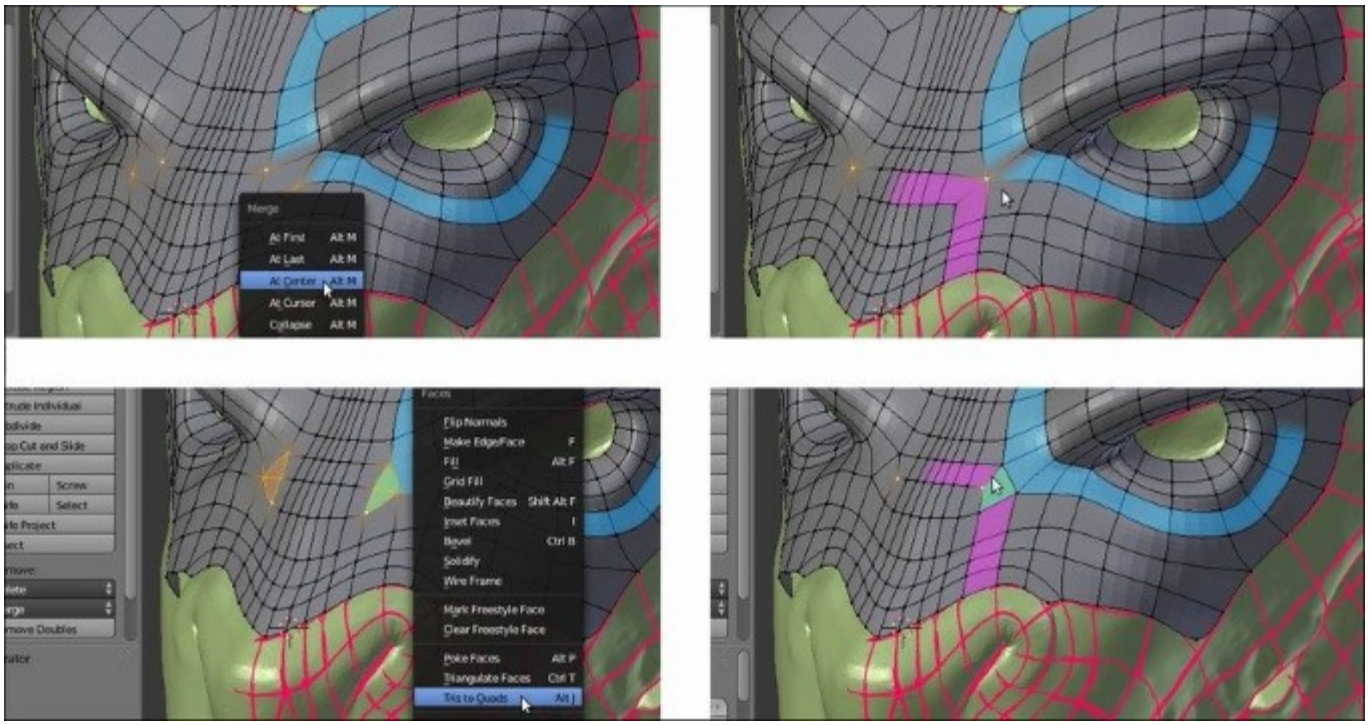
Building the eyebrows and dividing N-gons into two quad faces

19. Assign a **Subdivision Surface** modifier to the low resolution mesh and set **Subdivisions** to **2**. Check the **Optimal Display** item; if you want, click on the *Adjust edit cage to modifier result* icon, which is the last one in the row to the side of the modifier's name. To work with an already smoothed mesh (in the end, the mesh will be subdivided in any case) is a usual workflow; by the way, it depends on your preferences. If you prefer to work without the modifier, occasionally go out of **Edit Mode** to verify how the geometry behaves under the **Subdivision Surface** modifier.



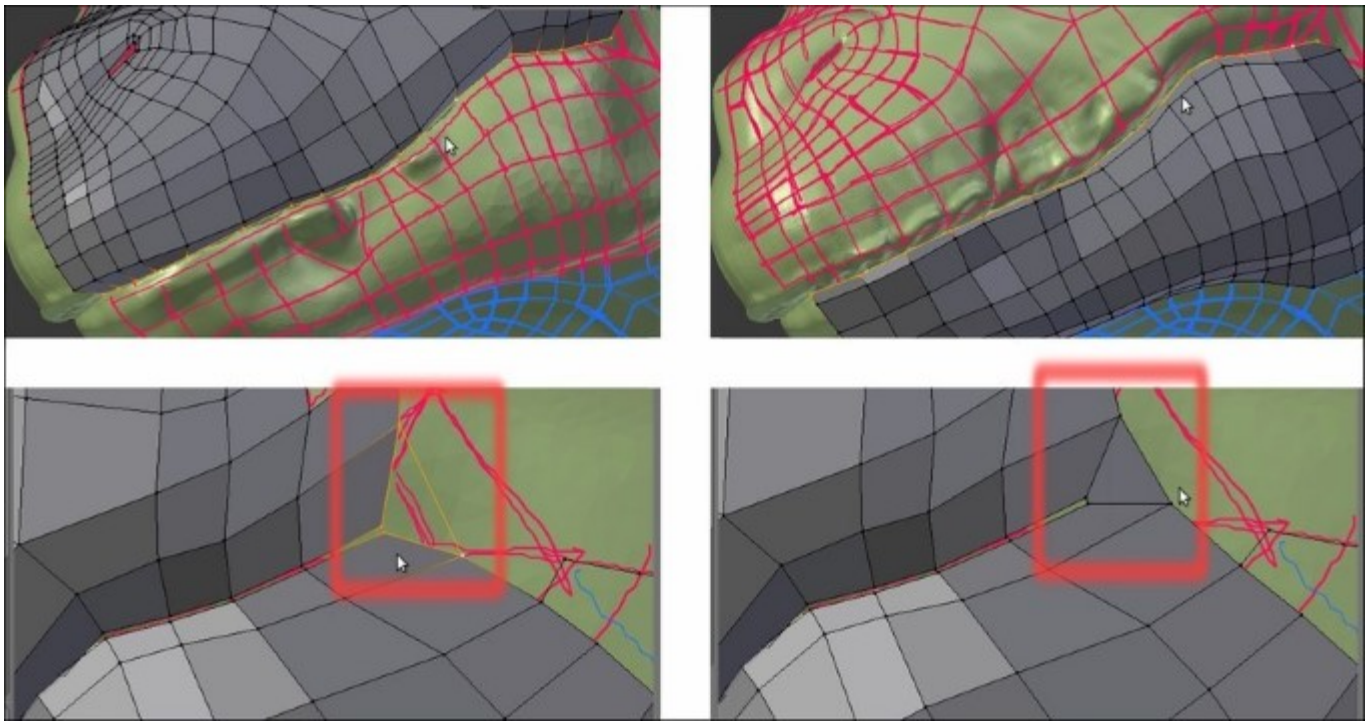
The created geometry in the Subdivision Surface visualization mode

20. Around the eyebrows, it is important to have continuous edge-loops to allow for better mesh deformation; often, it is enough to merge (*Alt* + *M*) two vertices to obtain the right flow. Note that this creates a pole (check out the screenshot at the top right) that can later be eliminated by a cut and then merges the two tris faces into a quad (*Alt* + *J*; the two screenshots at the bottom):



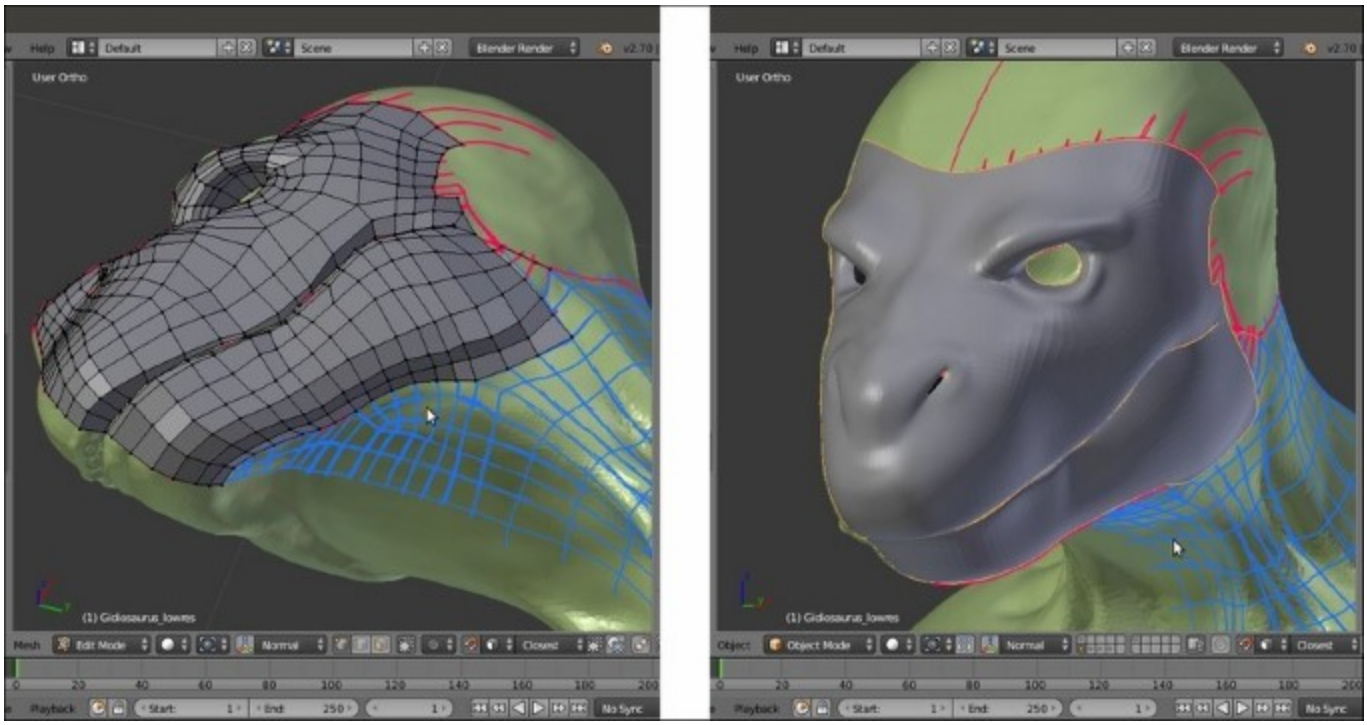
Closing two edge-loops

21. We have almost completed the **Gidiosaurus'** face. Select the vertices of the lower **jaw** and press the *H* key to hide them; select the **upper mouth rim** and extrude it and then adjust the vertices' position.
22. Deselect the vertices, press *Alt* + *H* to unhide the **mandible**, and press *Shift* + *H* to hide the unselected vertices (in this case, the **upper face**). Select the **mouth rim** of the **mandible** and extrude and then tweak the position of the vertices.
23. Connect the upper and the lower **jaws** by connecting the last vertices, as shown in the bottom-left of screenshot and then build a face. Tweak the vertices' position:



Connecting the jaw to the upper mouth

We can stop using the **Snap** tool at this point and continue with the re-topologizing by using different tools; we'll see this in the upcoming recipes.



The re-topologized face of the character

How it works…

The main requirement for a re-topology tool is the ability to trace the shape and volume of the high resolution mesh as easily as possible. In this recipe, we used the Blender **Snap** tool that, once set to **Face**, guarantees that every added vertex lies on the faces of any directly underlying object; this way, it is quite simple to concentrate on the flow of the polygons, while their vertices stay anchored to the mesh's surface.

To remark that the strokes are there only as a generic indication, note that in certain areas we are doubling the number of faces sketched with the **Grease Pencil** tool as well as to try to keep the density of the mesh as even as possible.

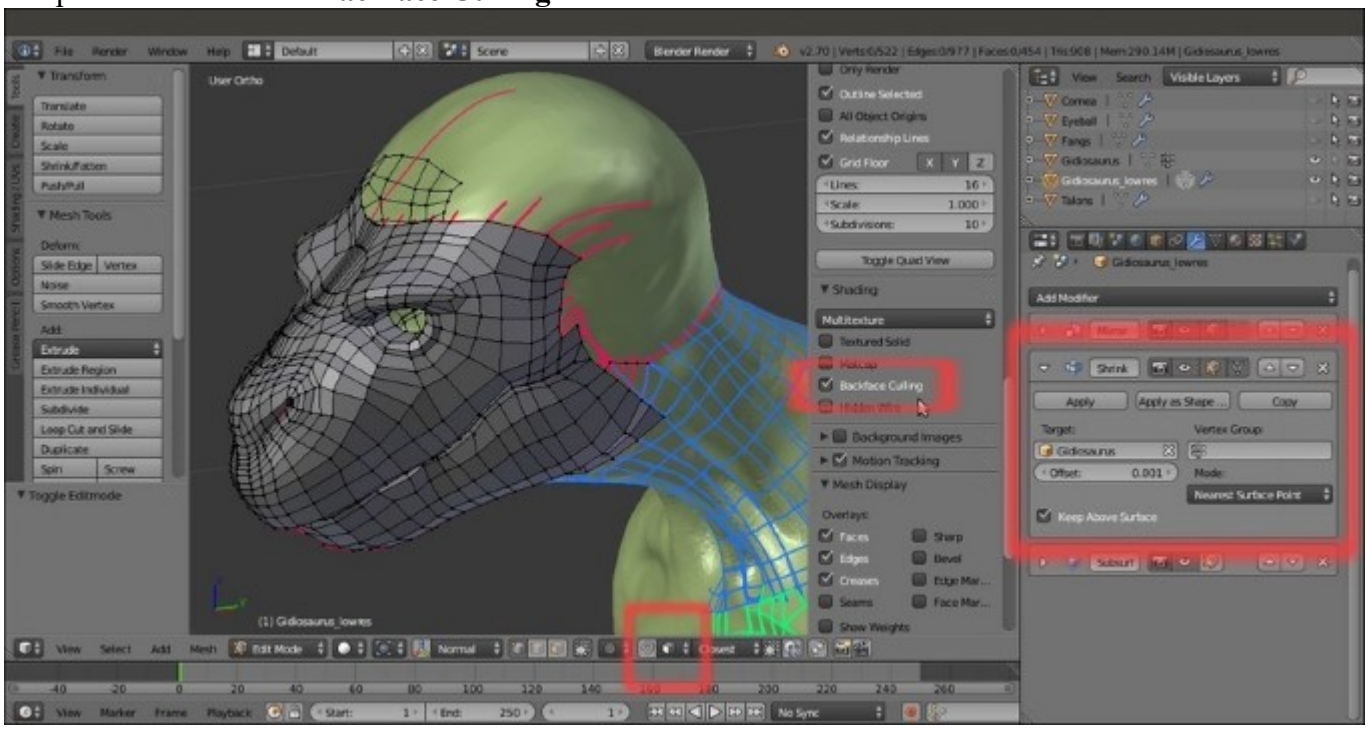
Using the Shrinkwrap modifier to re-topologize the mesh

Sometimes, the **Snap** tool is not enough or can be quite difficult to use because of a particular shape of the high resolution mesh; in these cases, the **Shrinkwrap** modifier can be very handy.

Getting ready

Basically, the usage of this method is all in the preparation of the modifier:

1. Assign the **Shrinkwrap** modifier to the **Gidiosaurus_lowres** mesh and, in the modifier stack, move it *before* the **Subdivision Surface** modifier.
2. Click on the **Target** field to select the **Gidiosaurus** mesh item and leave the **Mode** option to **Nearest Surface Point** (this seems to be the more efficient mode for this task; by the way, you can experiment with the other two modes that can reveal themselves useful in other situations).
3. Enable the *Display modifier in Edit mode* and *Adjust edit cage to modifier result* buttons (the penultimate one and the last one to the right, with the cube and four selected vertices image and with the upside-down triangle and three vertices image, respectively) and the **Keep Above Surface** item.
4. In **Edit Mode**, if it's necessary to make the low resolution mesh more easily visible against the high resolution one, change the **Offset** value to **0.001**.
5. Having the **X-Ray** item still active, go to the **Shading** subpanel under the **Properties** 3D view sidepanel and check the **Backface Culling** item:

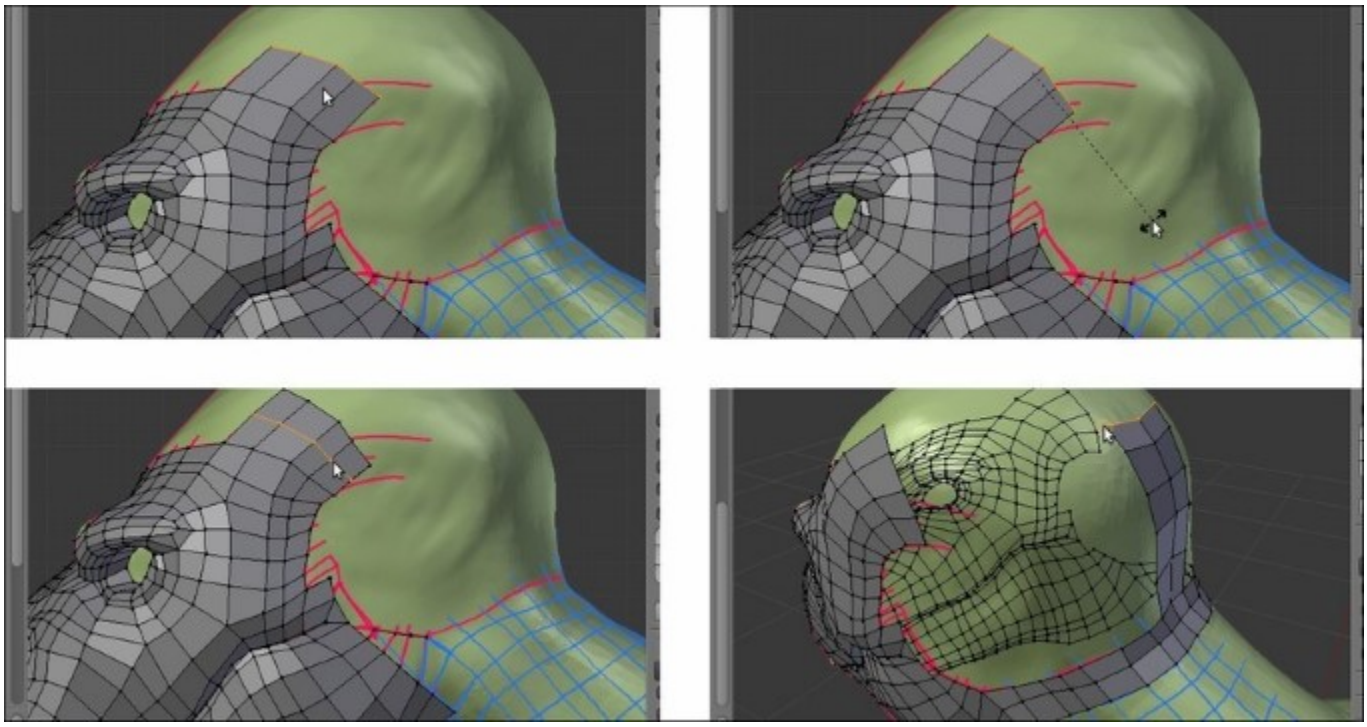


The Shrinkwrap modifier panel

How to do it…

In **Edit Mode**, select, extrude, and move the vertices as required! The **Shrinkwrap** modifier will take care of keeping the vertices adhering to the target mesh surface.

If you are having issues, such as vertices jumping everywhere as you try to move them, try to disable the **Snap** tool. This is not always the case, but sometimes the combination of both the tool and the modifier can give unexpected results; other times, it can be the opposite.



Extruding and cutting an edge-loop under the Shrinkwrap modifier

Remember that if you are using this method to re-topologize, at the end of the process, you *must apply the Shrinkwrap modifier*.

Also, save the file.

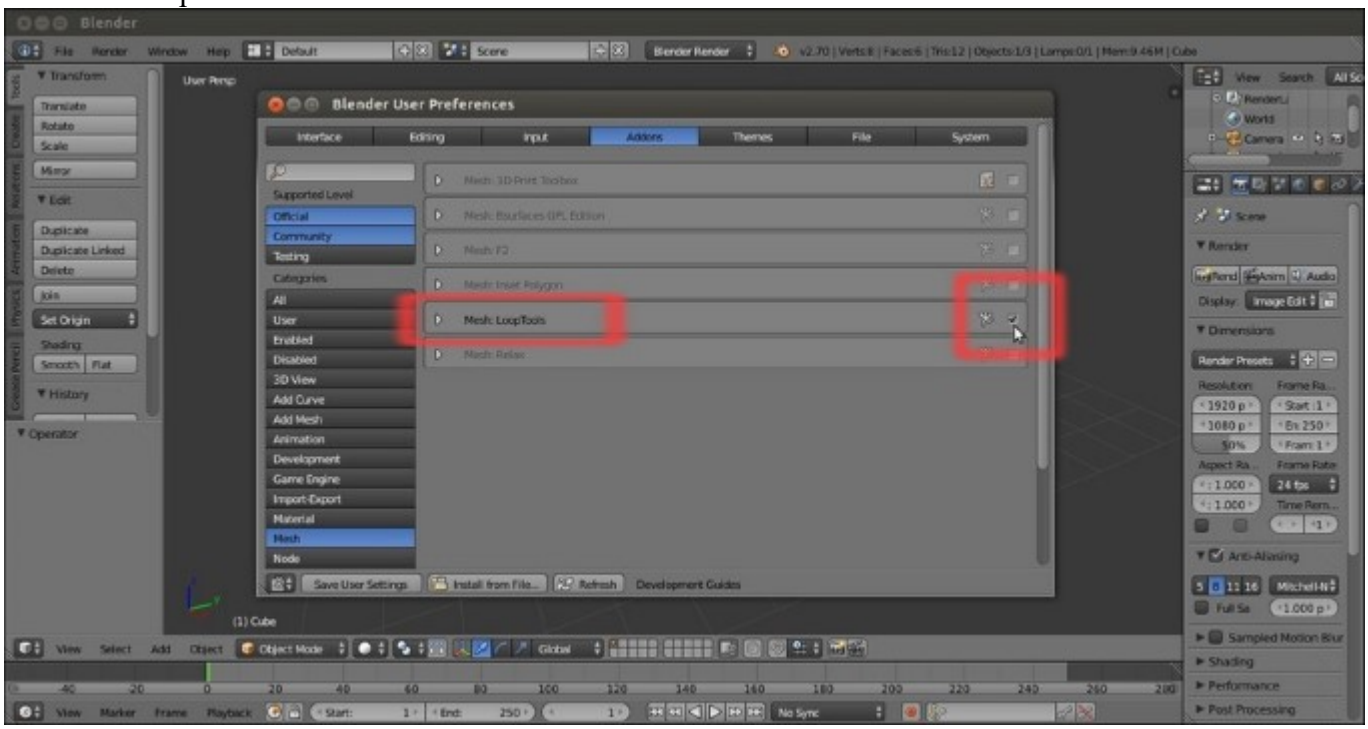
Using the LoopTools add-on to re-topologize the mesh

We have already seen the **LoopTools** add-on in Chapter 3, *Polygonal Modeling of the Character's Accessories*. This incredibly useful Python script can even be used for the re-topology!

Getting ready

If the **LoopTools** add-on isn't enabled yet, perform the following steps:

1. Start Blender and call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*); go to the **Addons** tab.
2. Under the **Categories** item on the left-hand side of the panel, click on **Mesh**.
3. Check the empty little box to the right of the **Mesh: LoopTools** add-on to enable it.
4. Click on the **Save User Settings** button at the bottom-left of the panel to save your preferences and close the panel:



The User Preferences panel and the LoopTools add-on enabled

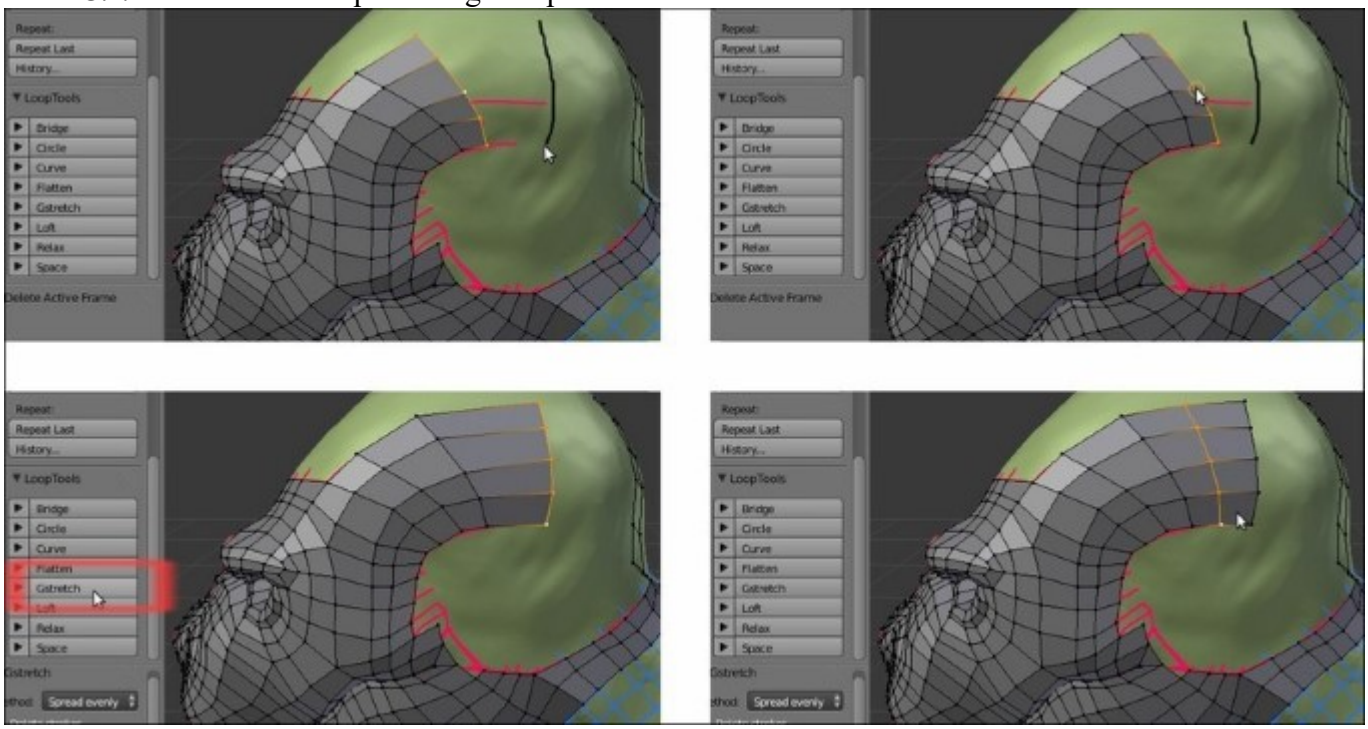
5. Load the `Gidiosaurus_retopology.blend` file.
6. Click on the *Snap during transform* button on the 3D view toolbar (or else, press *Shift* + *Tab*) to enable the **Snap** tool again.

How to do it…

In the **LoopTools** add-on, there are at least three tools that can be used for the re-topology: **Gstretch**, **Bridge**, and **Loft** (the last two seem to have almost the same effect so, at least for our present goal, we can consider them to be interchangeable).

Let's first see the **Gstretch** tool:

1. Go to the **Grease Pencil** subpanel under the **Properties** 3D view sidepanel to the right. Be sure that the **Grease Pencil** checkbox is checked and click on the + icon button to add a fourth layer after the **Arm** layer (actually, you can also delete the preexisting **GPencil** data block and start with a brand new one, or in any case disable the visibility of the other layers); leave the strokes color as it is by default—that is, pure black.
2. In **Edit Mode**, press *D* and sketch one edge-loop stroke.
3. Select the edges of the low resolution mesh and press *E* to extrude them and then right-click; click on the **Gstretch** button (or press *W* | **Specials** | **LoopTools** | **Gstretch**).
4. In the last operator panel at the bottom of the **Tool Shelf** (or else, press *F6* to make the pop-up window appear at the mouse cursor location), check the **Delete** strokes item.
5. Press *Ctrl* + *R* to cut the required edge-loops in the new faces:

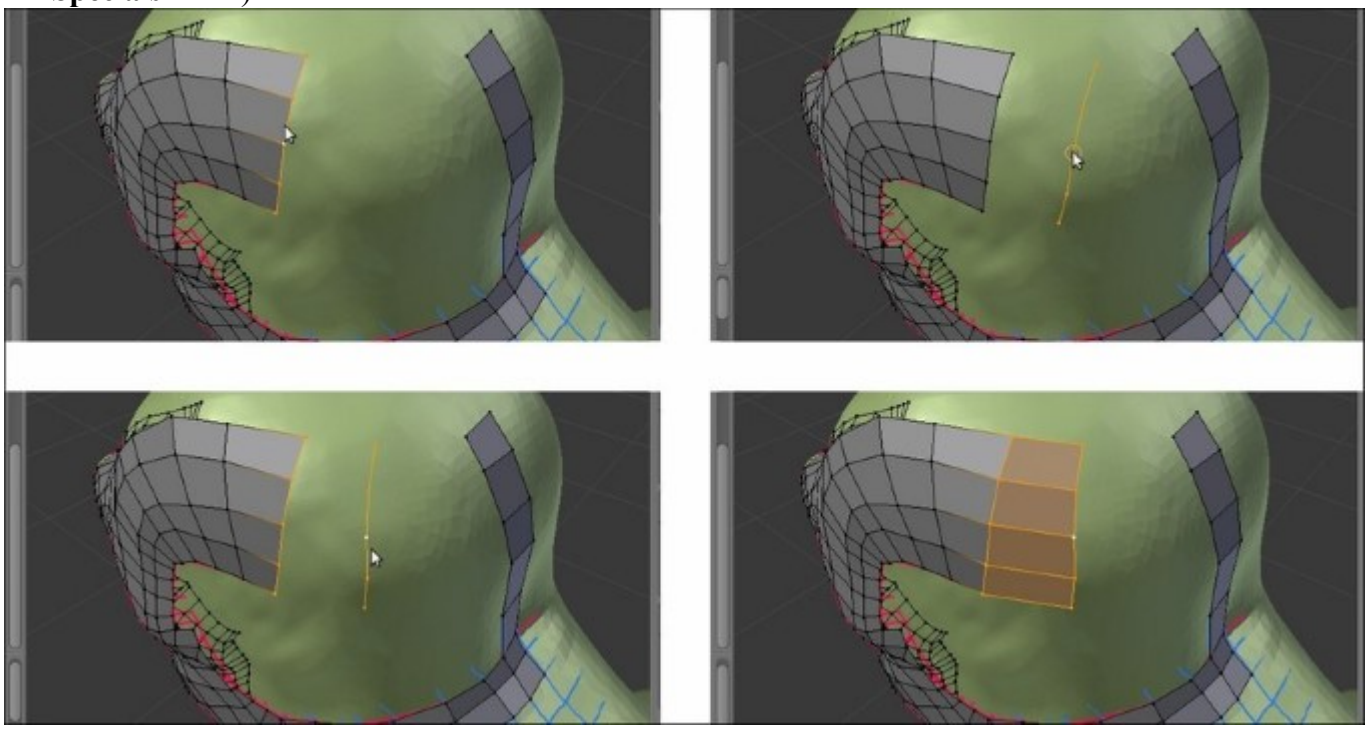


Using the Gstretch tool in conjunction with the Grease Pencil tool

Yes, it's that simple; it's enough to stroke the target position line and the new extruded vertices will be moved to that target position.

Also, now let's see the **Bridge** and the **Loft** tools:

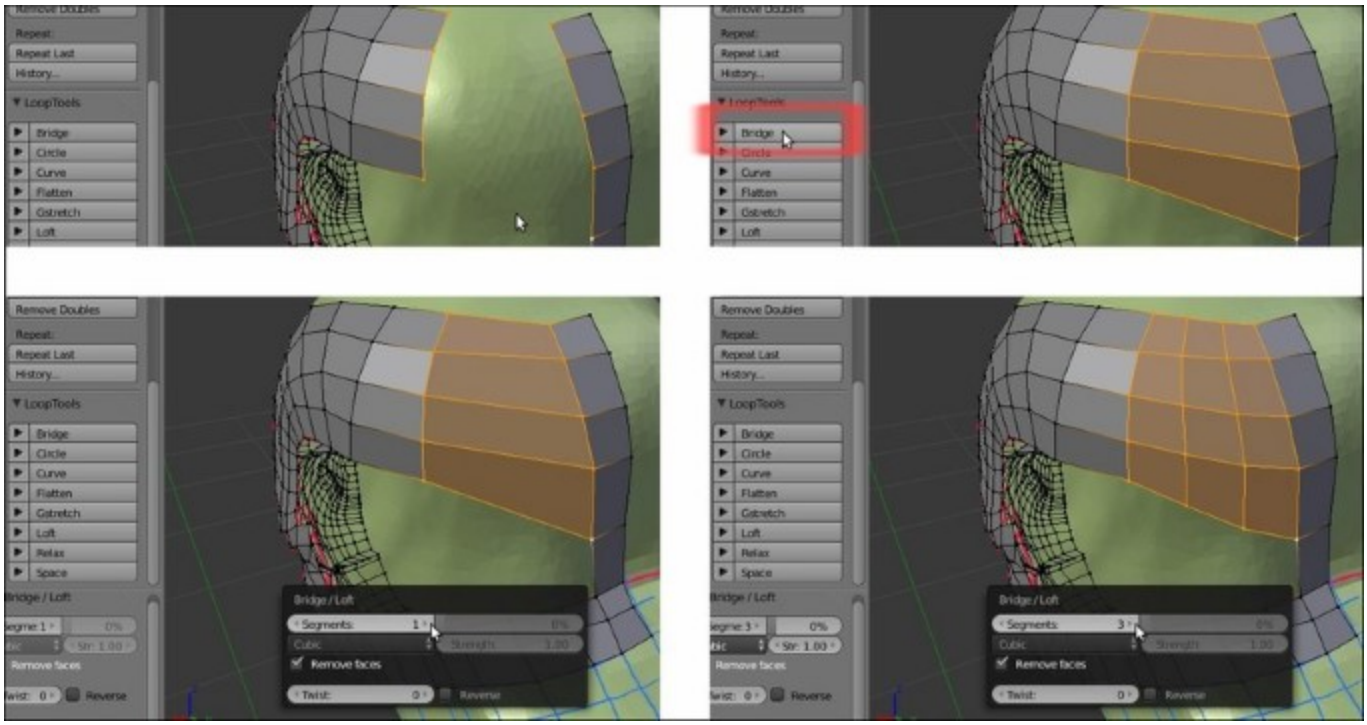
1. Select a group of edges and press *Shift* + *D* to duplicate them.
2. Move them into a new position and adjust the vertices as required.
3. Select both the new edges as the previous group.
4. Go to the **LoopTools** panel and click on the **Bridge** button (or again, through the *W* key to call the **Specials** menu).



Using the Bridge tool

5. If you need to add cuts, instead of the usual *Ctrl* + *R* shortcut, go to the last operator panel (*F6*) and change the value of the **Segments** slot to the number required.

It's not mandatory to duplicate new edges, it's enough to select the same number of vertices in the two edge-loops to be connected; here, after the **Bridge** tool operation, we have set the **Segments** value to **3**:

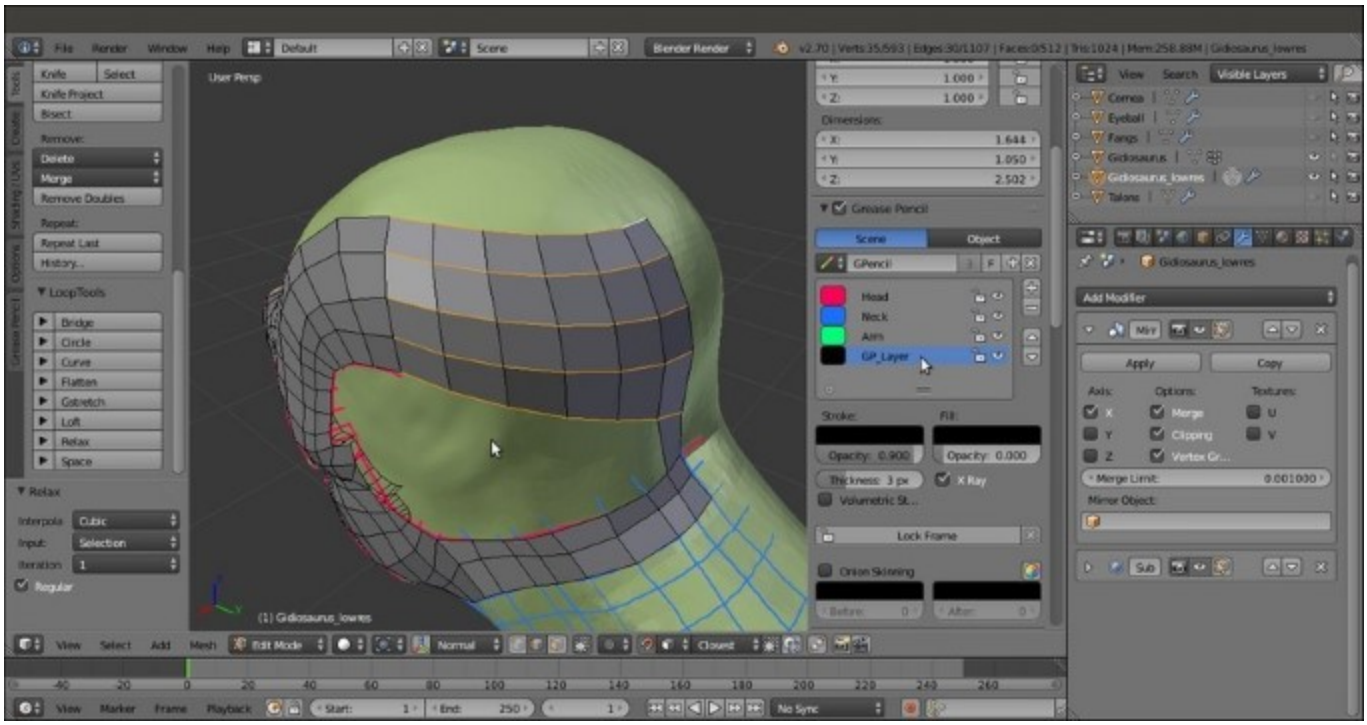


Adding cuts to the bridge operation

You can repeat the operation and the add-on will keep the last values you entered.

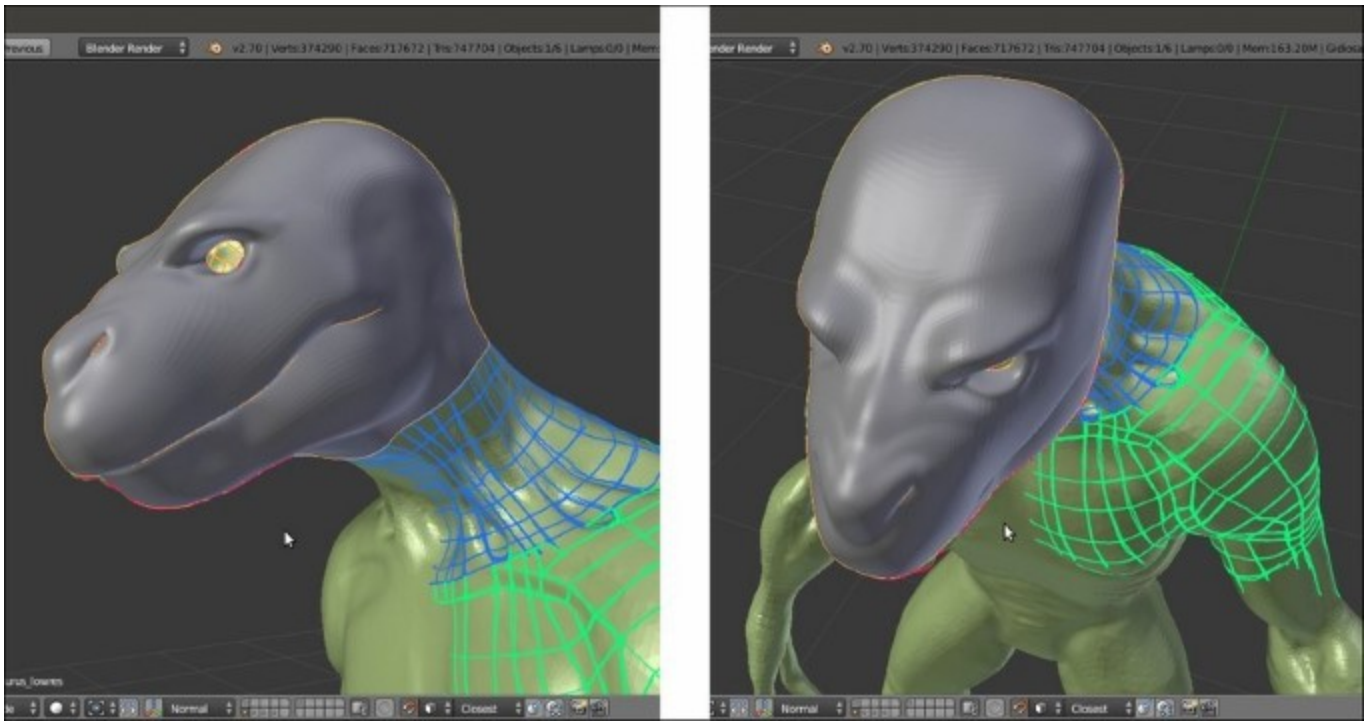
Repeat the steps, and this time click on the **Loft** button. The effect is almost the same, but if the new faces come out really messy, just click twice on the **Reverse** checkbox in the last operator panel; this should fix the issue.

You can then use all the other buttons to refine the added geometry; in the following screenshots, I tweaked the new geometry a little bit by selecting the horizontal edges and clicking on the **Space**, **Flatten**, and **Relax** buttons:



Completing the Gidiosaurus head

Using a mix of all the previous methods, in a short time, we have completed the head and the joining of the neck of our **Gidiosaurus_lowres** mesh; as you can see, particularly in the second screenshot at the bottom, the technique of following the main features and folders of the sculpted surface with the edge-loops can highlight the organic shapes even with a low resolution mesh:



The completed head

Don't forget to save the file and quit Blender.

Concluding the re-topologized mesh

The **Shrinkwrap** modifier method can be the way to quickly finish the rest of the re-topology of the **Gidiosaurus** sculpted mesh, by quickly re-topologizing the simpler cylindrical shapes and then completing the more difficult parts by hand.

Getting ready

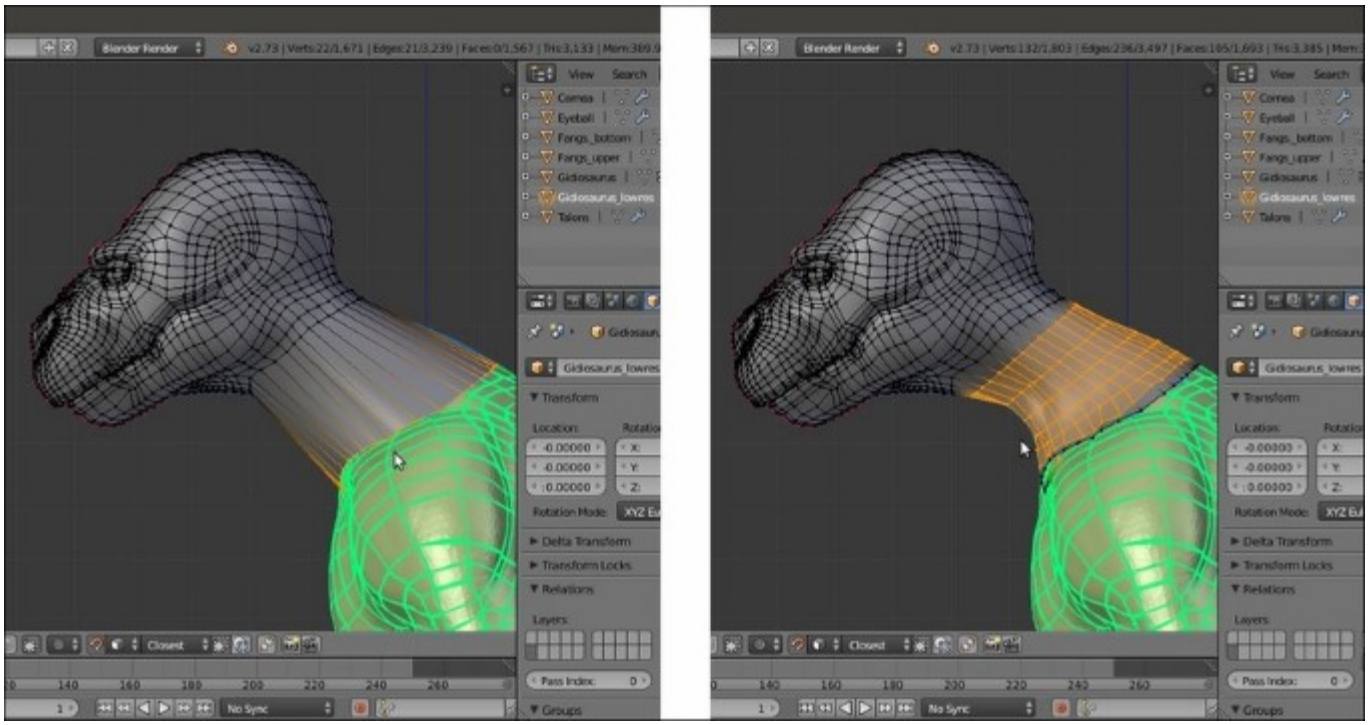
If necessary, repeat the steps to set up the **Shrinkwrap** modifier technique:

1. Assign the **Shrinkwrap** modifier to the **Gidiosaurus_lowres** mesh and in the modifier stack, move it *before* the **Subdivision Surface** modifier.
2. Click on the **Target** field to select the **Gidiosaurus** mesh item and leave **Mode** to **Nearest Surface Point**.
3. Enable the *Display modifier in Edit mode* and *Adjust edit cage to modifier result* buttons and the **Keep Above Surface** item.
4. In **Edit Mode**, to make the low resolution mesh more easily visible against the high resolution one, change the **Offset** value to **0.001**.
5. Having the **X-Ray** item still active, go to the **Shading** subpanel under the **Properties** 3D view sidepanel and check the **Backface Culling** item.
6. Then, to conclude the re-topology, we also need to enable the **Copy Attributes Menu** add-on; go to **Blender User Preferences** | **Addons** | **3D View** | **3D View: Copy Attributes Menu**.

How to do it…

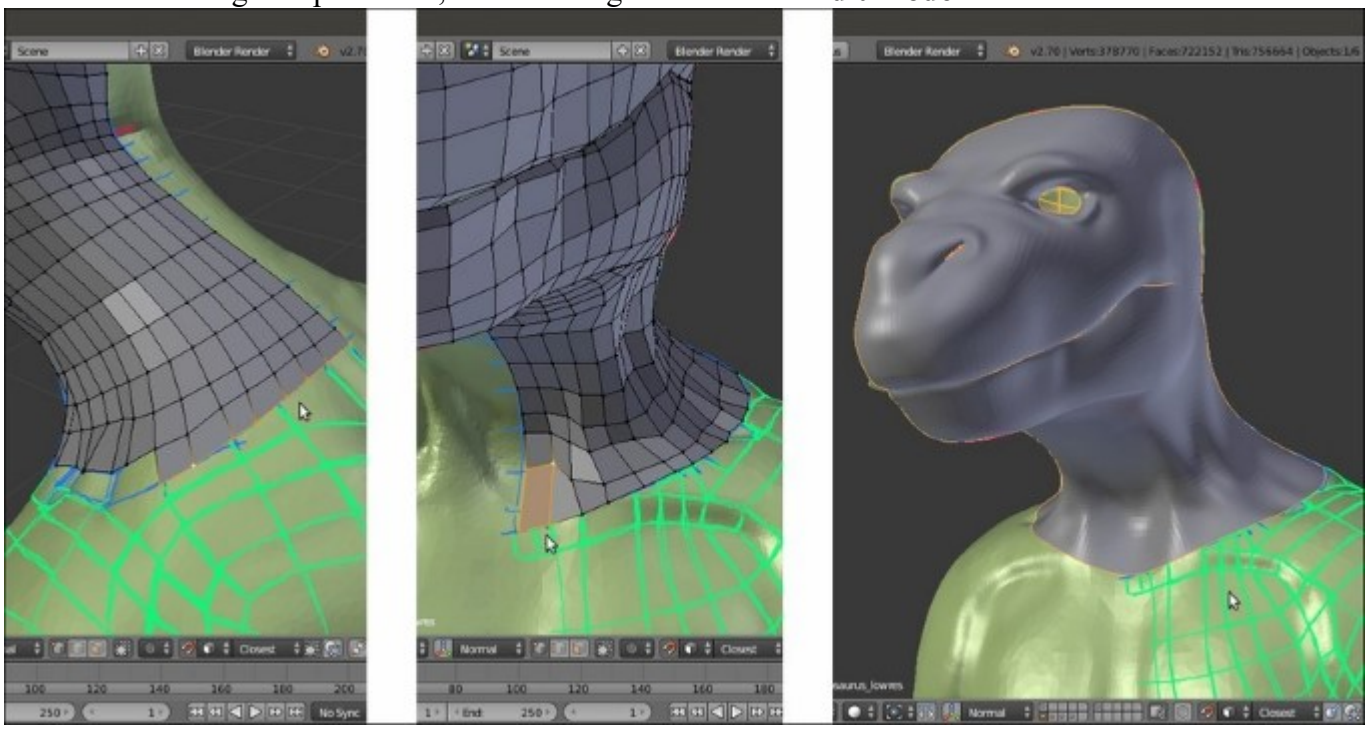
Let's go on by building the geometry of the **neck**:

1. While still in **Edit Mode**, just select the **head's** last edge loop on the **neck** and extrude it (*E* key) towards the **shoulders**.
2. Press the *Ctrl* + *R* keys and add at least **7** or **8** widthwise edge-loops:



Extruding the neck

3. Also, with the aid of the **Snap** tool, tweak the position of the bottom row of vertices, extrude them to add an edge-loop of faces, and tweak again. Go out of **Edit Mode**.



The re-topology of the neck

We can use the same technique as in steps 1, 2, and 3 to quickly re-topologize the left **arm** and **leg** of the character. Instead of extruding the new geometry from the **Gidiosaurus_lowres** mesh, in this case, it's better to add a new simple primitive: a **Circle** or also a **Plane**; whatever the primitive, when you add it, be sure that the **3D Cursor** is at the character's origin pivot point.

As you can see in the following screenshot, at first we just created the geometry only for the *main* cylindrical sections of the **limbs**:

Arms and legs re-topologized

Do the same for the **body**: just a couple of edge-loops placed at the **waist** to extrude the geometry from; remember that the **chest** is covered with the **armor breastplate**, so only the **exposed area** needs to be re-topologized.

One **Mirror** and one **Subdivision Surface** modifier has been assigned to the **three** objects (**head/neck**, **arm**, and **hips/leg**). Also, because of the **Mirror** modifier, the vertices of the half side of the **abdomen's** edge-loops have been deleted.

There's more…

After the main parts have been re-topologized, we can start to tweak the position of the vertices on the **arm** and **leg**, to better fit the flow and shapes of the muscles and tendons in the sculpted mesh.

Thanks to the aid of the **Shrinkwrap** modifier, we can do it quite freely; however, before we start with the tweaking, we require a little bit of preparation for a better visibility of the working objects, to affect and modify the new geometry (visible as a wireframe) and have the underlying sculpted mesh visible at the same time.

To do this, we have two ways:

The first way is as follows:

1. Go to the **Shrinkwrap** modifier panel and set the **Offset** value to **0.002**.
2. Go to the **Object** window and disable the **X-Ray** item; in the **Maximum Draw Type** slot, under the **Display** subpanel, select **Wire**:

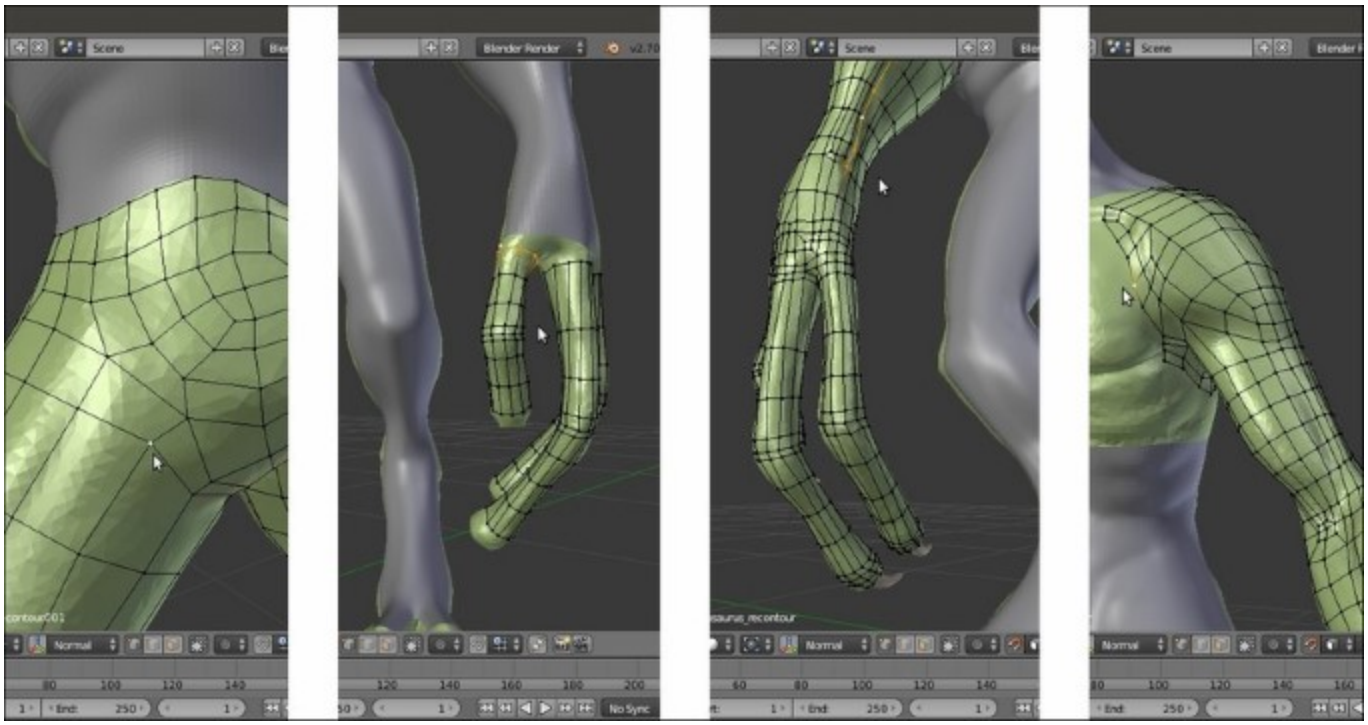
The mesh visualized in wireframe mode

The second way is as follows:

1. Go to the **Shrinkwrap** modifier and set the **Offset** value back to **0.000**.
2. If this is the case, go to the **Object** window and, under the **Display** subpanel, enable the **X-Ray** item. In the **Maximum Draw Type** slot, under the **Display** subpanel, select **Textured**.
3. Go to the **Properties** 3D view sidepanel (press *N* if not already present); if necessary, enter **Edit Mode** and under the **Shading** subpanel, check the **Hidden Wire** item.
4. In both ways (I used the second one), if you want to enable the *Display modifier in Edit mode* and *Adjust edit cage to modifier result* buttons for the **Subdivision Surface** modifier to see its effect in **Edit Mode**, it is better to move the **Shrinkwrap** modifier *after* the **Subdivision Surface** modifier in the stack, to have a better looking result.

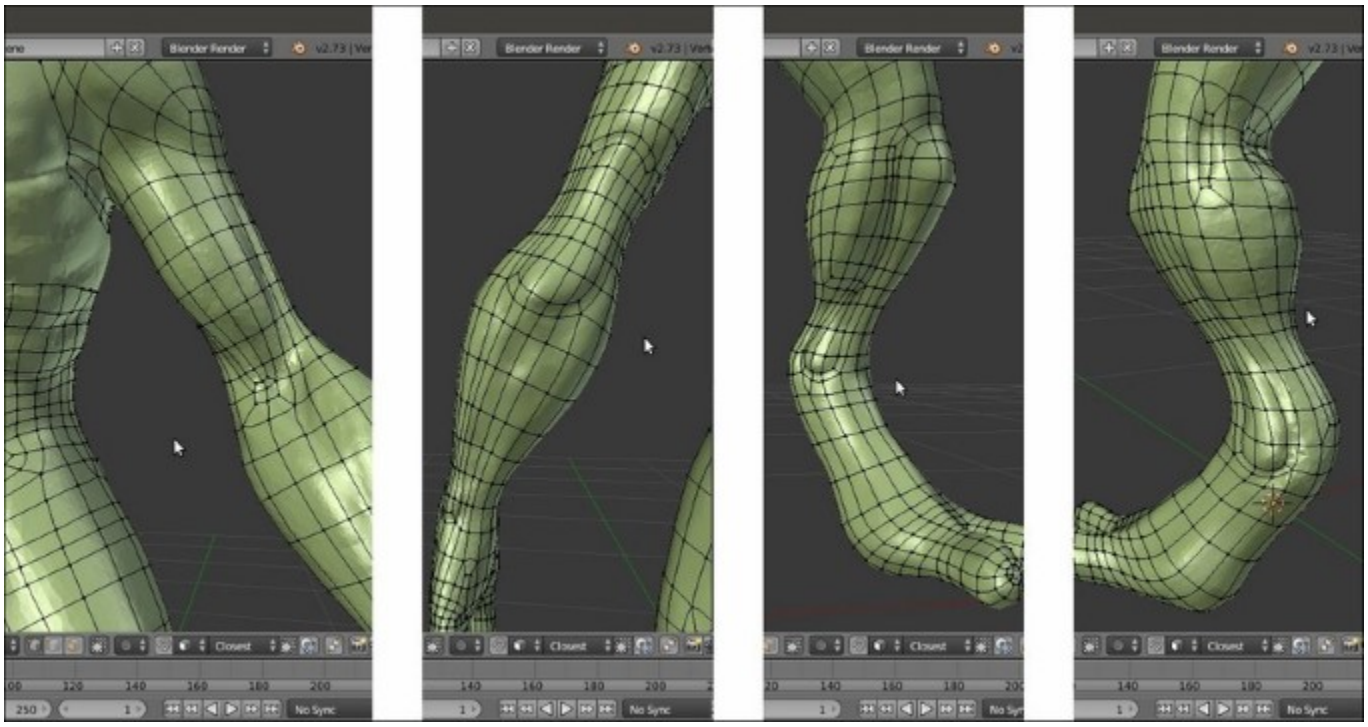
The second wireframe visualization method

We can now start to add the missing parts, by extruding and moving the vertices to better fit the sculpted features and also adding, if necessary, new edge-loops to better define these features:



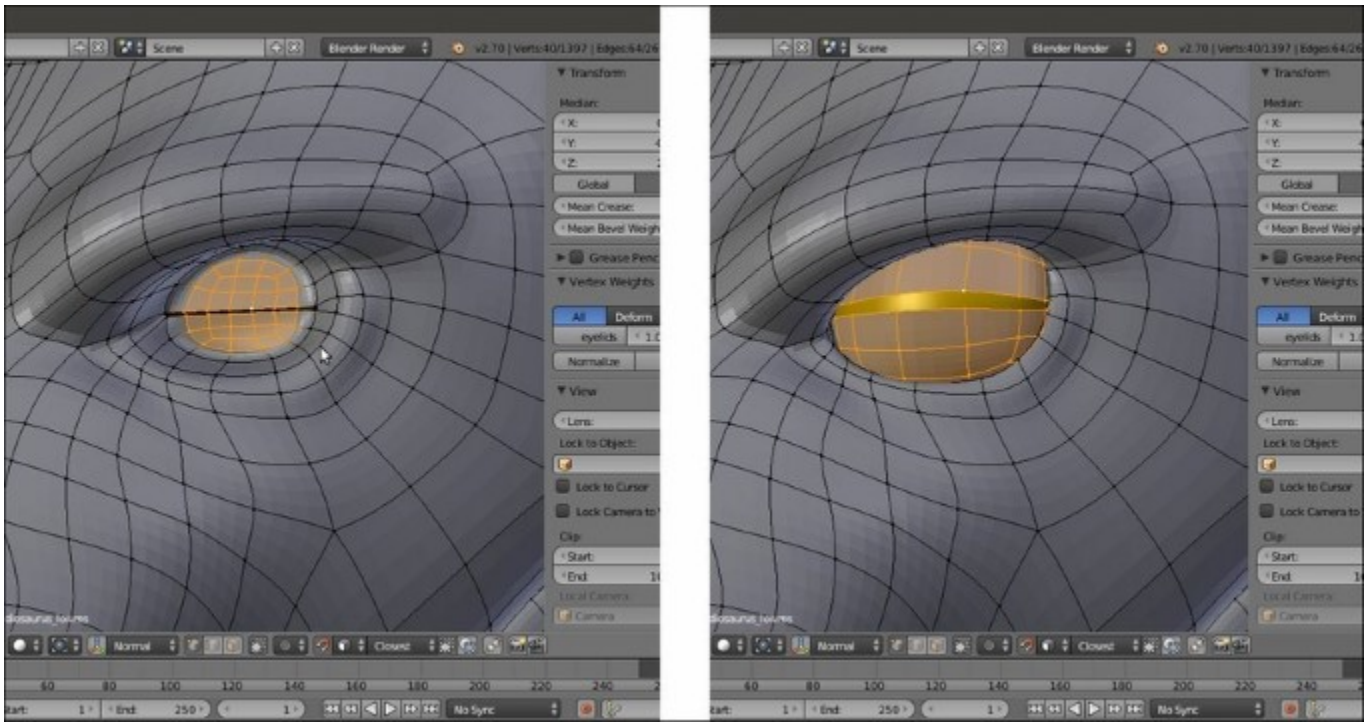
Refining and completing the remaining features

After the **wireframe** setup, it's easy to tweak the low resolution geometry to better fit the character's anatomy:



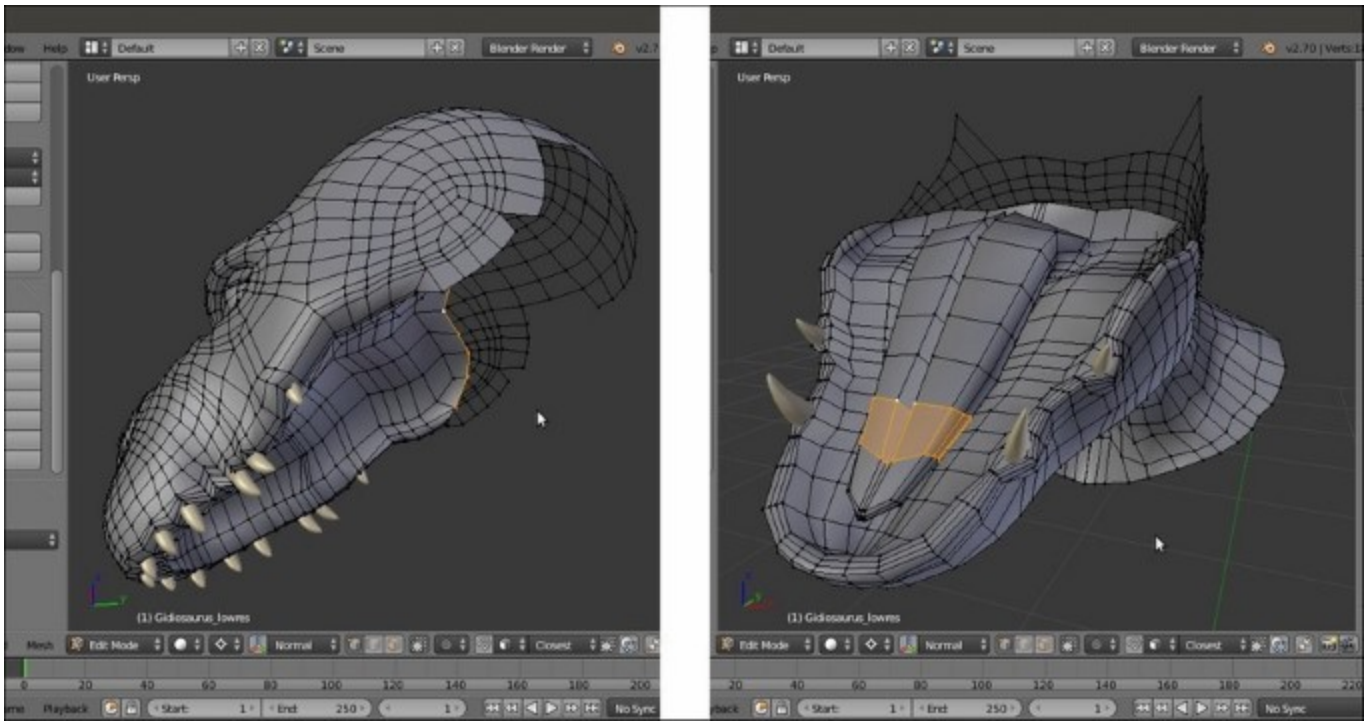
The character's anatomy

The still missing parts are modeled at this stage, such as the inside of the **nostrils** or the **eyelids**, again with the aid of the **Shrinkwrap** modifier; this time, targeted to the **Cornea** object to project the **eyelids** geometry onto it with an **Offset** value of **0.0035**:



The character's eyelids

Also, we built the **inner mouth** and the **tongue** of our character and refined the **dental alveoli**:



The character's alveoli and tongue

As in every project, we can go on with the refining, adding edge-loops, and so on, and this would seem a never-ending work; instead, at this point, we can consider the **Gidiosaurus** re-topology at the end, so it's time to apply the **Shrinkwrap** modifiers and, if this is the case, select the **Gidiosaurus** body's still separated objects and join them together to have a single mesh.

It's time to do the same with the **armor** that is still waiting on the **13th** scene layer:

The totally completed re-topologized character with the armor

How it works…

First, we have to quickly build the geometry using the **Shrinkwrap** modifier technique and then set the visibility of this geometry to wireframe (**Wire**), to make the underlying sculpted mesh visible.

The **Shrinkwrap** modifier, in the first case with the **Offset** value set high enough to allow the wireframe visibility over the sculpted surface, ensured that all the moved vertices and the new added geometry are automatically wrapped around the target mesh to preserve the volume.

At the end, we took back the **Offset** value to **0.000** anyway and we applied the **Shrinkwrap** modifier; then, we joined the re-topologized **arm** and **leg** objects together to the **Gidiosaurus_lowres** one.

As you have probably noticed, we haven't applied the **Mirror** modifiers yet. This is because it will still be useful in the next chapter.

Chapter 5. Unwrapping the Low Resolution Mesh

In this chapter, we will cover the following recipes:

- Preparing the low resolution mesh for unwrapping
- UV unwrapping the mesh
- Editing the UV islands
- Using the Smart UV Project tool
- Modifying the mesh and the UV islands
- Setting up additional UV layers
- Exporting the UV Map layout

Introduction

So, at this point, we have **sculpted** our high resolution character and through the **retopology** process, we have obtained a low resolution *copy*, which is easier to use for rigging, texturing, and animation.

There are several ways to apply textures to a mesh in Blender, as in any other 3D package. In our case, we are going to use **UV Mapping**, which is certainly one of the most commonly used and efficient methods for organic shapes.

Before the unwrapping process, the mesh must be prepared to make the task easier.

Preparing the low resolution mesh for unwrapping

In this recipe, we'll fix the last details such as the position of some of the character's parts (for instance, the **closed mouth**) and in general, anything that is needed to facilitate the unwrapping.

Getting ready

To be more precise, before the unwrapping, we must perform the following tasks in the right order:

1. Join the **teeth** and **talons** to the **body**.
2. Create the vertex group for the **mandible**.
3. Open the **mouth**.
4. Mark the seams to unwrap the **body**.

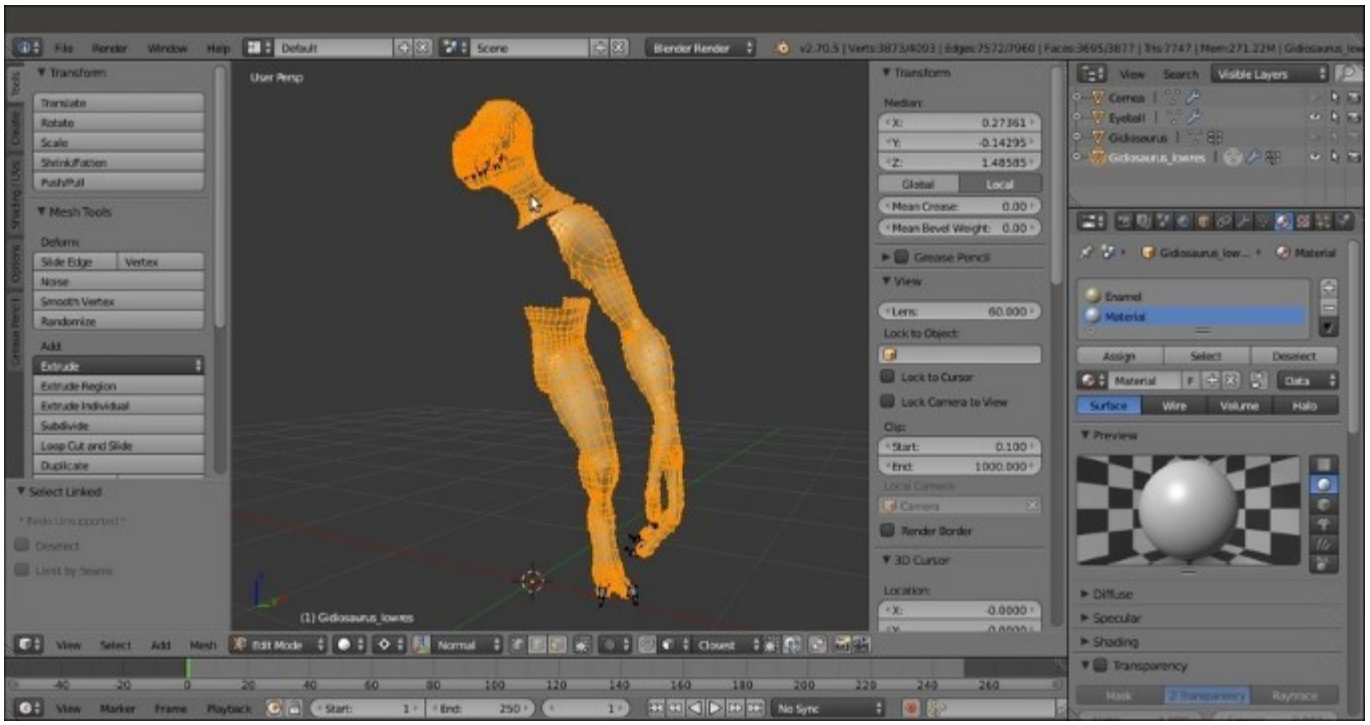
So, open the `Gidiosaurus_retopology.blend` file and deactivate the layer with the **armor** to hide it; select the **Gidiosaurus** object and save the file as `Gidiosaurus_unwrap.blend`.

How to do it…

The simplest of the *four* tasks just so happens to be the first, joining the **body** with the **teeth** and **talons**.

To join the body parts, follow these steps:

1. Select the **Talons** item in the **Outliner**, and then hold *Shift* and select the **Fangs_bottom**, **Fangs_upper**, and **Gidiosaurus_lowres** items.
2. Press *Ctrl* + *J* to join them.
3. Right away we will notice that, because the retopologized mesh didn't have any material assigned, the whole object gets the only material available, which is the `Enamel` material we had assigned to the **talons** and **teeth** earlier.
4. To fix this, assign a new material, or you can also assign the already existing `Body` material, to the retopologized mesh before the joining operation.
5. Alternatively, after the joining, click on the + icon to the side of the material names, and then select the **New** button in the **Material** window to create a new material. Now, enter **Edit Mode**, put the mouse pointer on the **Gidiosaurus** mesh, and press the *L* key to select all the **connected vertices**. Because the **talons** and **teeth** vertices are joined, *but not connected* to the **face** vertices, they don't get selected; for the same reason, you have to repeat the operation three times to select the **head**, **arm**, **hips,** and **leg** vertices:

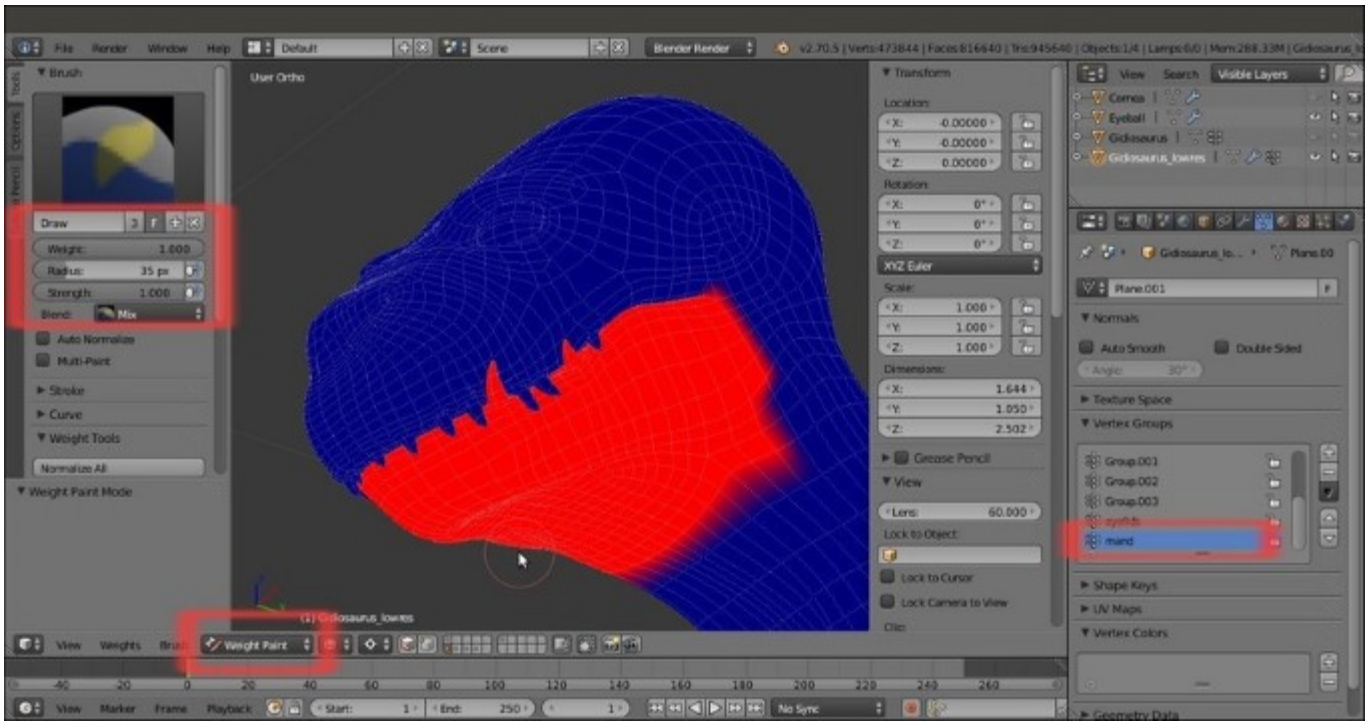


The head, arm and hip/leg vertices selected in Edit Mode

6. Click on the **Assign** button and go out of the **Edit Mode**. Now, edit the name and color of the new material or whatever, or else switch it with the `Body` one.

   The second task is a bit more complex and is covered in more detail in Chapter 7, *Skinning the Low Resolution Mesh*, which is about the **skinning** process. However, we need to explore this subject a little bit now, as it will help us operate on a small portion of the mesh easily.

   To create a **vertex group** to open the **mouth**, follow these steps:
7. Go to the **Side** view and zoom in on the **head** of the character.
8. Go to the **Object Data** window; under the **Vertex Groups** subpanel, add a new group and rename it `mand` (short for **mandible**).
9. Press *Ctrl* + *Tab* to go into **Weight Paint** mode (or left-click on the mode button on the 3D window toolbar to switch from **Edit Mode** to **Weight Paint** mode); press *Z* to go into **Wireframe** viewport shading mode so that you can see the edges of the topology.
10. By using a combination of vertex selection mode, both in **Edit Mode** and by painting with **Weight** and **Strength** as **1.000** in the **Weight Paint** mode, assign vertices to the group of the **mandible** area and the part of the **neck**; obviously, you have to include the vertices of the inner bottom **jaw**, as well as the **tongue** and bottom **teeth**:

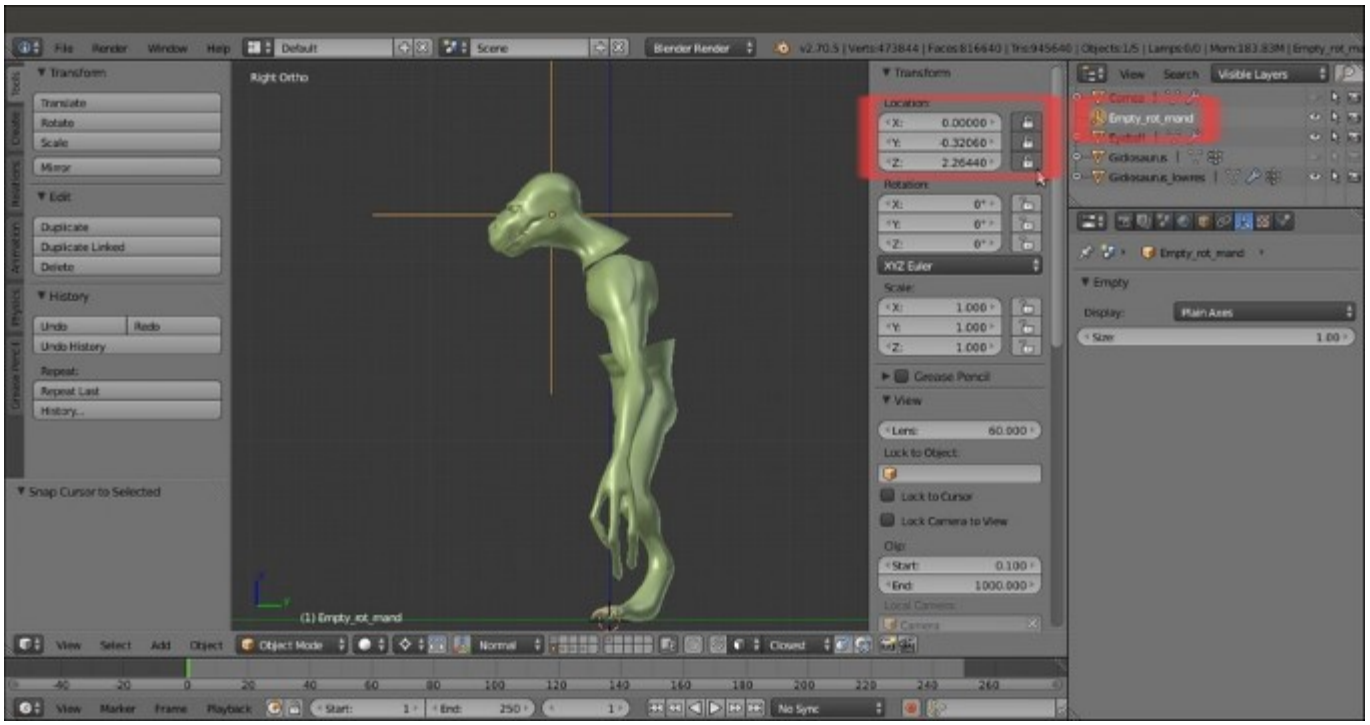


The visualization of the mand vertex group

11. Press *Ctrl* + *Tab* to exit the **Weight Paint** mode.

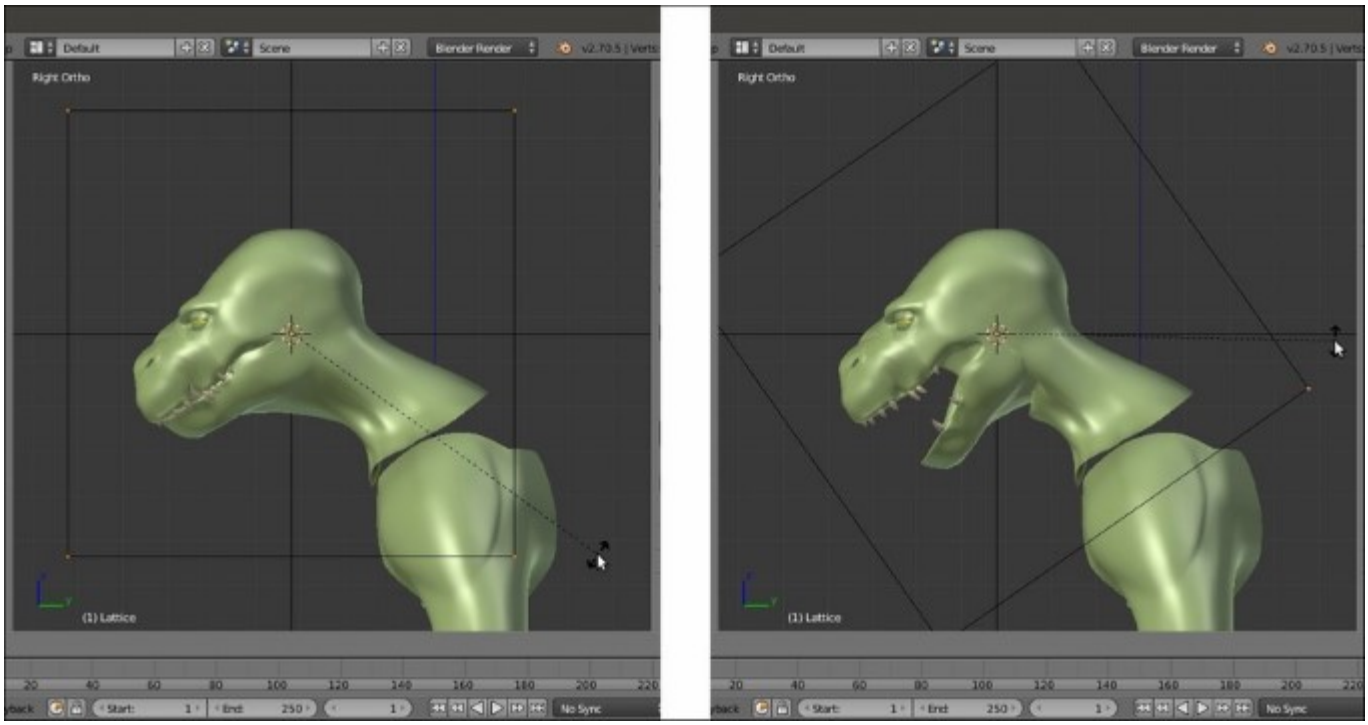
    Note that a vertex group can be edited at a later time, so it will be easier to set the exact amount of weight on the vertices by looking at the **Lattice** modifier feedback, which is the next step.

    So, to open the **mouth**, perform the following steps:
12. Add an **Empty** object to the center of the scene (*Shift* + *A* | **Empty** | **Plain Axes**).
13. Go to the **Side** view (press the *3* key on the numpad). Move the **Empty** to the position where the **mandible** should join the skull (to be precise, I placed it at this location: **X** = **0.0000**, **Y** = **-0.3206**, and **Z** = **2.2644**; go to the **Properties** 3D view sidepanel, and under the **Transform** subpanel, enter the values in the first three slots under the **Location** item).
14. To ensure that the **Empty** cannot be moved anymore, click on the lock icon on the right-hand side of its slot in the **Outliner** and also rename it to `Empty_rot_mand`:



The Empty_rot_mand in place

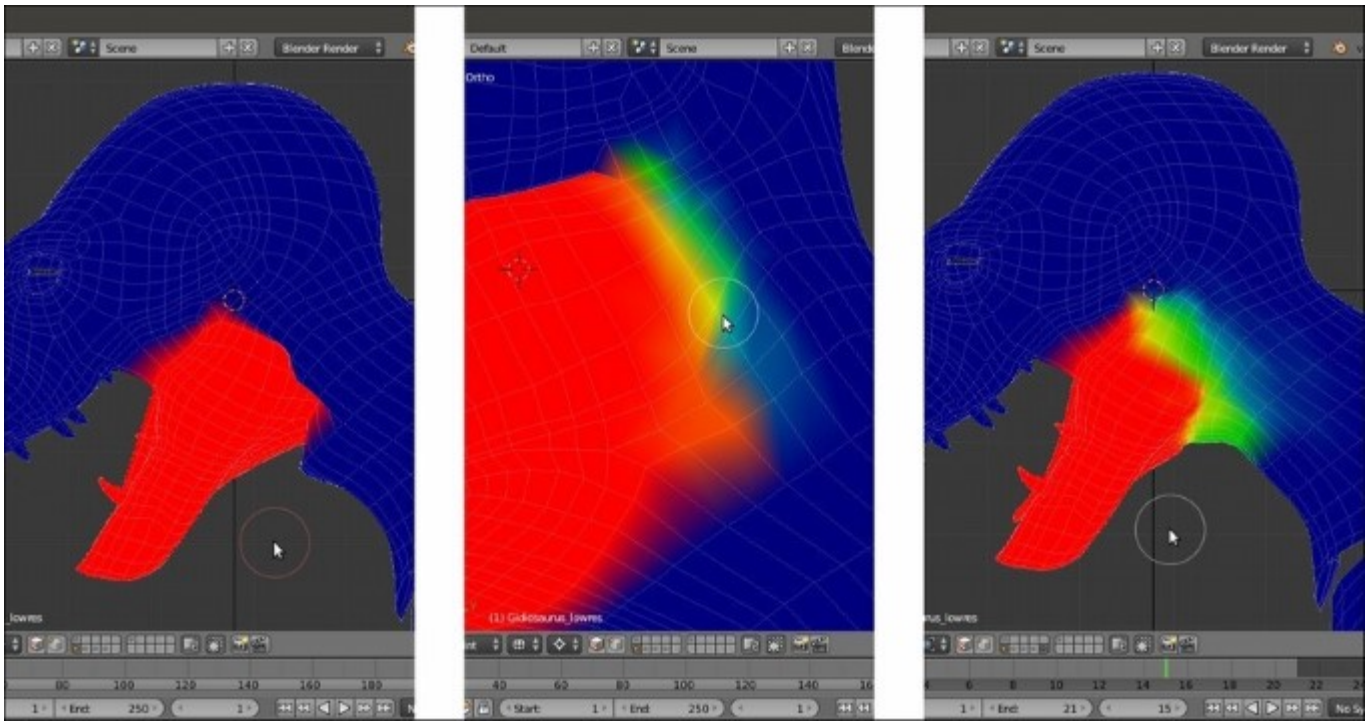
15. With the **Empty** still selected, press *Shift* + *S* | **Cursor to Selected**.
16. Add a **Lattice** object to the scene (*Shift* + *A* | **Lattice**), and in the **Object Data** window, set **Interpolation Type** for **U**, **V**, and **W** to **Linear**; select the **Gidiosaurus** object and go to the **Objects Modifier** window; assign a **Lattice** modifier. Move it before the **Subdivision Surface** modifier.
17. In the **Object** field, select the **Lattice** item; in the **Vertex Group** field, select the **mand** item.
18. In the **Side** view, select the **Lattice** object, go into **Edit Mode**, and select all the vertices and rotate them **35** degrees counterclockwise around the *x* axis:



Rotating the Lattice to open the mouth

As you can see, the **Lattice** only affects the vertices inside the **mand** vertex group; however, there is a clear indentation on the throat where the **mand** vertex group ends abruptly, so now we must blur this boundary to keep the smooth curved transition from the bottom **jaw** to the **neck**, and remove the abrupt edge.

19. Go back into the **Weight Paint** mode (*Ctrl* + *Tab*) and click on the **Brush** icon at the top of the **Tools** tab to switch the **Draw** brush with the **Blur** brush, and then start to blur the boundaries of the **mand** vertex group.
20. Sometimes, blurring the edge weights is not enough, so go back to the **Draw** brush, set the **Strength** to **0.500** (or whatever value you find works best), and paint on the vertices; then refine the transition again with the **Blur** brush:

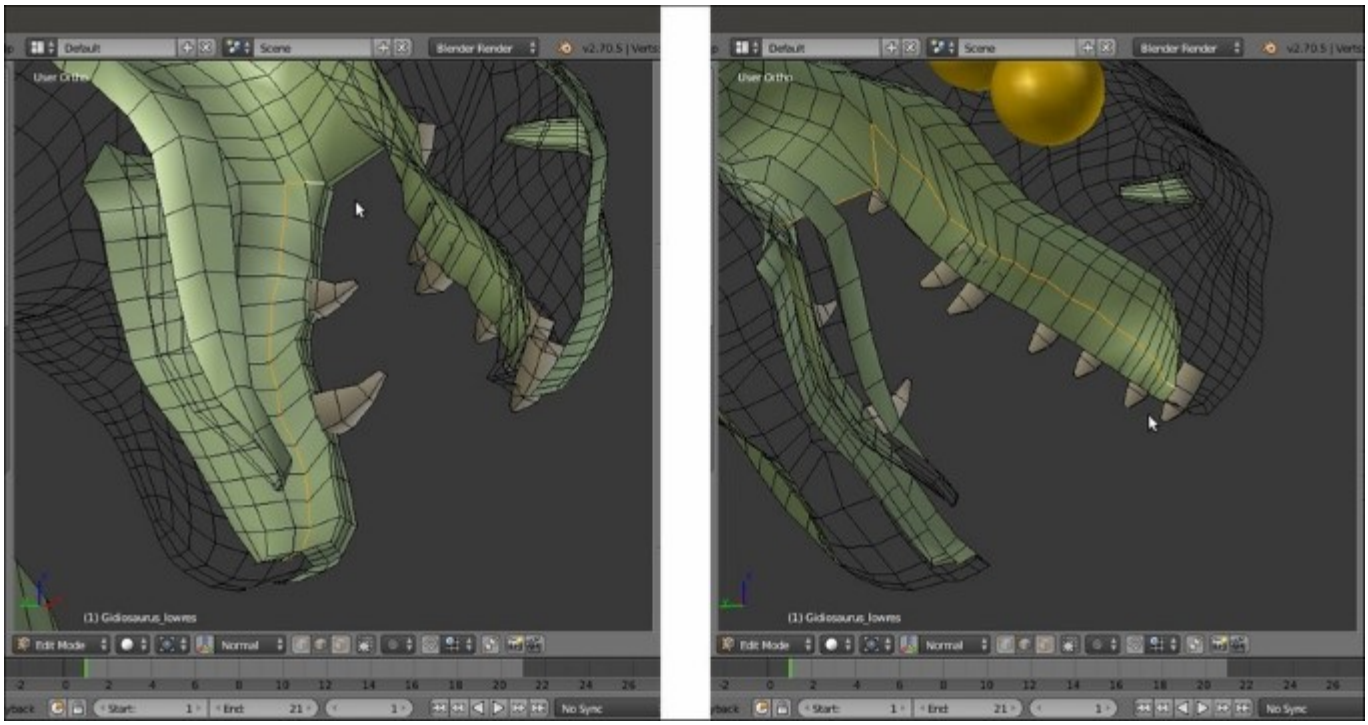


Blurring and painting the weights

21. To make the job easier and faster, you can temporarily disable the **Lattice** modifier, as well as the **Subdivision Surface** modifier.
22. When you are done, go out of the **Weight Paint** mode, apply the **Lattice** modifier, and delete the **Lattice** object.
23. Make sure to keep the **Empty_rot_mand**, which that will turn out to be useful when rigging the character. For now, just hide or move it onto a different layer.

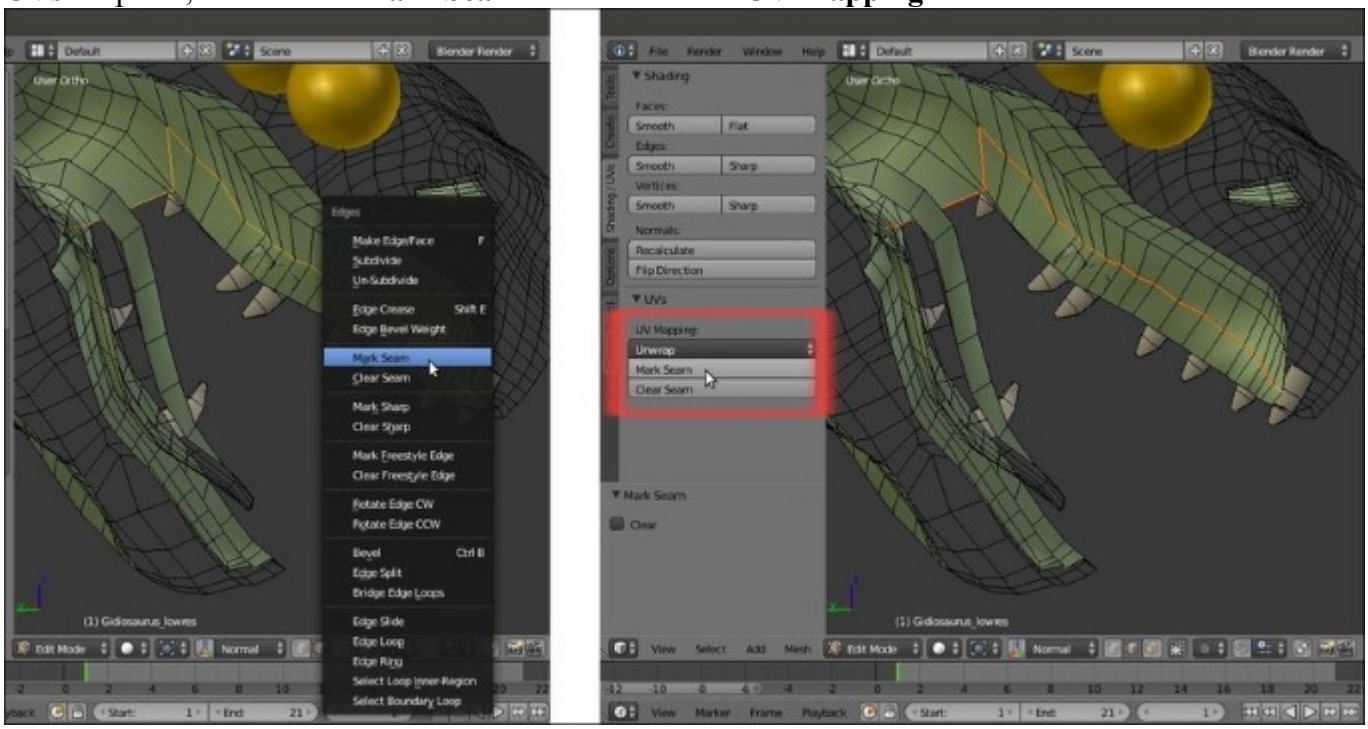
    At this point, we can obviously edit the **throat** area vertices as usual: relaxing and tweaking them and so on. Actually, this is the right moment to tweak all the vertices and any areas that couldn't be done before, such as the inside of the **mouth**, the inner **cheeks**, and so forth, because now we are going to do the last preparation task before the unwrapping.

    To mark seams for the unwrapping of the body, we have to perform the following steps. Because our low resolution mesh is actually still only one half side, we don't need to place seams as median cuts, we only need to divide different areas (for example, the **inside of the mouth** from the **outside of the mouth**) and unroll cylindrical parts such as the **arms**, **fingers**, and **teeth**:
24. Go into **Edit Mode** and zoom in to the character's **head**; press *Ctrl* + *Tab* to call the **Mesh Select Mode** pop-up menu and select the **Edge** item, and then start to select the edge-loop inside the **mouth** (*Alt* + right-click to select an edge-loop); start from the bottom **jaw**, switching direction at the end of the **mouth rim** to go upward, and finish on the inside of the upper **jaw**:



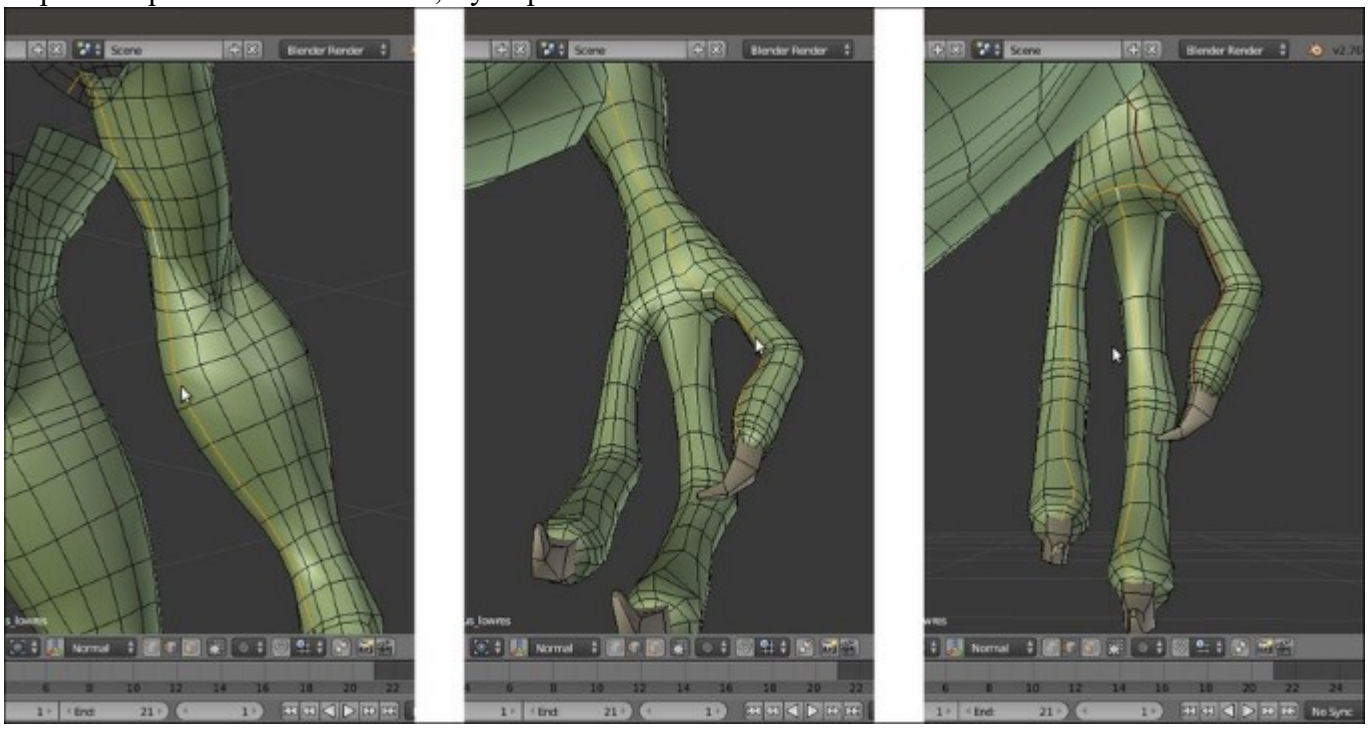
The selected edge-loops inside the mouth

25. Press *Ctrl + E* to open the **Edges** pop-up menu and select the **Mark Seam** item. Alternatively, click on the **Shading / UVs** tab in the **Tool Shelf**, to the left-hand side of the screen, and in the **UVs** subpanel, click on the **Mark Seam** button under the **UV Mapping** item:



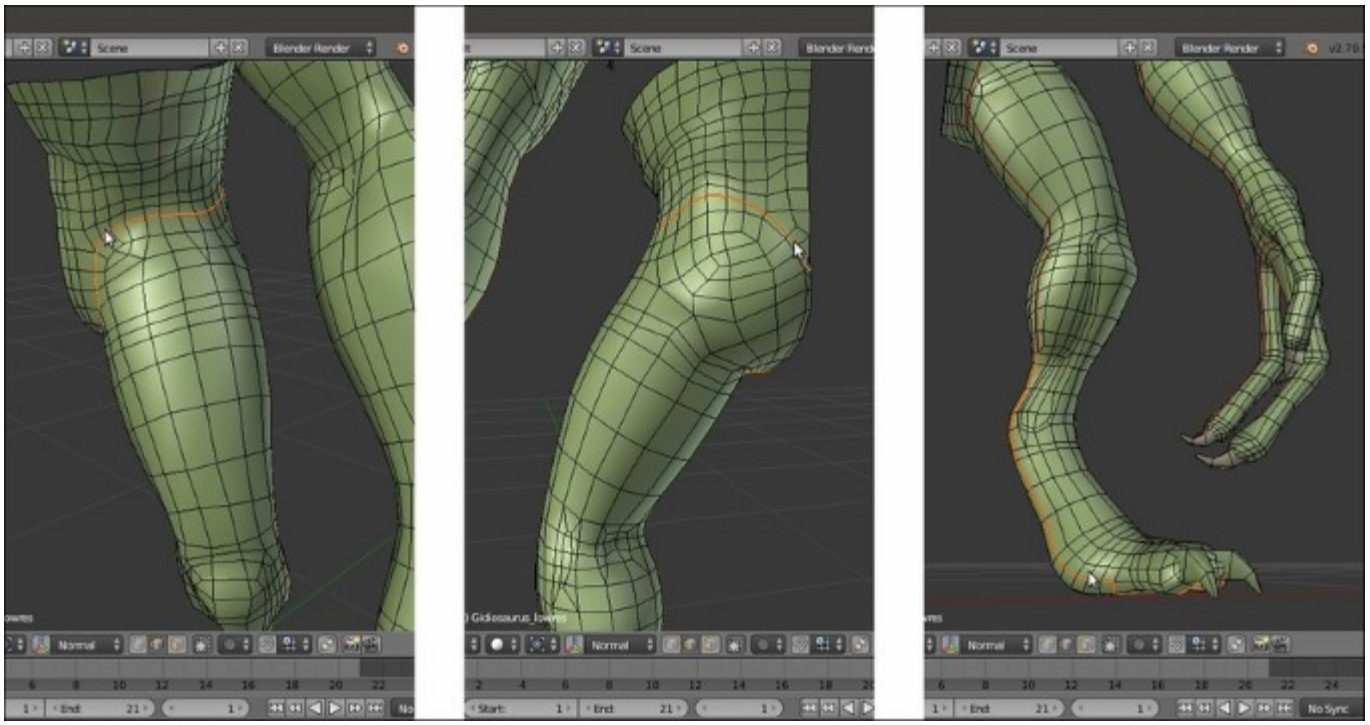
Marking the seams

26. Repeat the procedure for the **arm**; try to place the seams in the less visible areas:



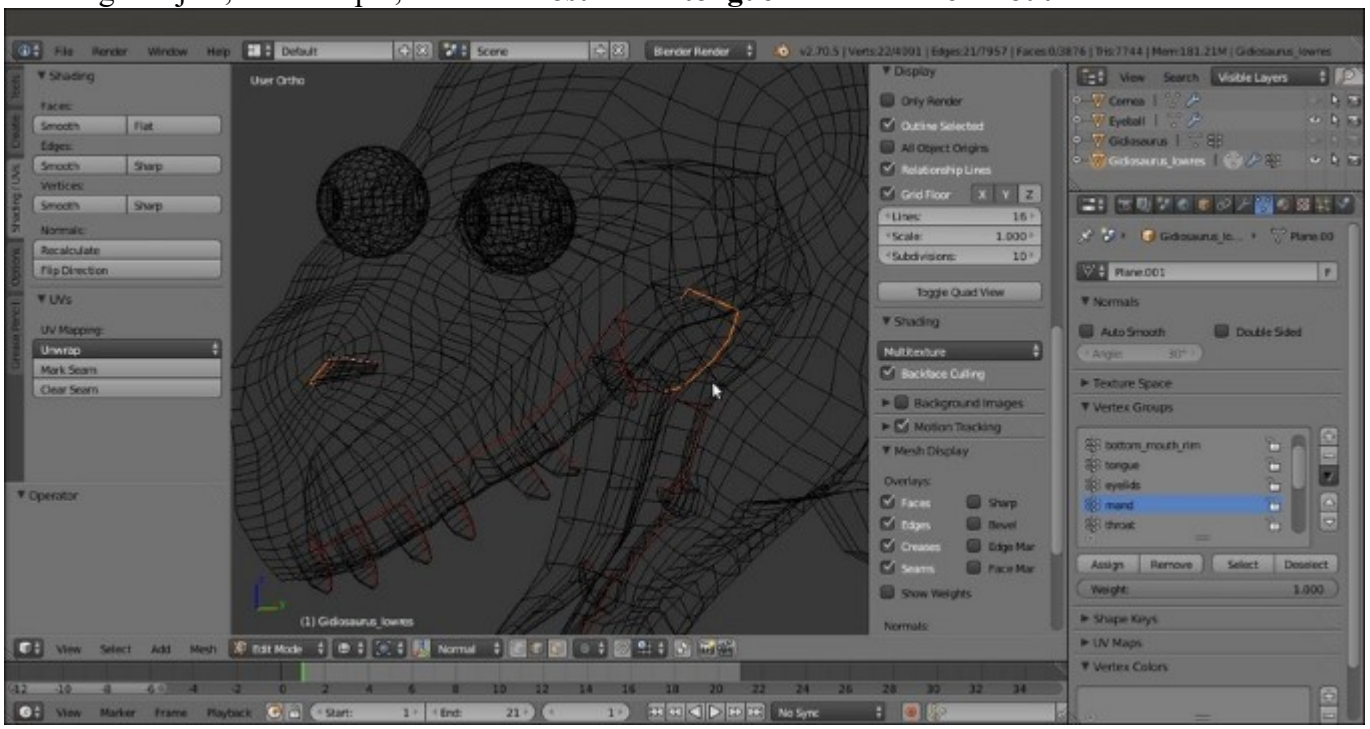
The seams on the arm

27. Do the same for the **pelvis** and **leg**; divide them into two parts with the seams and also try to place the seams inside the natural body folds, if possible:



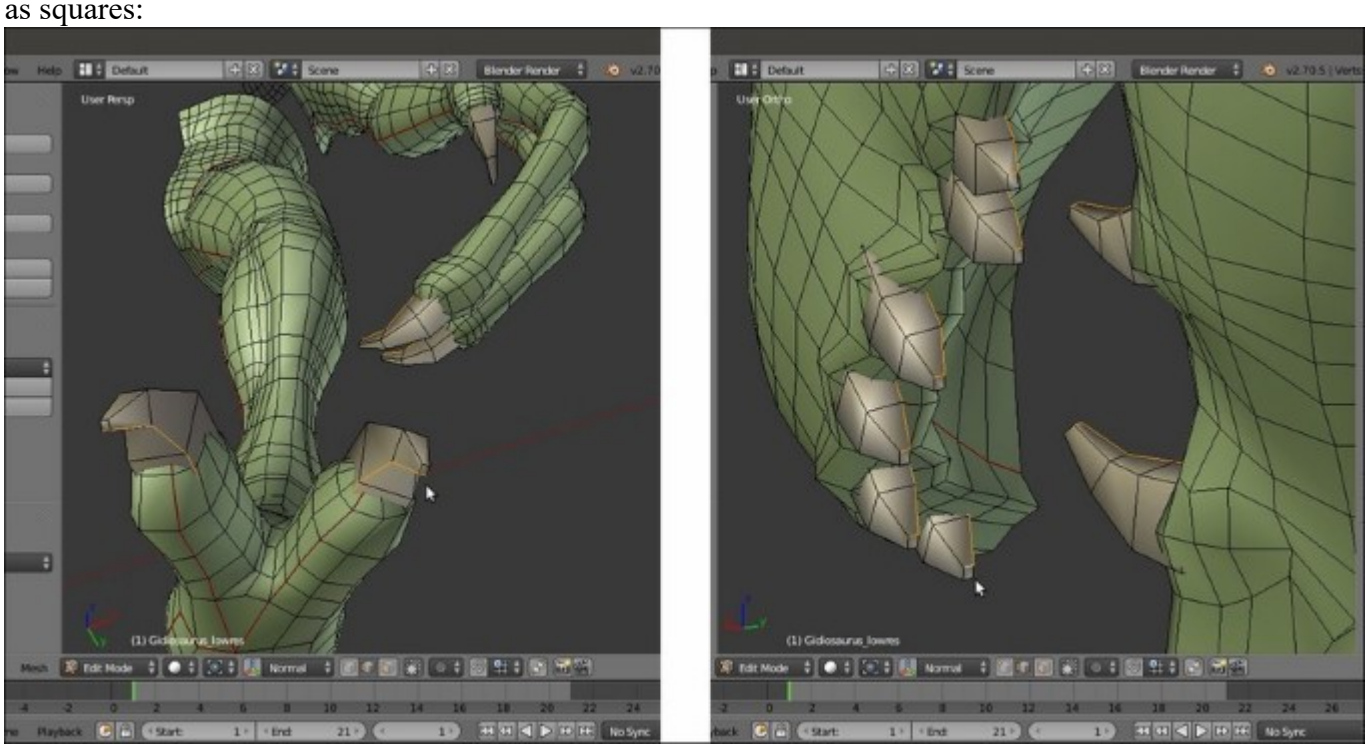
The seams on the pelvis/leg parts

28. It is important to try to place the seams to *divide* parts that would get unwrapped badly if treated as a single object; for example, the inner **nostril** and **tongue** from the **inner mouth**:



The seams inside the head

29. The final seams to add are for the teeth and talons, which would otherwise get badly unwrapped as squares:



The seams of the small parts

30. Save the file.

UV unwrapping the mesh

At this point, everything is ready for the unwrapping.

Getting ready

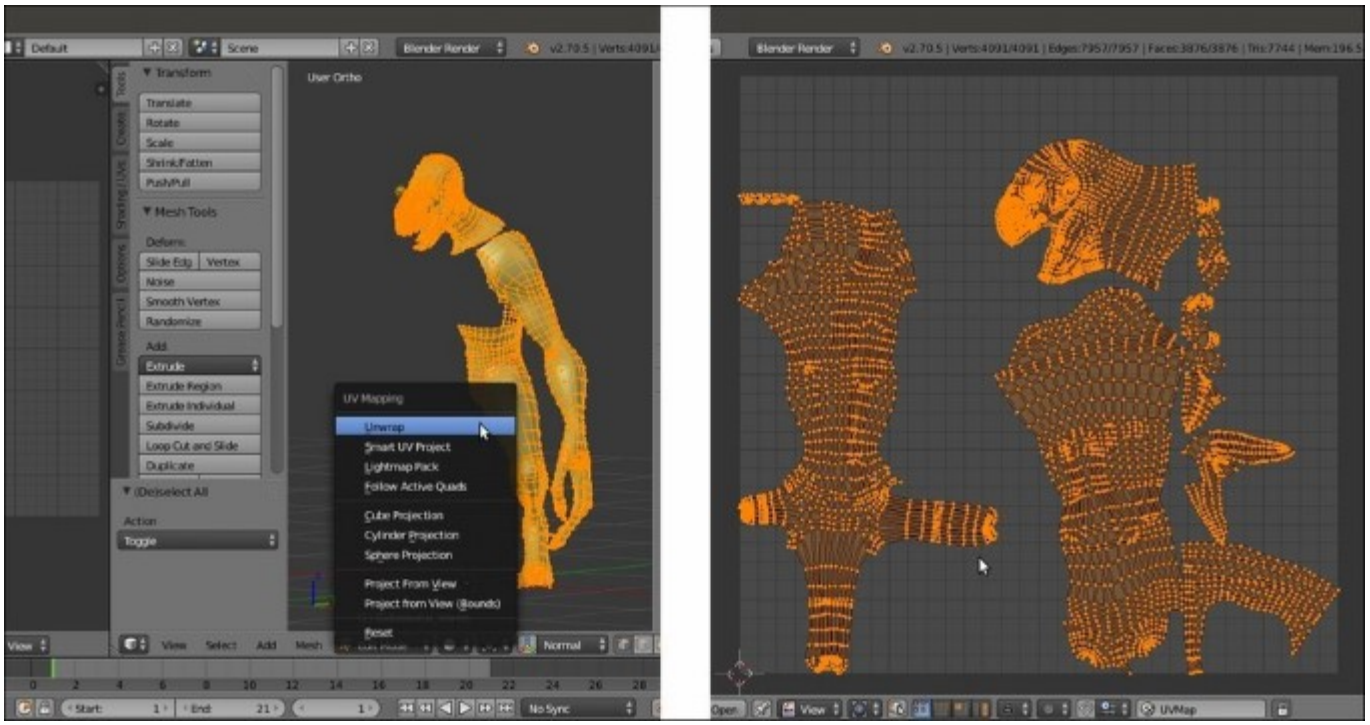
Put the mouse pointer on the bottom or on the top horizontal borders of the 3D window. As the mouse pointer changes to a double-arrow icon, right-click and in the **Area Options** pop-up menu select the **Split Area** item; then, left-click to obtain two windows and switch the left one to **UV/Image Editor**.

The two windows

How to do it…

To unwrap the mesh in Blender, several options are available; however, the one we are going to use now is the basic unwrap, the result of which we will edit and refine later:

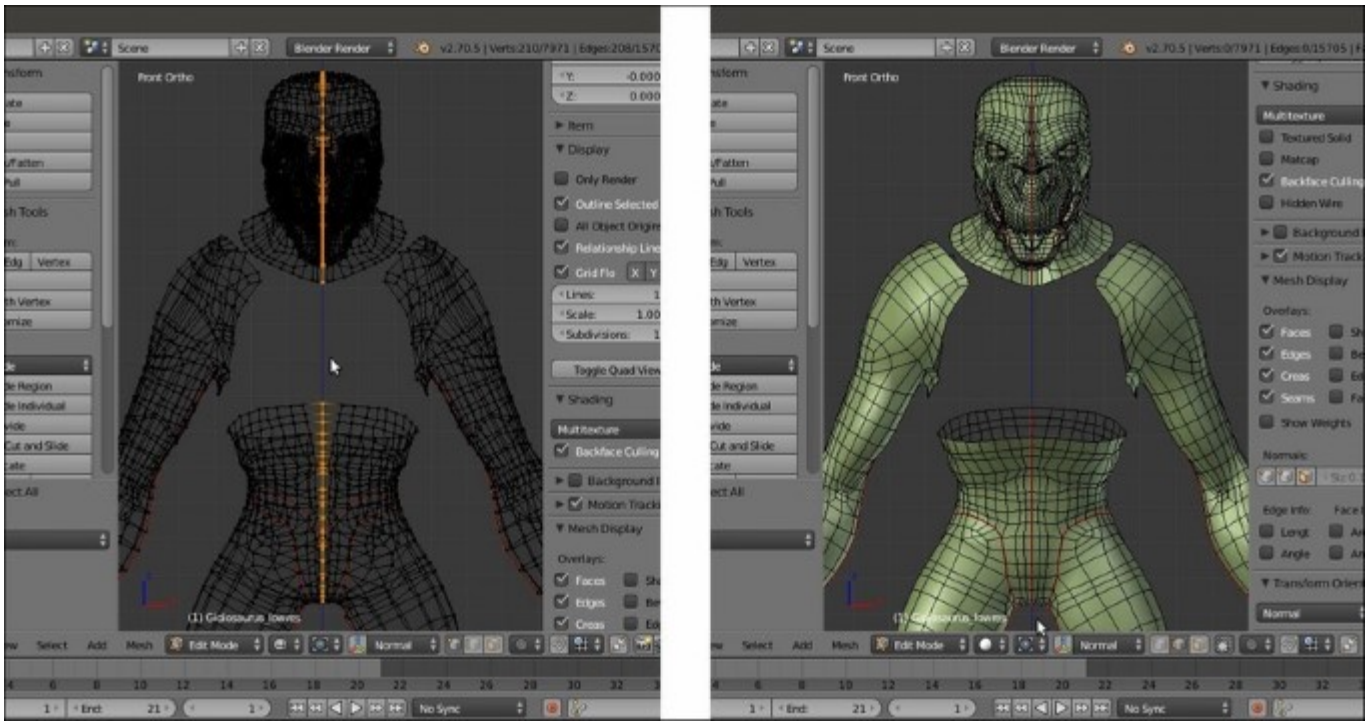
1. Ensure that the **UV/Image Editor** window is not set to **Render Result**, otherwise it won't display the **UV islands**.
2. Select the **Gidiosaurus_lowres** object and enter **Edit Mode**. Select all the vertices (*A* key) and press the *U* key; in the **UV Mapping** pop-up menu, select the first item, **Unwrap**:



Unwrapping the mesh

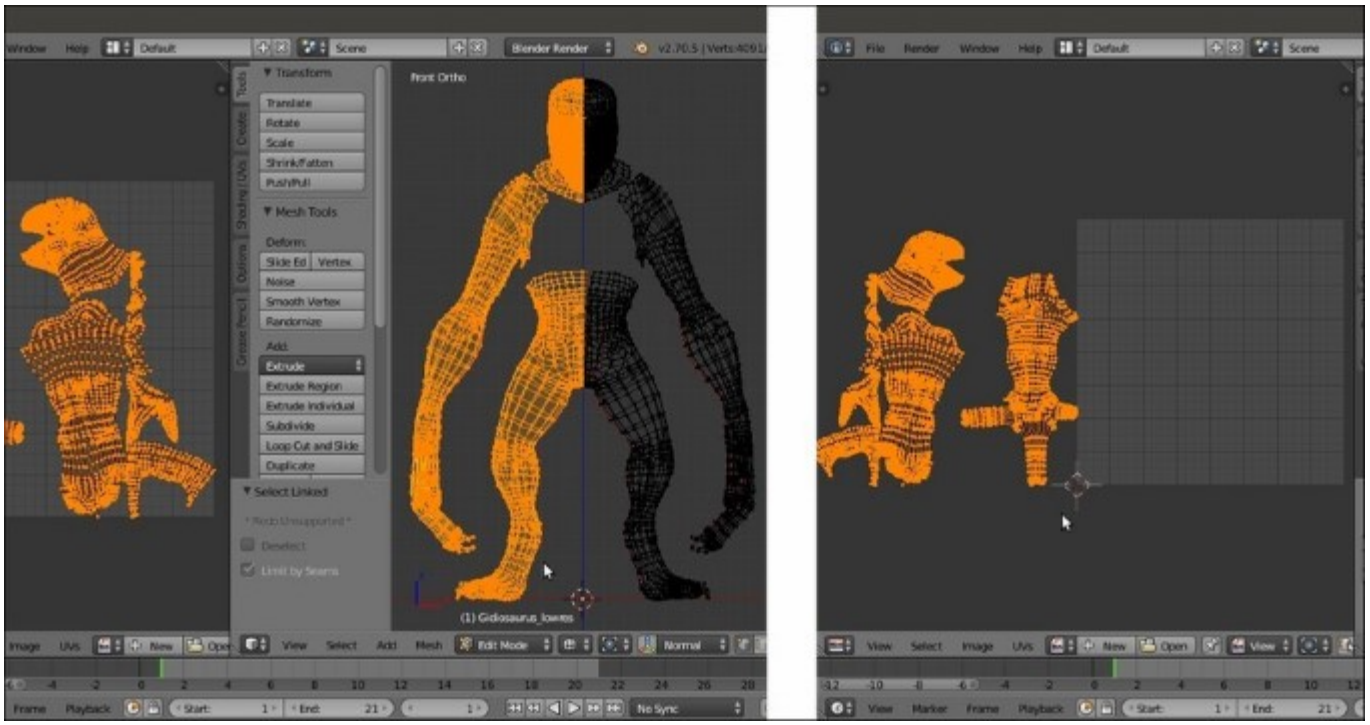
After a while, the **UV layer** of the unwrapped mesh appears in the **UV/Image Editor** window; as you can see, several things can be improved. Moreover, we are still using only half of a mesh.

3. Go out of the **Edit Mode** and go to the **Object Modifiers** window; apply the **Mirror** modifier.
4. Go back into **Edit Mode** and press *1* on the numpad to go to the **Front** view; press *Ctrl + R* and place a median seam through the **head** part of the mesh, as well as through the **pelvis** part:



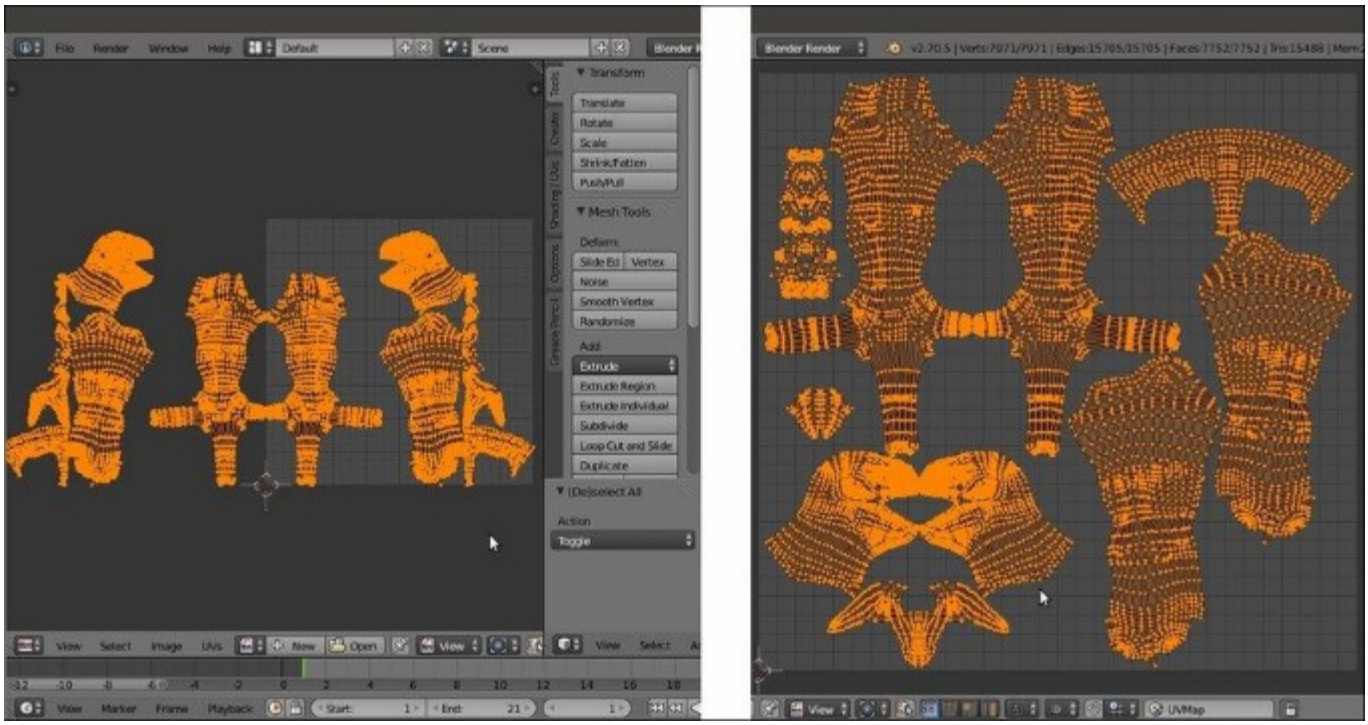
A new loop cut

5. Press *Ctrl* + *Tab* to switch to vertex selection and press *Z* to go into the **Wireframe** viewport shading mode, and then box-select the vertices on the left-hand side of the mesh (which is the side created by the **Mirror** modifier).
6. Go to the **UV/Image Editor** window; if not already selected, press *A* to select all the **UV islands** of the UV layer, and then press *Ctrl* + *M* | *X* | *Enter* to mirror these selected islands.
7. Press *G* to move them (temporarily) outside the default **U0/V0** tile space, as shown in the following screenshot:



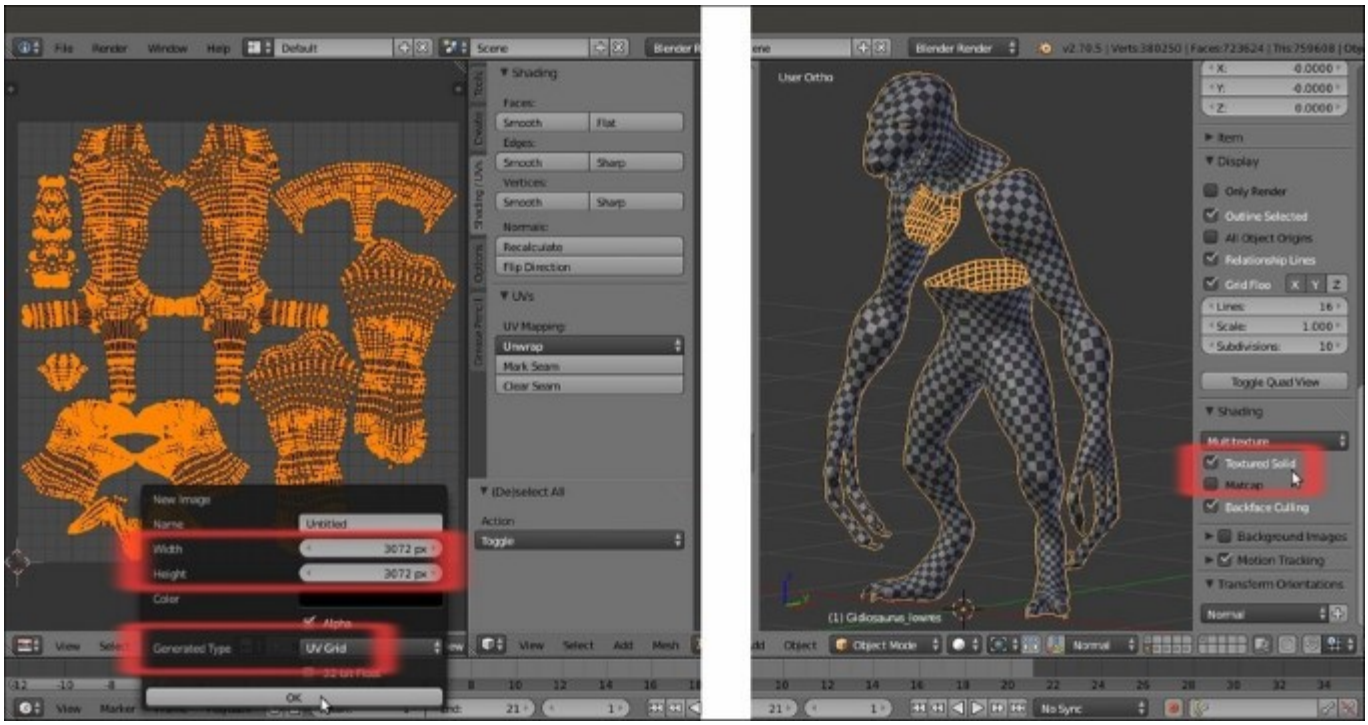
The selected half body vertices and the corresponding UV islands outside the U0/V0 tile space

8. Go to the 3D view and press *A* twice to select all the vertices; go to the **UV/Image Editor** window and press *Ctrl* + *A* to average the size of all the islands reciprocally.
9. Select all the islands and press *Ctrl* + *P* to automatically pack all of them inside the UV tile.
10. If you are not satisfied with the result of the **Pack islands** tool, adjust the position (*G* key), rotation (*R* key), and scale (*S* key) of the islands; group together the similar ones (for example, the **teeth**, **talons**, **arms**, and so on), but try to place them to fill the image tile space as much as possible. To select one island, just put the mouse over it and press *L*, and *Shift* + *L* to multiselect. Use the *X* and *Y* keys to constrain the movements of the islands on the corresponding axis:



Adjusting the UV islands' position

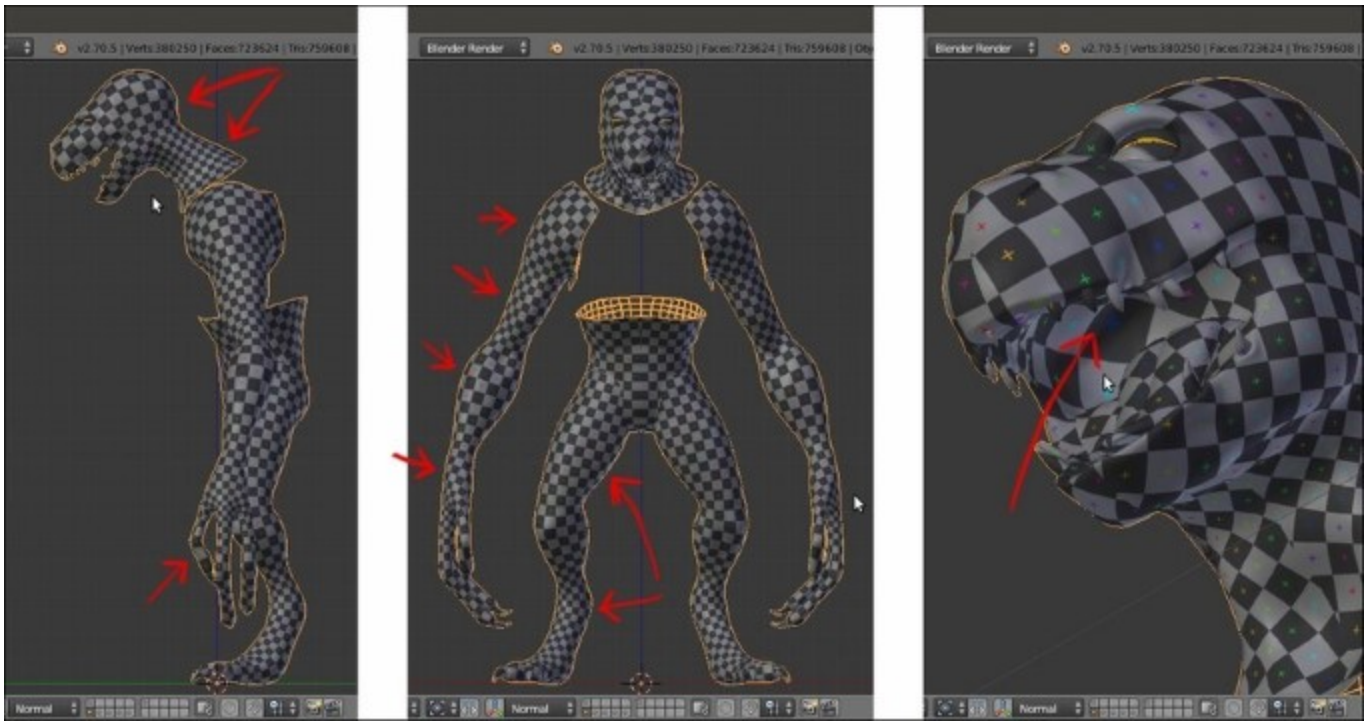
11. When you are done, ensure that all the vertices of all the islands in the **UV/Image Editor** window are selected, and click on the **New** button on the toolbar of the **UV/Image Editor** window; in the **New Image** pop-up panel, set **Width** and **Height** to **3072** pixels, and **Generated Type** should be set to **UV Grid**. Then, click on the **OK** button to confirm.
12. Go to the 3D window and press *Z* to go in the **Solid** viewport shading mode. Then, go to the **Properties** 3D view sidepanel and under the **Shading** subpanel, check the **Textured Solid** item.
13. Go out of the **Edit Mode** and save the file:



Assigning a grid image to the unwrapped UV islands

This should be enough; even if the halves of the mesh are disconnected, Blender can perfectly solve the mesh painting without visible seams.

However, if we look at the character shown in the **Textured Solid** mode in the 3D view, it's clear that the unwrap of some part of the mesh could be better; for example, you can see a difference in the size of the mapped grid in the **head/neck** area, inside the **mouth**, and on the **arms** and **legs** (look at the arrows in the following images):



Differences in the mapped grid image

Although this is not a very big issue, the unwrap can be refined further to avoid distortions as much as possible, as well as potential future problems when we'll paint the character textures; we are going to see this in the next recipe.

Editing the UV islands

We are now going to join the two UV islands' halves together, in order to improve the final look of the texturing; we are also going to modify, if possible, a little of the island proportions in order to obtain a more regular flow of the UV vertices, and fix the *distortions* we have seen in the last image of the previous recipe.

We are going to the use the **pin** tool, which is normally used in conjunction with the **Live Unwrap** tool.

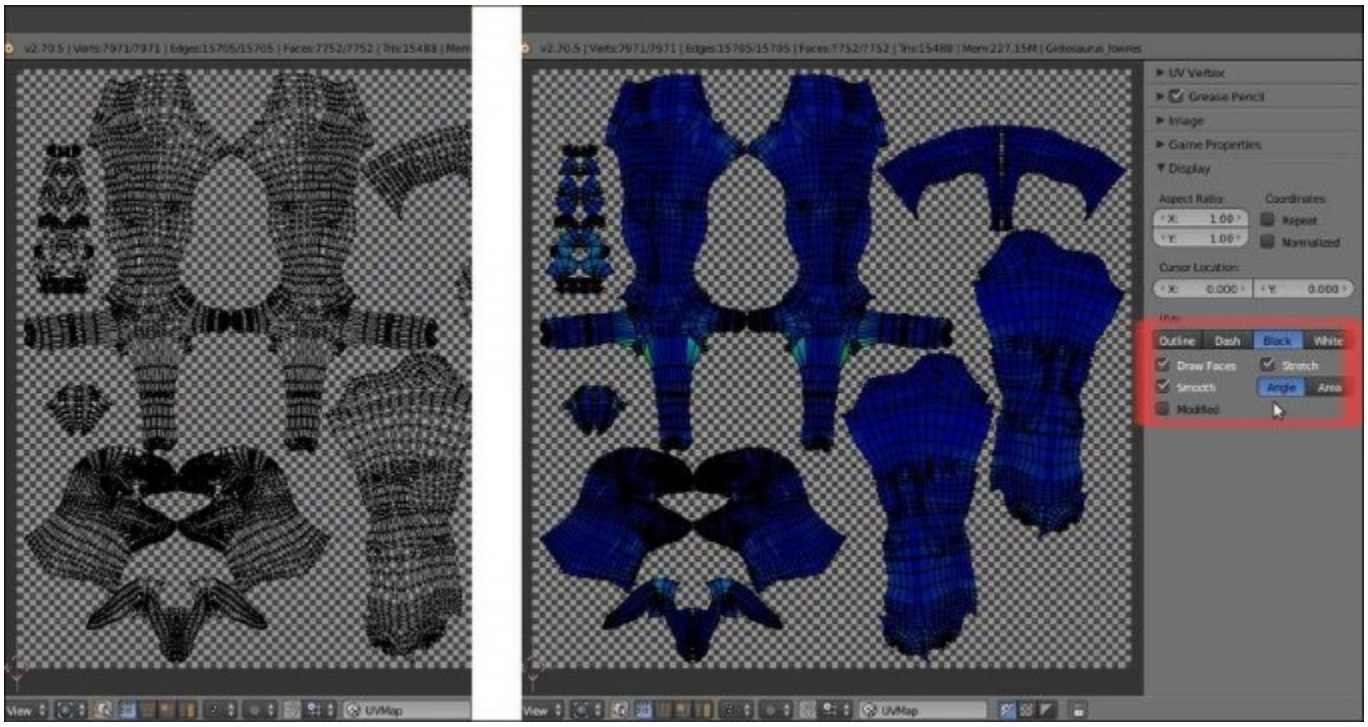
Getting ready

First, we'll try to recalculate the unwrap of some of the islands by modifying the seams of the mesh.

Before we start though, let's see if we can improve some of the visibility of the UV islands in the **UV/Image Editor**:

1. Put the mouse cursor in the **UV/Image Editor** window and press the *N* key.
2. In the **Properties** sidepanel that appears by pressing the *N* key on the right-hand side of the window, go to the **Display** subpanel and click on the **Black** or **White** button (depending on your preference) under the **UV** item. Check also the **Smooth** item box.
3. Also, check the **Stretch** item, which even though it was made for a different purpose, can increase the visibility of the islands a lot.
4. Press *N* again to get rid of the **Properties** sidepanel.

All these options enabled should make the islands more easily readable in the **UV/Image Editor** window:

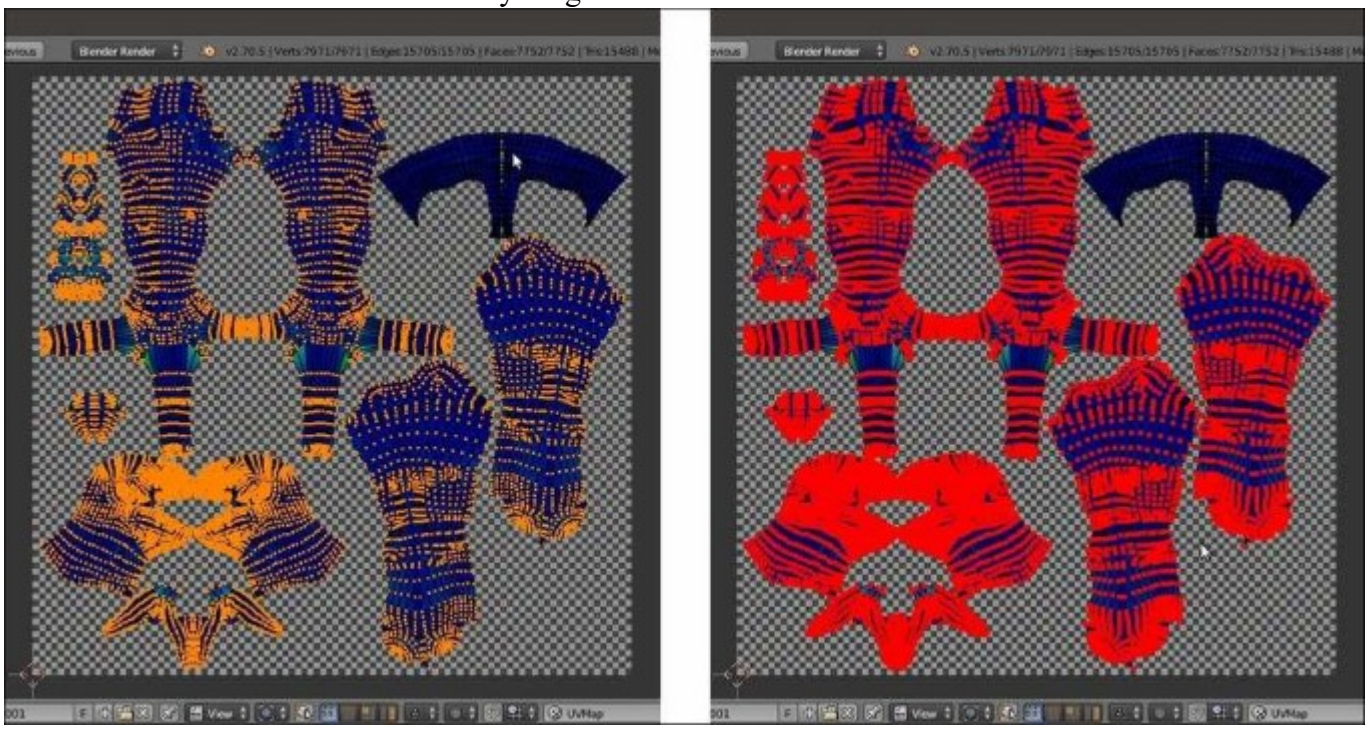


The UV islands made more easily readable by the enabled items

How to do it…

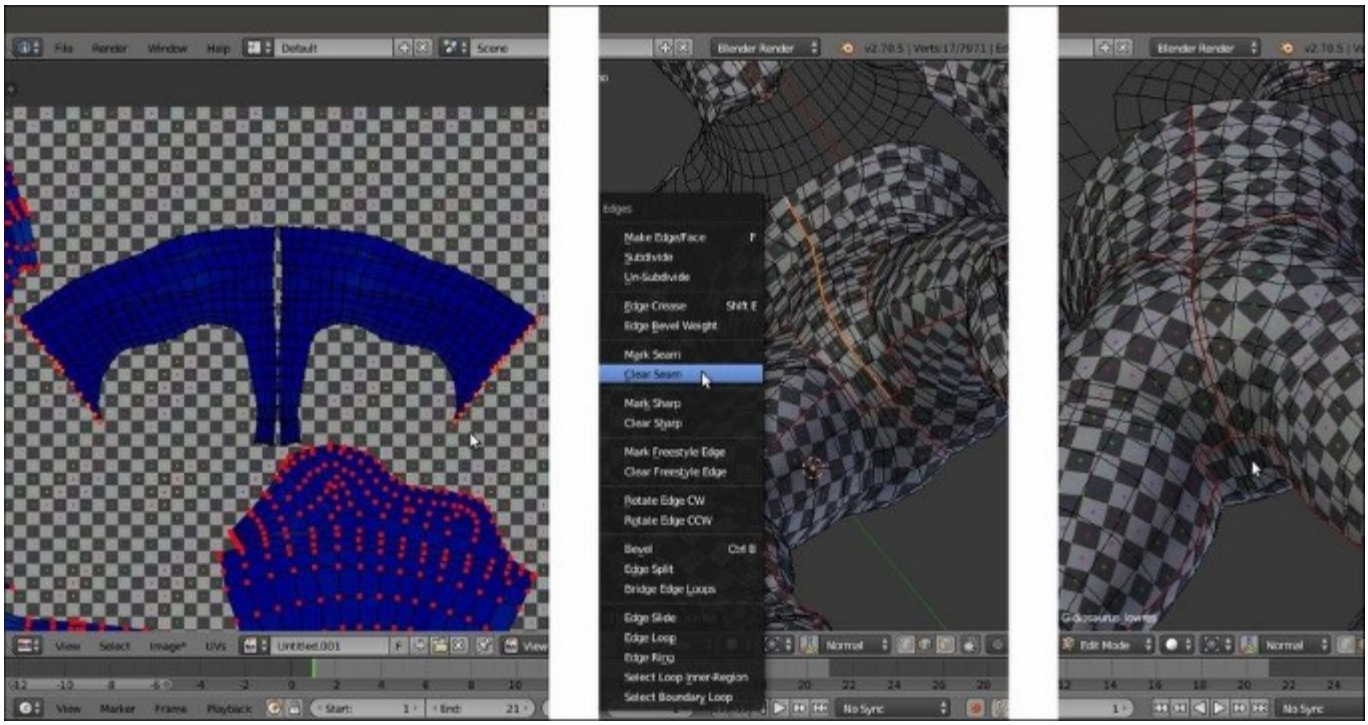
Now we can start with the editing; initially, we are going to freeze the islands that we don't want to modify because their unwrap is either satisfactory, or we will deal with it later. So, perform the following steps:

1. Press *A* to select all the islands, then by putting the mouse pointer on the two **pelvis** island halves and pressing *Shift* + *L*, multi-deselect them; press the *P* key to **pin** the remaining selected UV islands and then *A* to deselect everything:



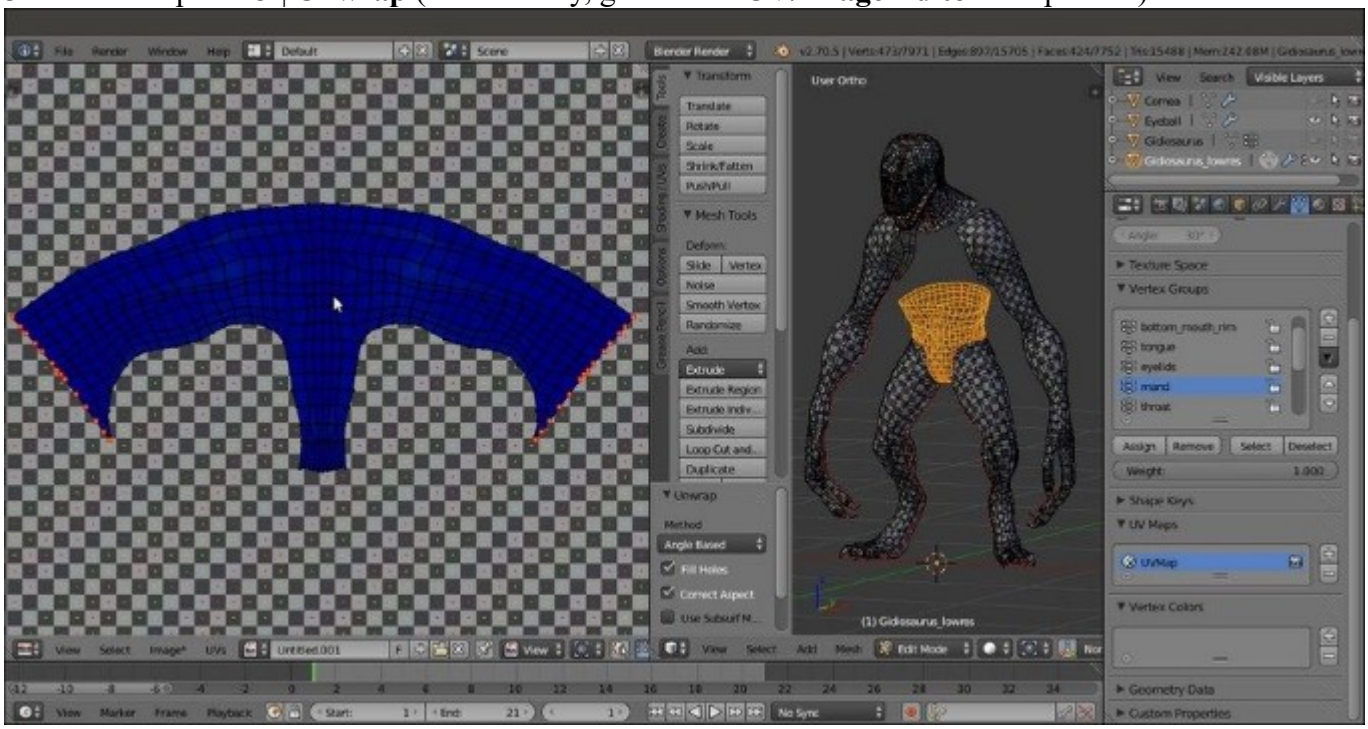
To the right-hand side, the pinned UV islands

2. Zoom in on the islands of the **pelvis**, select both the left and right outer edge-loops, as shown in the following left image, and press *P* to **pin** them.
3. Go to the 3D view and clear only the front part of the median seam on the **pelvis**. To do this, start to clear the seam from the front edges, go down and stop where it crosses the horizontal seam that passes the bottom part of the **groin** and **legs**, and leave the back part of the vertical median seam still marked:



Pinning the extreme vertices in the UV/Image Editor, and editing the seam on the mesh

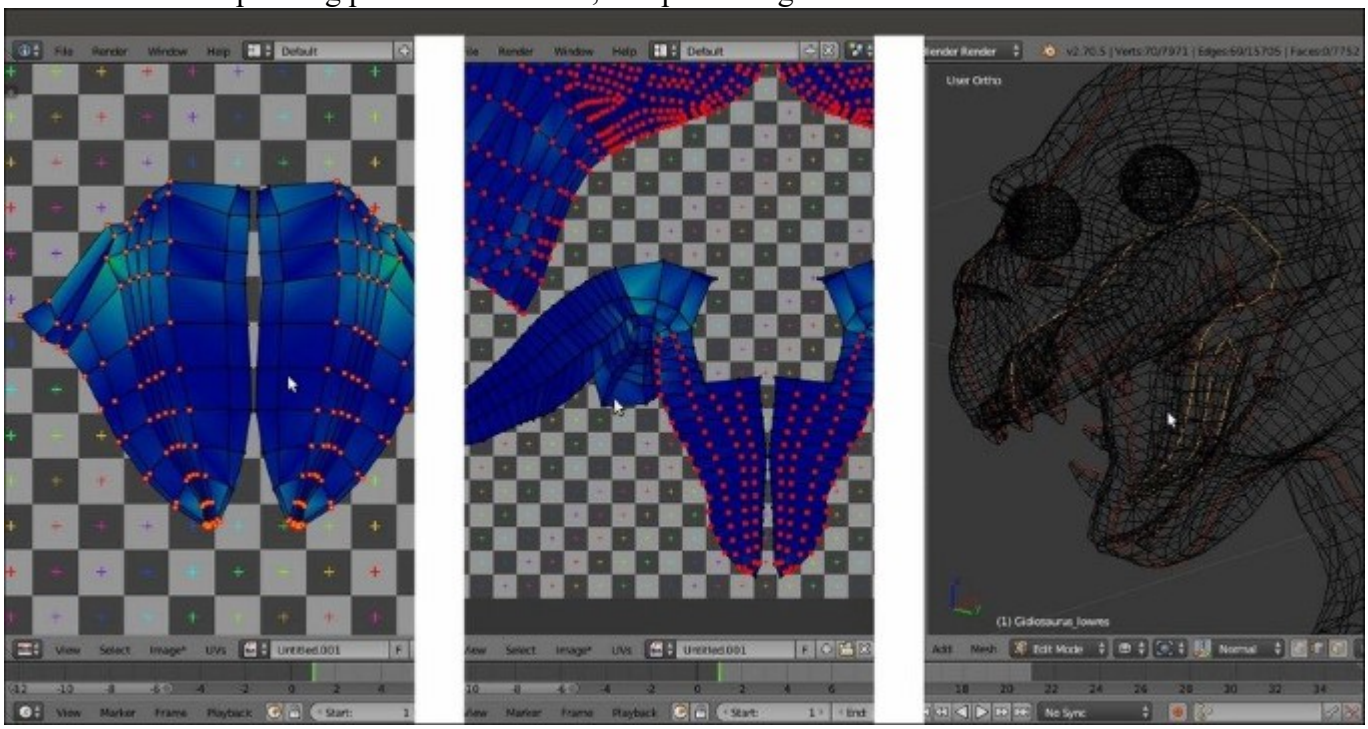
4. Go into **Face** selection mode and select all the faces of the **pelvis**; put the mouse pointer in the 3D view and press *U* | **Unwrap** (alternatively, go into the **UV/Image Editor** and press *E*):



Unwrapping again with the pinning and a different seam

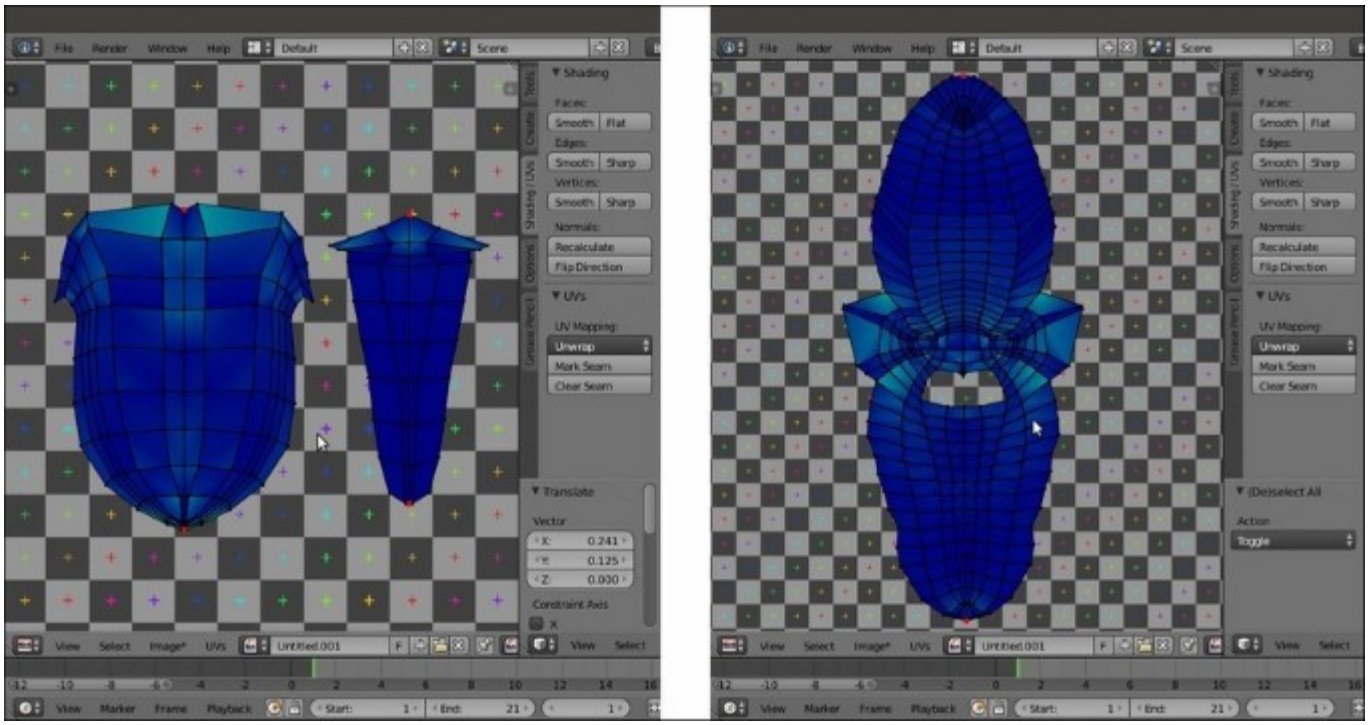
The island will keep the previous position because of the pinned edges, and is now unwrapped as one single piece (with the obvious exception of the seam on the back).

5. We won't modify the **pelvis** island any further, so select all its vertices and press *P* to pin all of them and then deselect them.
6. Press *A* in the 3D view to select all the faces of the mesh and make all the islands visible in the **UV/Image Editor**. Note that they are all pinned at the moment, so just select the vertices you want to **unpin** (*Alt* + *P*) in the islands of the **tongue** and **inner mouth**. Then, clear the median seam in the corresponding pieces on the mesh, and press *E* again:



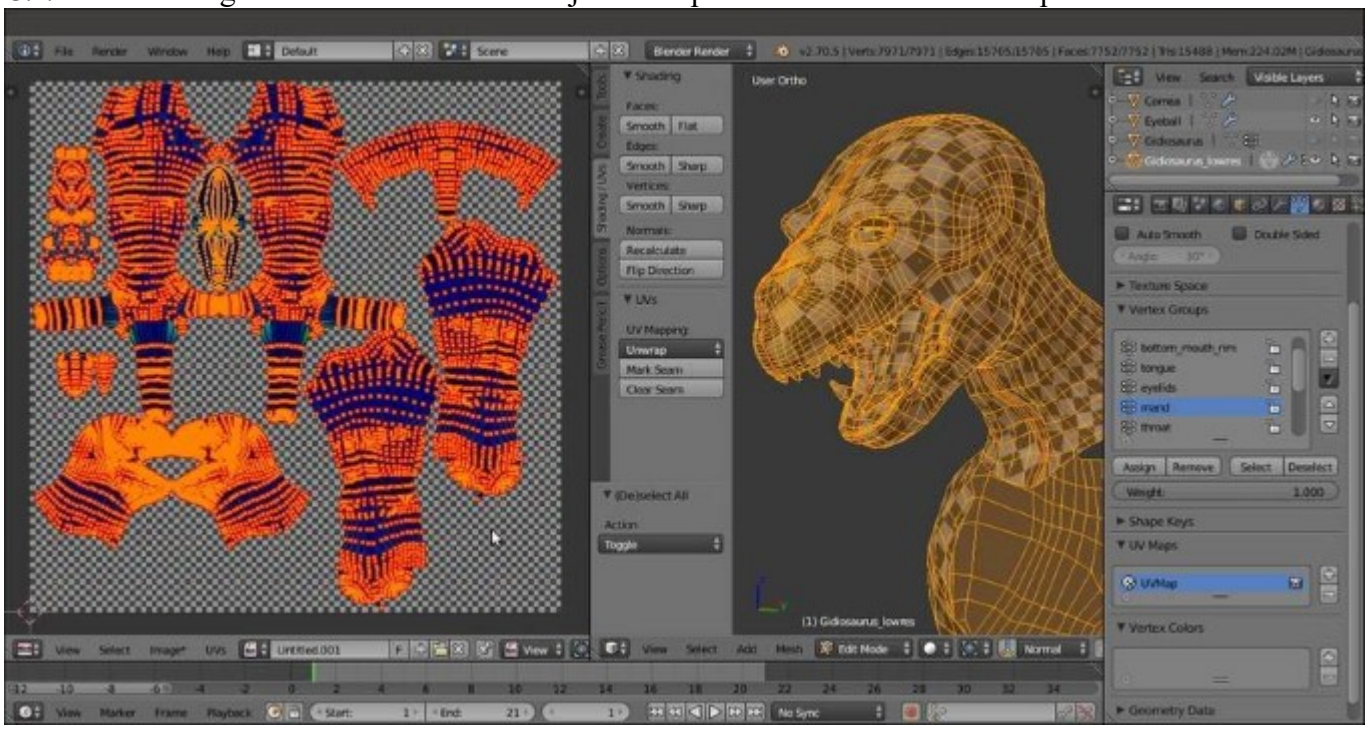
Re-unwrapping the tongue and inner mouth areas

7. Select the UV vertices of the resulting islands and unpin them all; next, pin just one vertex at the top of the islands and one at the bottom, and unwrap again. This will result in a more organically distributed unwrap of the parts:



Re-unwrapping again with a different pinning

8. Select all the faces of the mesh, and then all the islands in the **UV/Image Editor** window. Press *Ctrl + A* to average their relative size and adjust their position in the default tile space:



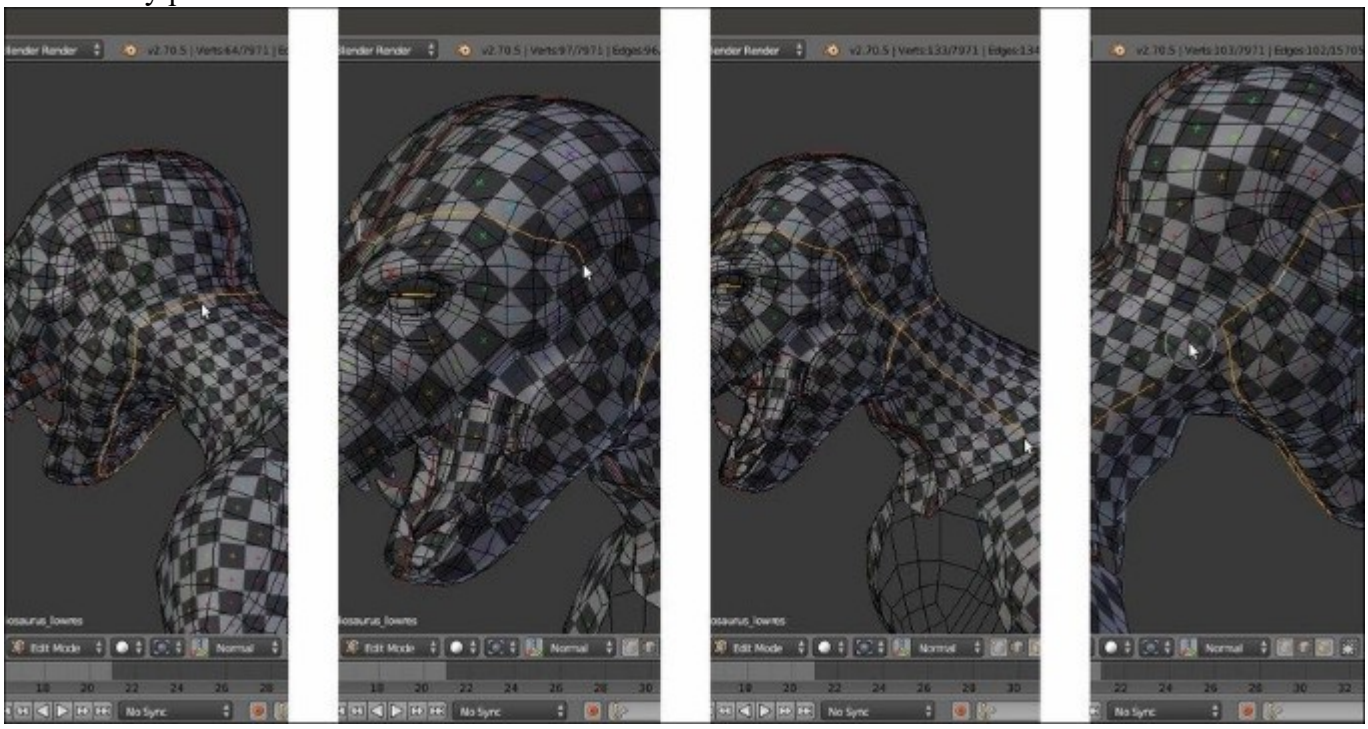
The rearranged UV islands

Now, let's work on the **head** piece that, as in every character, should be the most important and well-finished piece.

At the moment, the **face** is made using two separate islands; although this won't be visible in the final textured rendering of our character, it's always better, if possible, to join them in order to have a single piece, especially in the front mesh faces. Due to the elongated **snout** of the character, if we were to unwrap the **head** as a single piece simply without the median seam, we wouldn't get a nice evenly mapped result, so we must divide the whole **head** into more pieces.

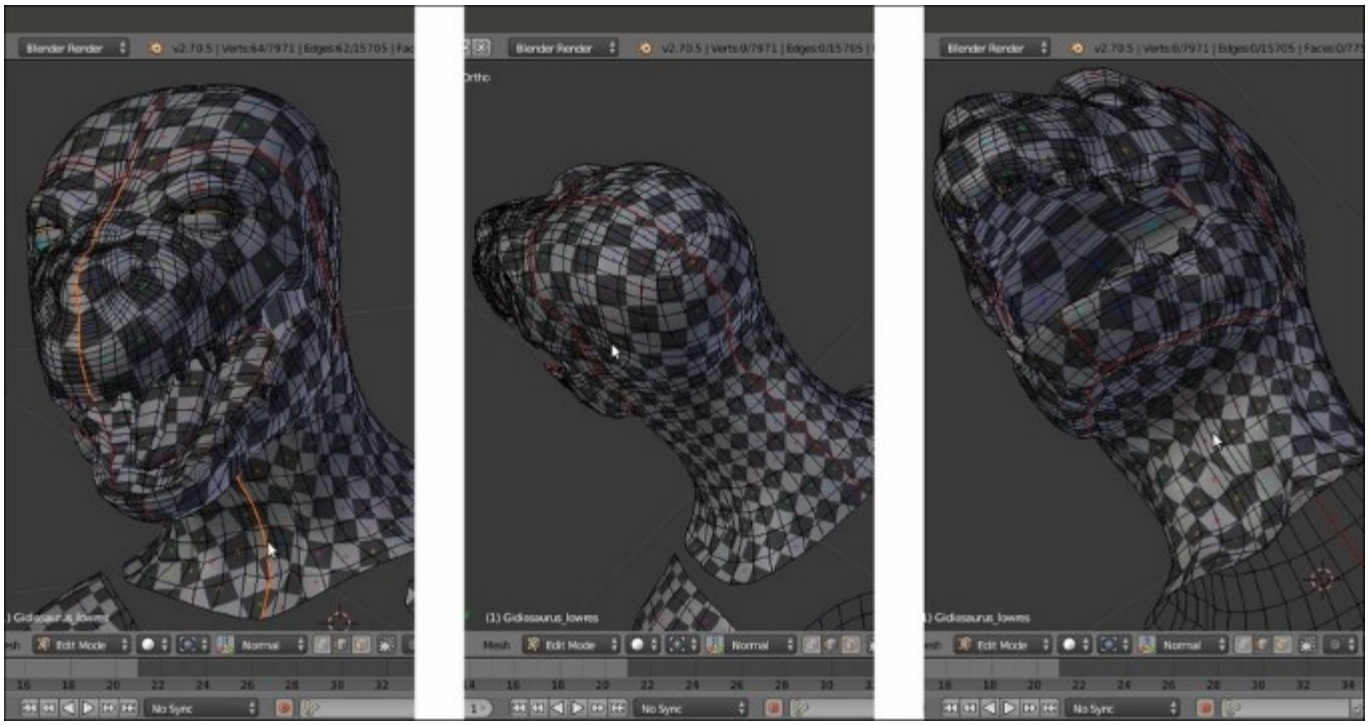
Actually, we can take advantage of the fact that the **Gidiosaurus** is wearing a **helmet** and that most of the **head** will be covered by it; this allows us to easily split the **face** from the rest of the mesh, hiding the seams under the **helmet**.

9. Go into **Edge** selection mode and mark the seams, dividing the **face** from the **cranium** and **neck** as shown in the following screenshots. Select the crossing edge-loops, and then clear the unnecessary parts:



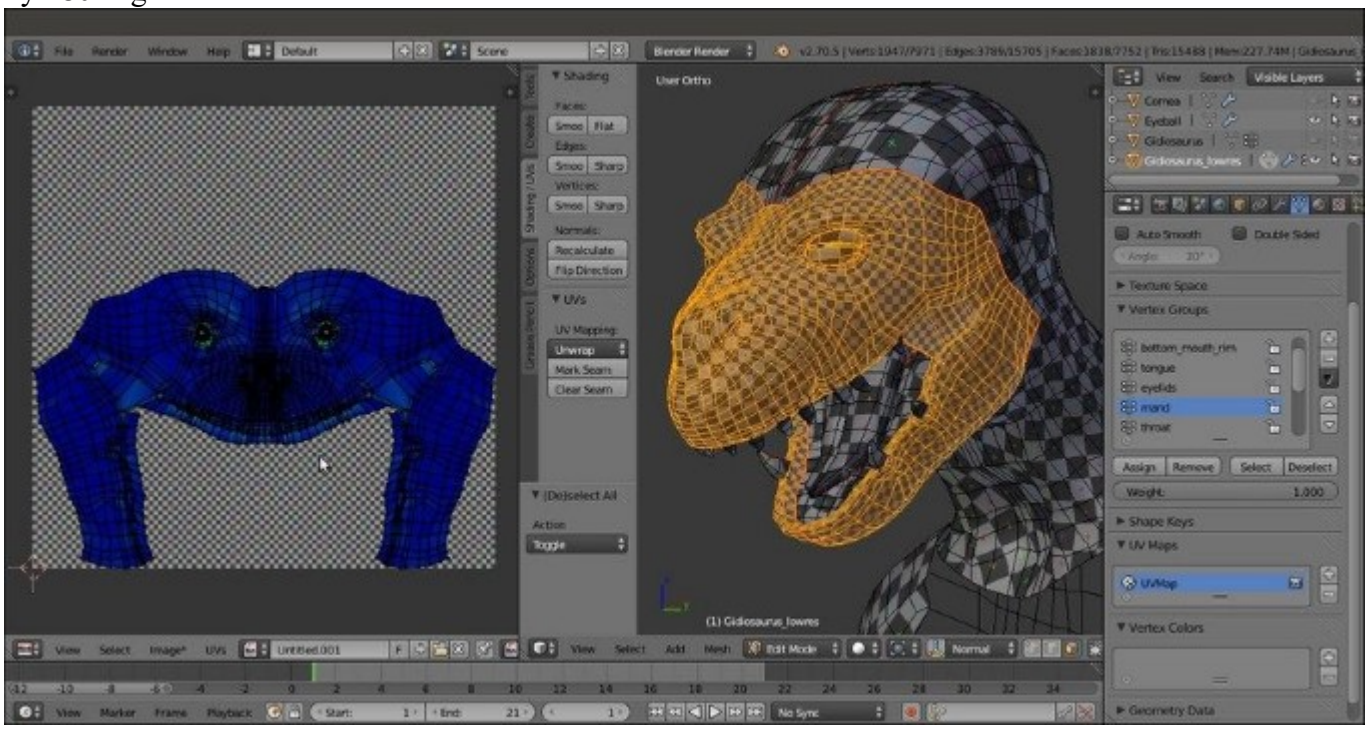
New seams for the character's head part 1

10. Also clear the median seam in the upper **face** part, and under the seam on the bottom **jaw**, leaving it only on the front **mandible** and on the back of the **cranium** and **neck**:



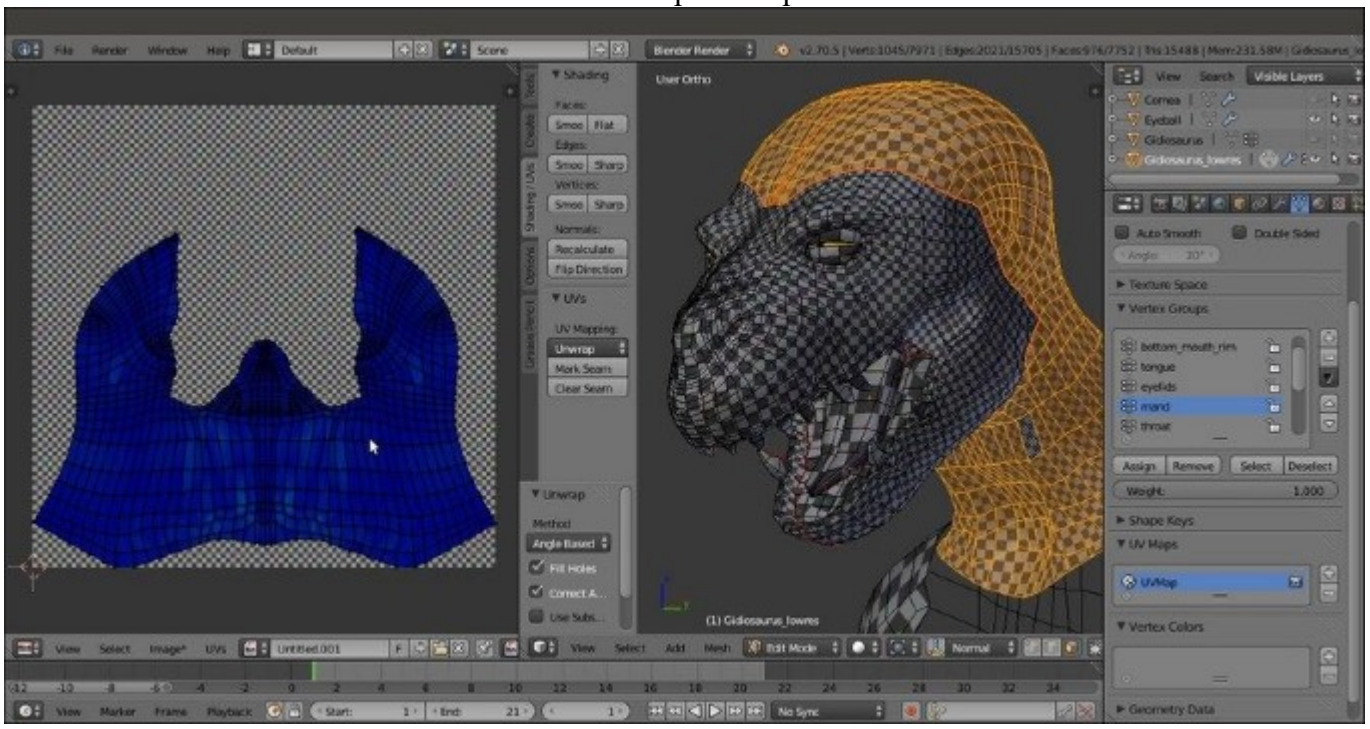
New seams for the character's head part 2

11. Go in the **Face** selection mode and select only the **face** section of the mesh, and then press *E* to unwrap. The new unwrap comes upside down, so select all the UV vertices and rotate the island by **180** degrees:



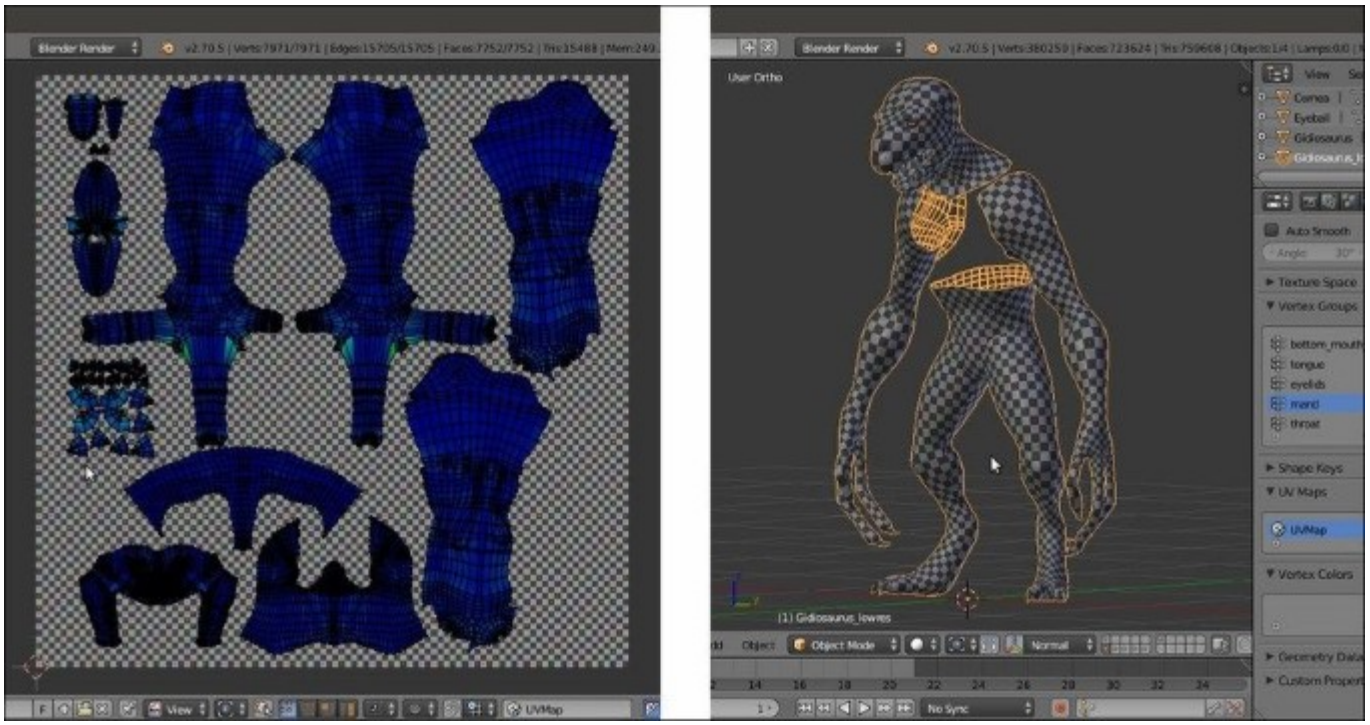
The character's face unwrapped

12. Select the **cranium/neck** section on the mesh and repeat the process:



The rest of the head mesh unwrapped as a whole piece

13. Now, select all the faces of the mesh and all the islands in the **UV/Image Editor**, and press *Ctrl + A* to average their reciprocal size.
14. Once again, adjust the position of the islands inside the UV tile (*Ctrl + P* to automatically pack them inside the available space, and then tweak their position, rotation, and scale):



The character's UV islands packed inside the default U0/V0 tile space

How it works…

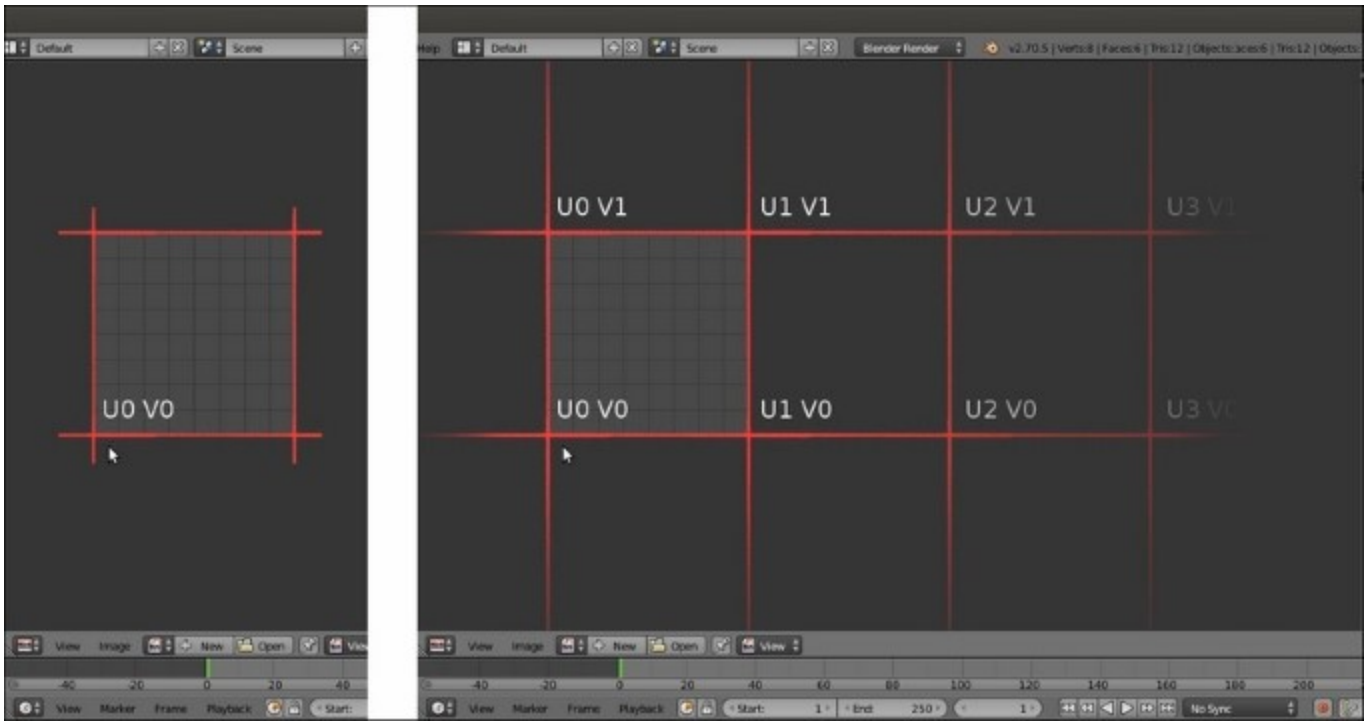
Starting from the UV unwrap in the previous recipe, we improved some of the islands by joining together the halves representing common mesh parts. When doing this, we tried to retain the already good parts of the unwrap by **pinning** the UV vertices that we didn't want to modify; this way, the new unwrap process was forced to calculate the position of the unpinned vertices using the constraints of the pinned ones (**pelvis**, **tongue**, and **inner mouth**). In other cases, we totally cleared the old seams on the model and marked new ones, in order to have a completely new unwrap of the mesh part (the **head**), we also used the character furniture (such as the **armor**) to hide the seams (which in any case, won't be visible at all).

There's more…

At this point, looking at the **UV/Image Editor** window containing the islands, it's evident that if we want to keep several parts in proportion to each other, some of the islands are a little too small to give a good amount of detail when texturing; for example, the Gidiosaurus's face.

A technique for a good unwrap that is the current standard in the industry is **UDIM UV Mapping**, which means **U-Dimension**; basically, after the usual unwrap, the islands are scaled bigger and placed outside the default **U0/V0** tile space.

Look at the following screenshots, showing the Blender **UV/Image Editor** window:

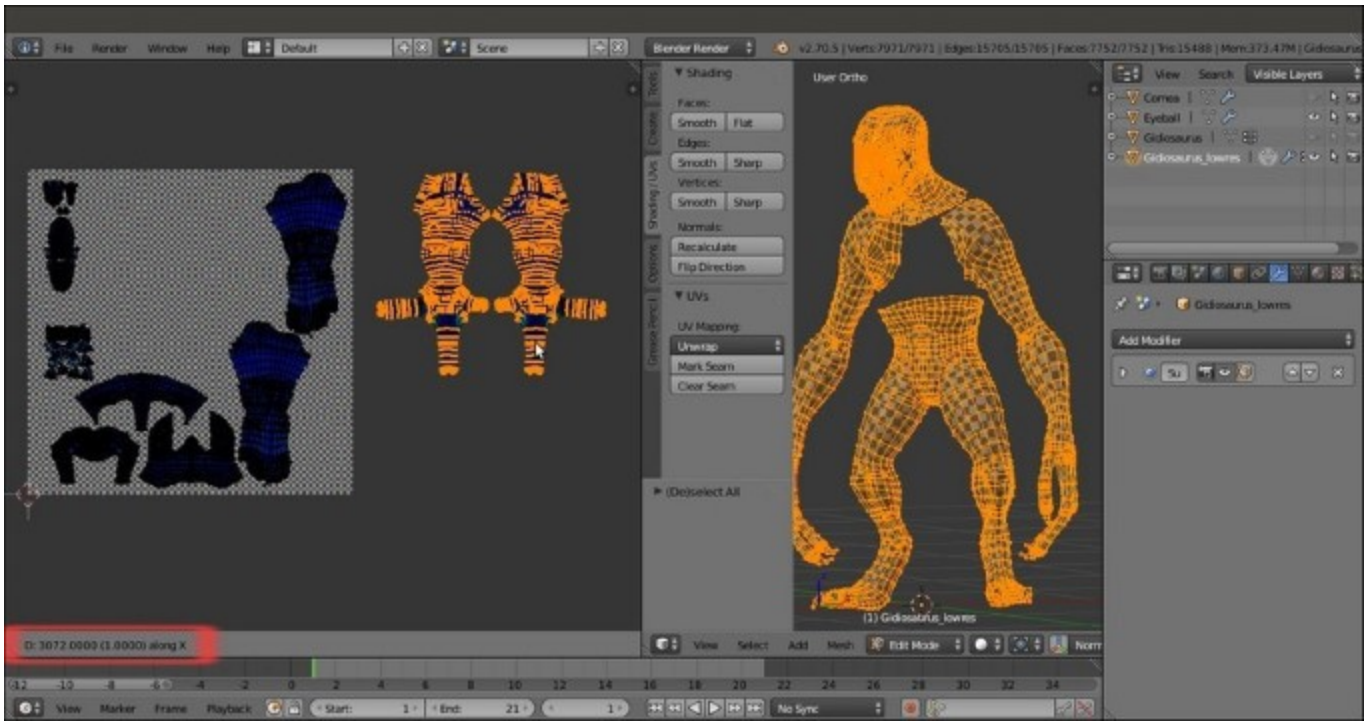


The default U0/V0 tile space and the possible consecutive other tile spaces

On the left-hand side, you can see, highlighted with red lines, the single UV tile that at present is the standard for Blender, which is identified by the **UV** coordinates **0** and **0**: that is, **U** (horizontal) = **0** and **V** (vertical) = **0**.

Although not visible in the **UV/Image Editor** window, all the other possible consecutive tiles can be identified by the corresponding UV coordinates, as shown on the right-hand side of the preceding screenshot (again, highlighted with red lines). So, adjacent to the tile **U0/V0**, we can have the row with the tiles **U1/V0**, **U2/V0**, and so on, but we can also go upwards: **U0/V1**, **U1/V1**, **U2/V1**, and so on.

To help you identify the tiles, Blender will show you the amount of pixels and also the number of tiles you are moving the islands in the toolbar of the **UV/Image Editor** window. In the following screenshot, the **arm** islands have been moved horizontally (on the negative *x* axis) by **-3072.000** pixels; this is correct because that's exactly the X size of the grid image we loaded in the previous recipes. In fact, in the toolbar of the **UV/Image Editor** window, while moving the islands we can read **D: -3072.000** (pixels) and (inside brackets) **1.0000** (tile) **along X**; effectively, **3072** pixels = **1** tile.



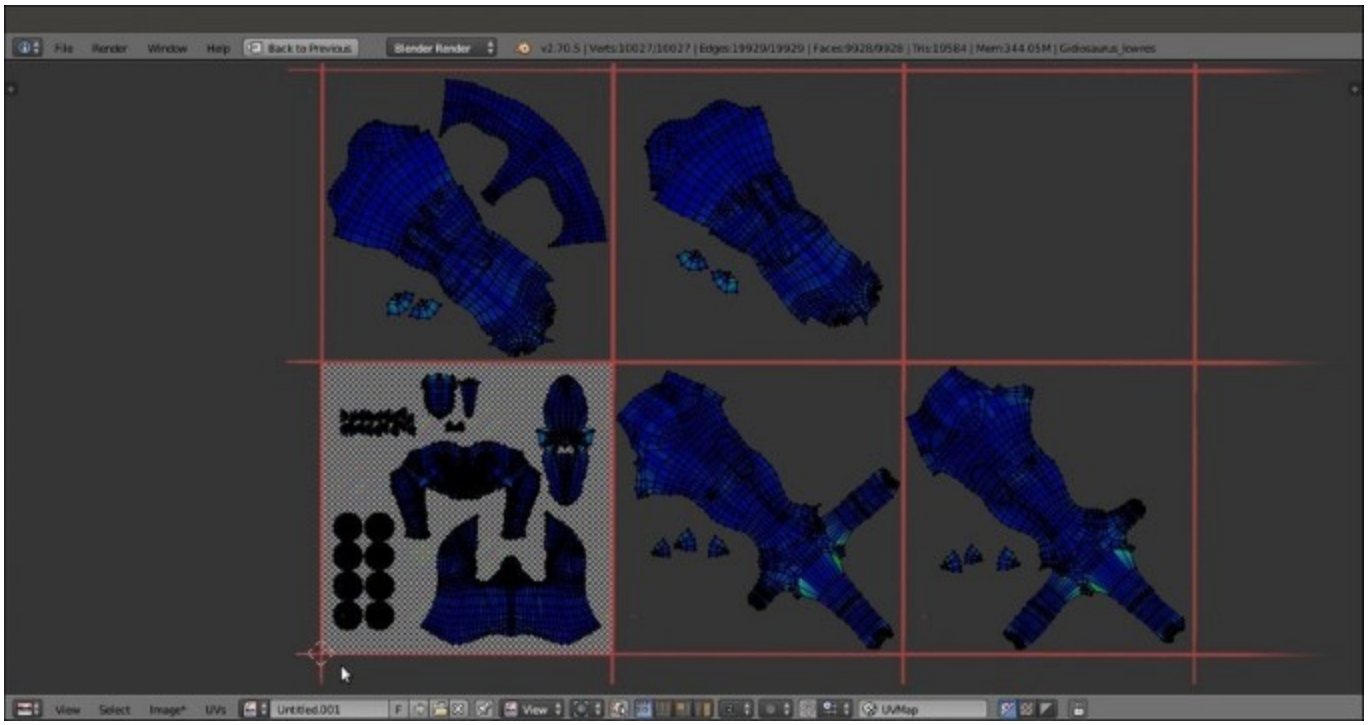
Moving the arm islands to the U1/V0 tile space

When moving UV islands from tile to tile, remember to check that the **Constrain to Image Bounds** item in the **UVs** menu on the toolbar of the **UV/Image Editor** window is disabled; also, enabling the **Normalized** item inside the **Display** subpanel under the *N* key *Properties* sidepanel of the same editor window will display the UV coordinates from **0.0** to **1.0**, rather than in pixels. More, pressing the *Ctrl* key while moving the islands will constrain the movement to intervals, making it easy to translate them to exactly 1 tile space.

Because at the moment Blender doesn't support the **UDIM UV Mapping** standard, simply moving an island outside the default **U0/V0** tile, for example to **U1/V0**, will *repeat the image* you loaded in the **U0/V0** tile and on the faces associated with the moved islands. To solve this, it's necessary, after moving the islands, to *assign a different material, if necessary with its own different image textures*, to each group of vertices/faces associated with each tile space. So, if you shared your islands over **4** tiles, you need to assign **4** different materials to your object, and each material must load the proper image texture.

The goal of this process is obviously to obtain bigger islands mapped with bigger texture images, by selecting all the islands, scaling them bigger *together* using the largest ones as a guide, and then tweaking their position and distribution.

One last thing: it is also better to unwrap the **corneas** and **eyes** (which are separate objects from the **Gidiosaurus** body mesh) and add their islands to the tiles where you put the **face**, **mouth**, **teeth**, and so on (use the **Draw Other Objects** tool in the **View** menu of the **UV/Image Editor** window to also show the UV islands of the other *nonjoined* unwrapped objects):



UV islands unwrapped, following the UDIM UV Mapping standard

In our case, we assigned the **Gidiosaurus** body islands to **5** different tiles, **U0/V0**, **U1/V0**, **U2/V0**, **U0/V1**, and **U1/V1**, so we'll have to assign **5** different materials. However, we will cover this in a later recipe.

Note that for exposition purposes only, in the preceding screenshot, you can see the cornea and eye islands together with the **Gidiosaurus** body islands because I temporarily joined the objects; however, it's usually better to maintain the eyes and corneas as separate objects from the main body.

Using the Smart UV Project tool

Now, we are going to use a much easier and faster method to do the unwrapping of the **Armor**: the **Smart UV Project** tool.

Getting ready

The first thing to do is to prepare the **armor** pieces for the unwrap process, so perform the following steps:

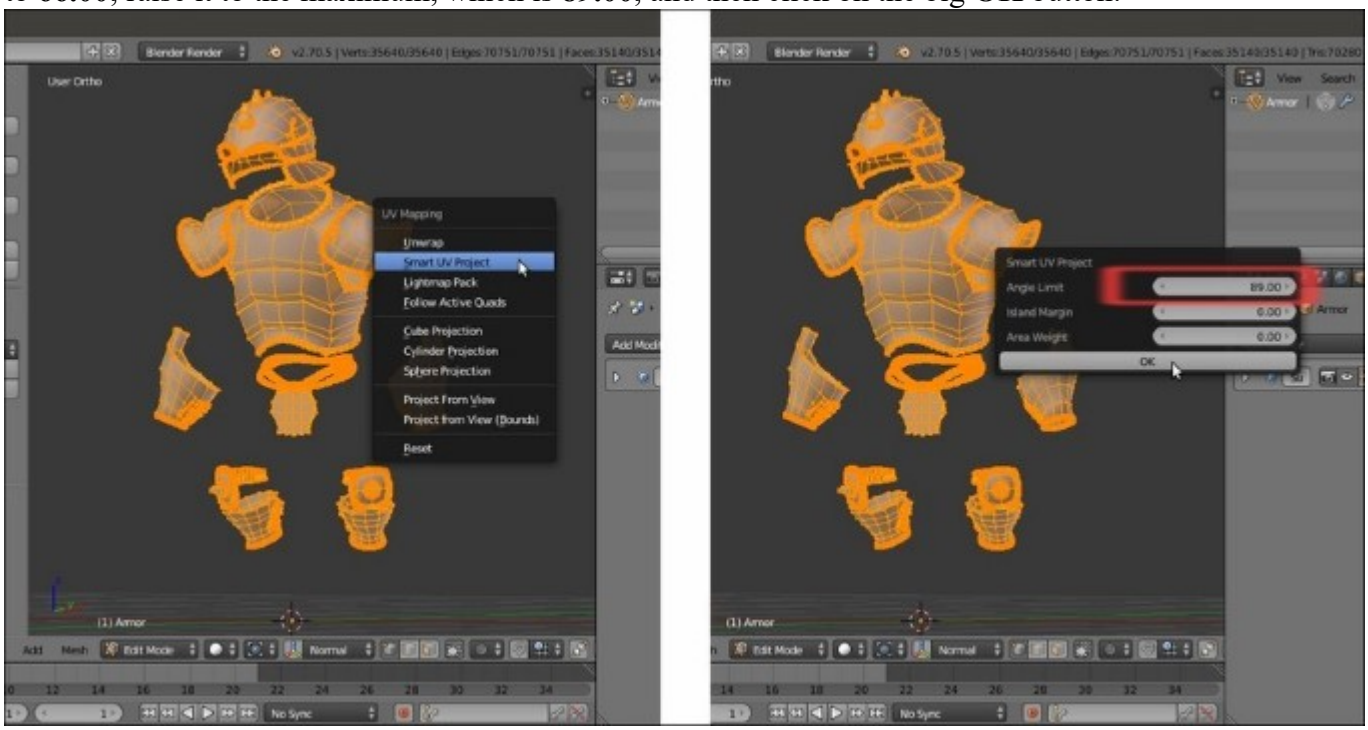
1. Starting from the last `Gidiosaurus_unwrap.blend` file you saved, click on the **13th** scene layer to reveal the **armor** and at the same time, hide the **Gidiosaurus_lowres** object.
2. Go to the **Outliner** and select the first item, the **Breastplate**; then, use *Shift* to multiselect all the other visible objects.
3. Press *Ctrl* + *J* to join them into a single object, and then in the **Outliner**, rename the result as `Armor`.
4. Go to the **Object Modifiers** window and expand the **Mirror** modifier subpanel; be sure that the **Clipping** item is activated and click on the **Apply** button:

The Armor as a single object and the Mirror modifier

How to do it…

Here is the unwrap process:

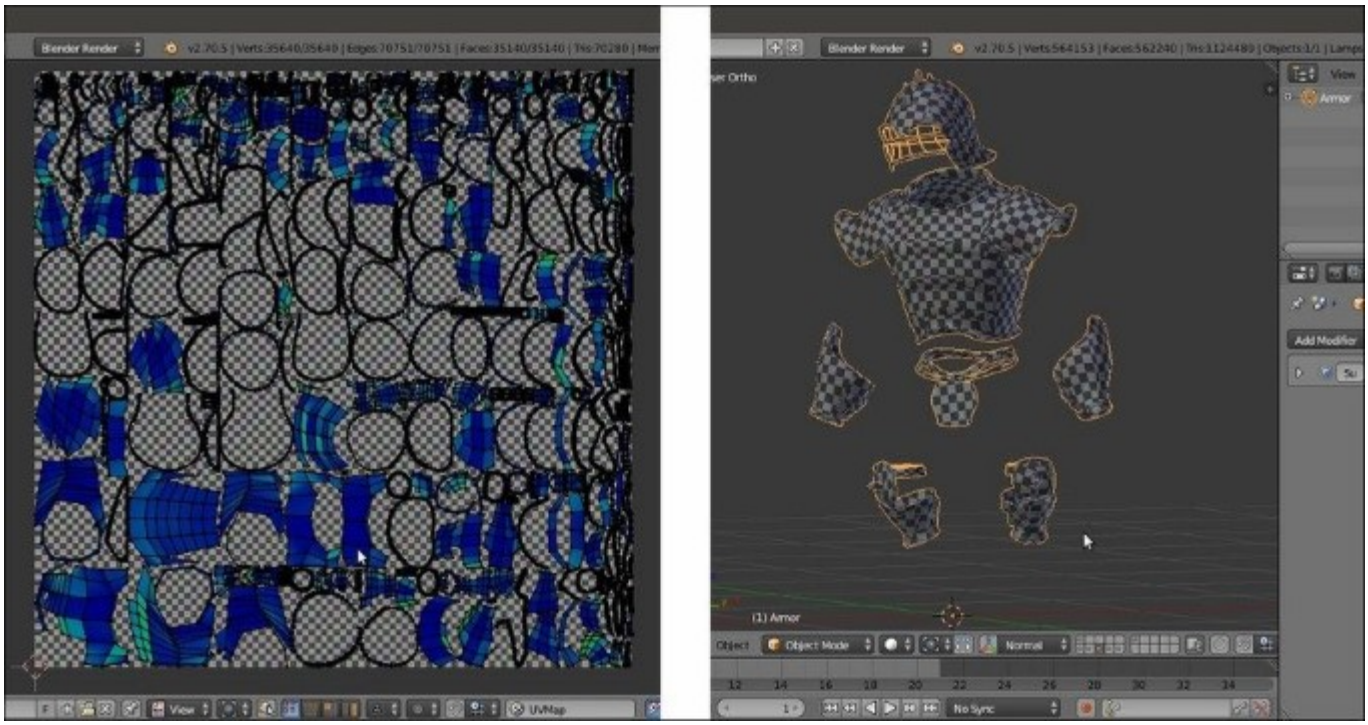
1. Press *Tab* to go into **Edit Mode** and press the *A* key to select all the vertices of the **Armor**.
2. With the mouse cursor in the 3D view, press the *U* key, and in the **UV Mapping** pop-up menu that just appeared, select the second item from the top, **Smart UV Project**.
3. A second pop-up appears with some options that you can leave as they are, besides **Angle Limit** (the maximum angle in the mesh used by the tool to separate the islands), which by default is set to **66.00**; raise it to the maximum, which is **89.00**, and then click on the big **OK** button:



The Smart UV Project tool

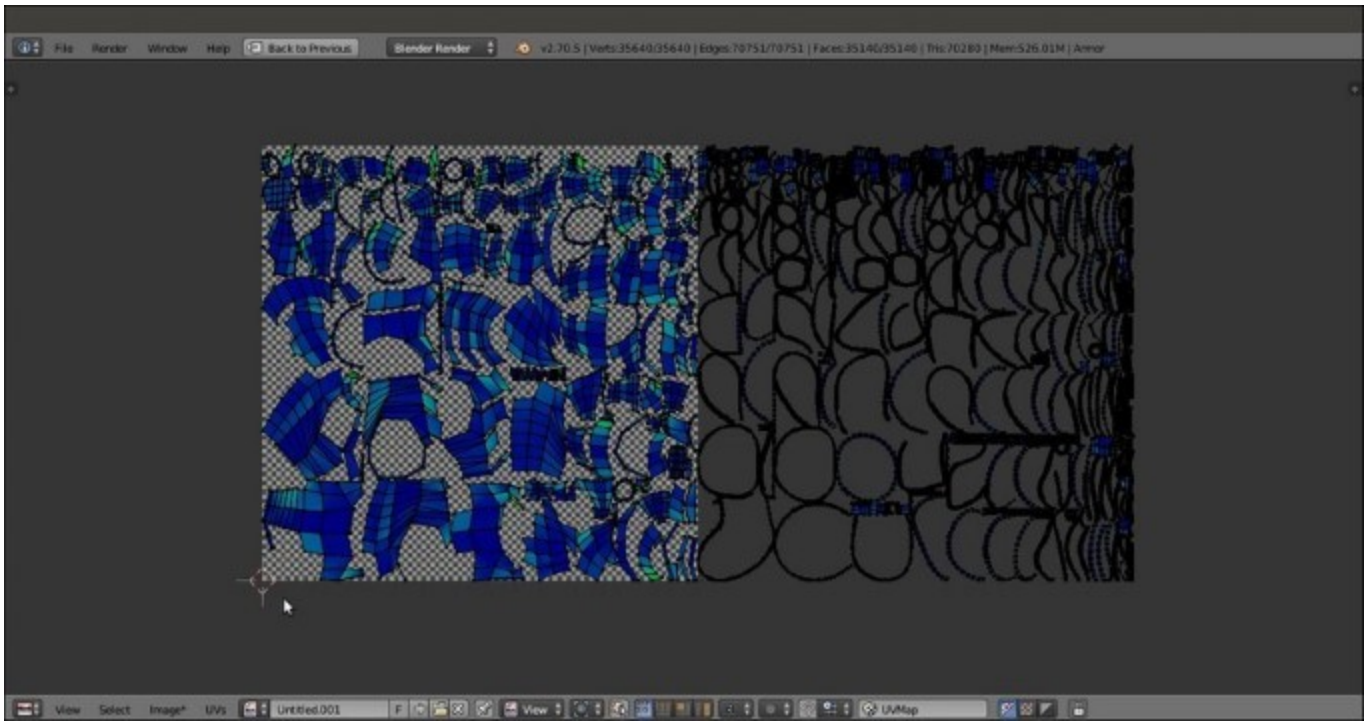
The mesh has been divided into several smaller unwrapped parts and is automatically packed inside the **U0/V0** UV tile.

4. Select all the islands in the **UV/Image Editor** window, click on the small double-arrow icon on the toolbar, close to the **New** and **Open** buttons, and select the **Untitled.001** image (the same grid image we used for the **Gidiosaurus** unwrap).
5. Press *Tab* to go out of **Edit Mode**:



The unwrapped Armor

Considering the amount of tiny islands that the tool created, it's better to separate the big **armor** parts (basically, the **plates**) from the smaller ones (**belts**, **borders**, and so on) and re-unwrap them with the **Smart UV Project** tool, as we did for the **Gidiosaurus** body in the previous recipe; then, place them into two adjacent tiles:



The Armor islands inside the U0/V0 and U1/V0 tiles

Modifying the mesh and the UV islands

At this point, when we look at the **Gidiosaurus** mesh, we realize that some detail in the model is still missing; for example, the **lower teeth**. In fact, we modeled the **mouth** closed and the **lower teeth**, enclosed in the **upper mouth rim**, weren't visible.

It's now time to add them; in fact, even though we have already done the unwrapping stage, it's still possible to modify the mesh further and also update the UV islands accordingly.

Getting ready

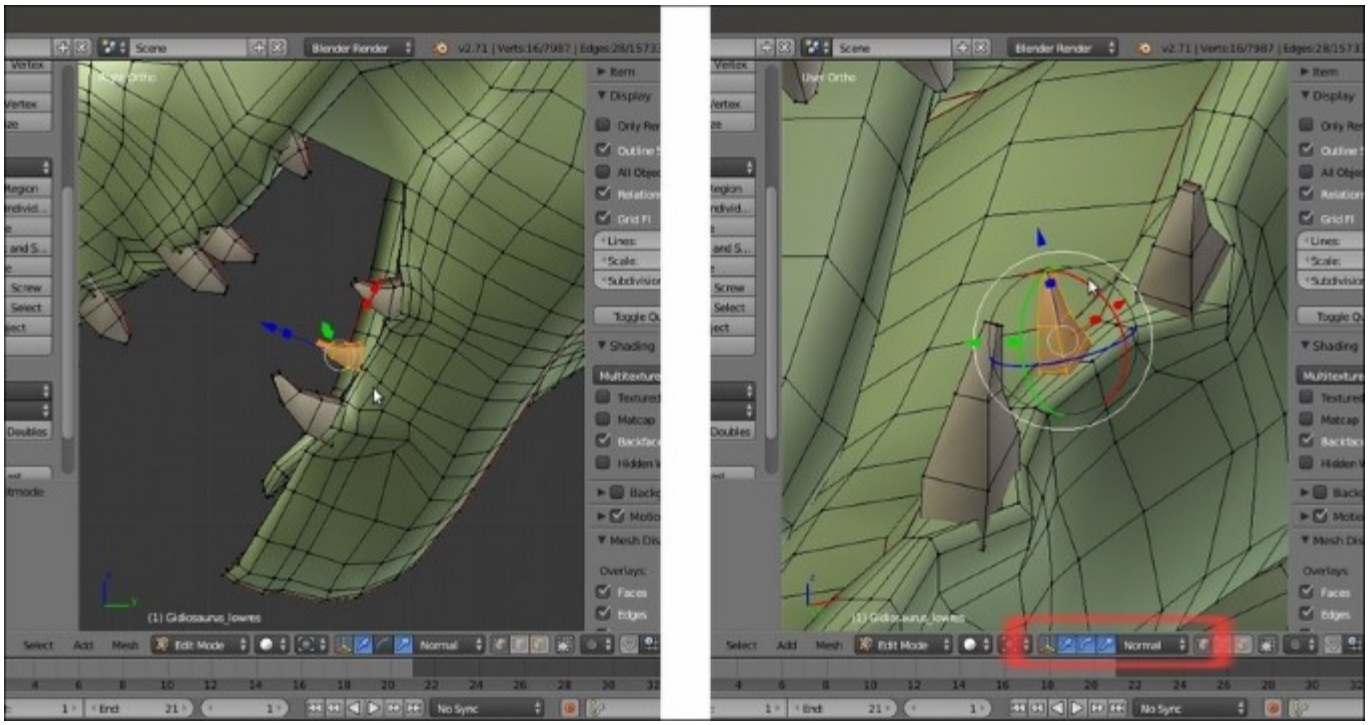
Start from the last `Gidiosaurus_unwrap.blend` file you saved:

1. Press *N* to open the **Properties** 3D view sidepanel, and disable the **Textured Solid** item under the **Shading** subpanel.
2. Click on the **11th** scene layer button to reveal the **Gidiosaurus_lowres** object and select it; go into the **Side** view, zoom in to the **head**, and enter **Edit Mode**. Press the *A* key to select all the vertices of the mesh.
3. Put the mouse pointer inside the **UV/Image Editor**, select all the UV vertices (again the *A* key), and pin them by pressing the *P* key; then deselect everything (the *A* key once more).

How to do it…

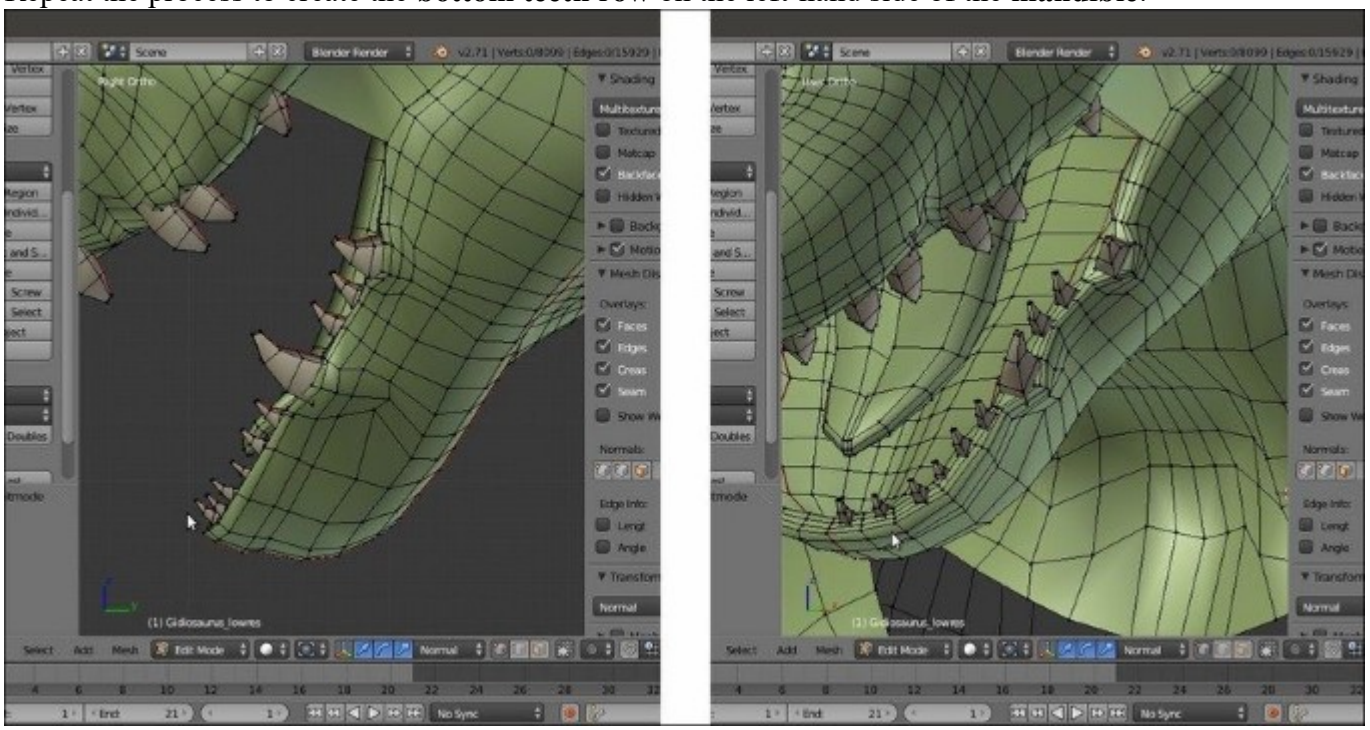
We can now start to add new **teeth**:

1. Select the vertices of one **lower tooth** and press *Shift* + *D* to duplicate it; using the **Transform Orientation** widget set to **Normal**, scale it smaller, rotate and modify it a bit, and then move it in a new position along the **mandible rim**:



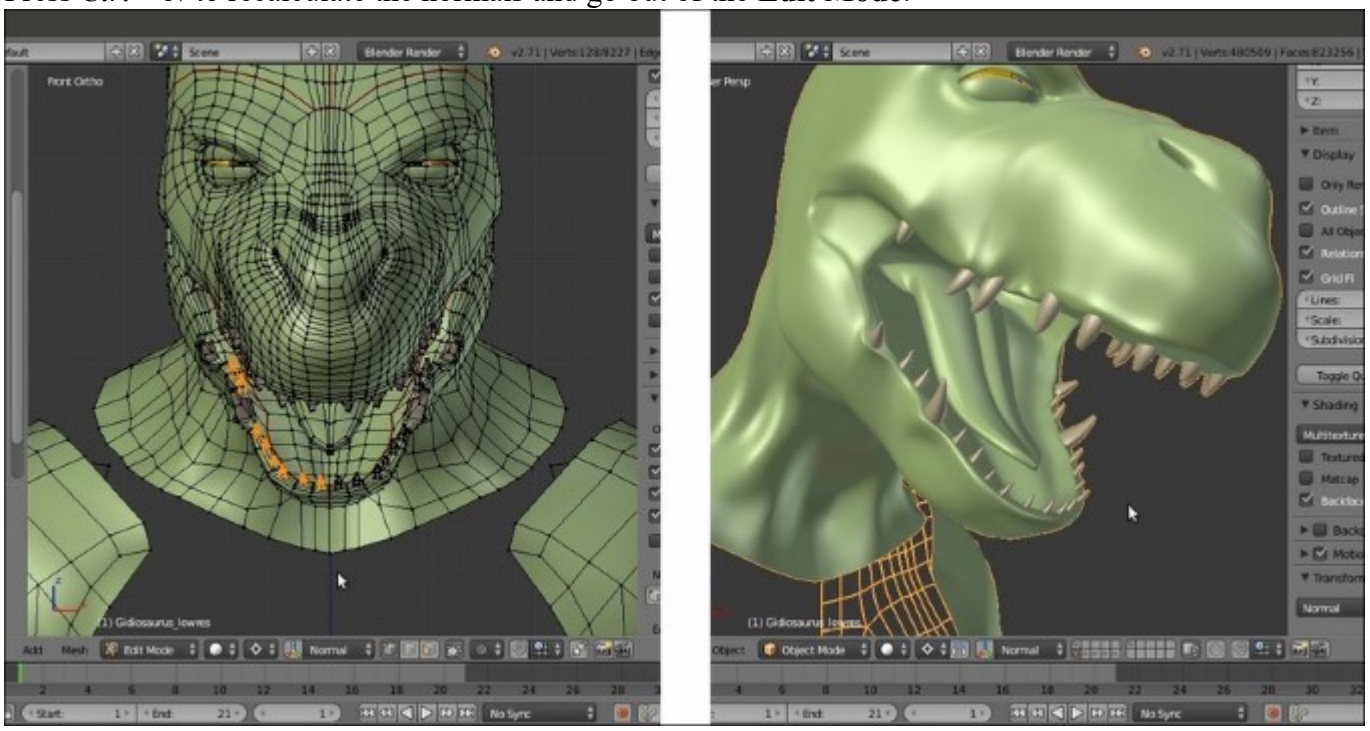
The new added tooth

2. Repeat the process to create the **bottom teeth row** on the left-hand side of the **mandible**:



Adding the missing teeth

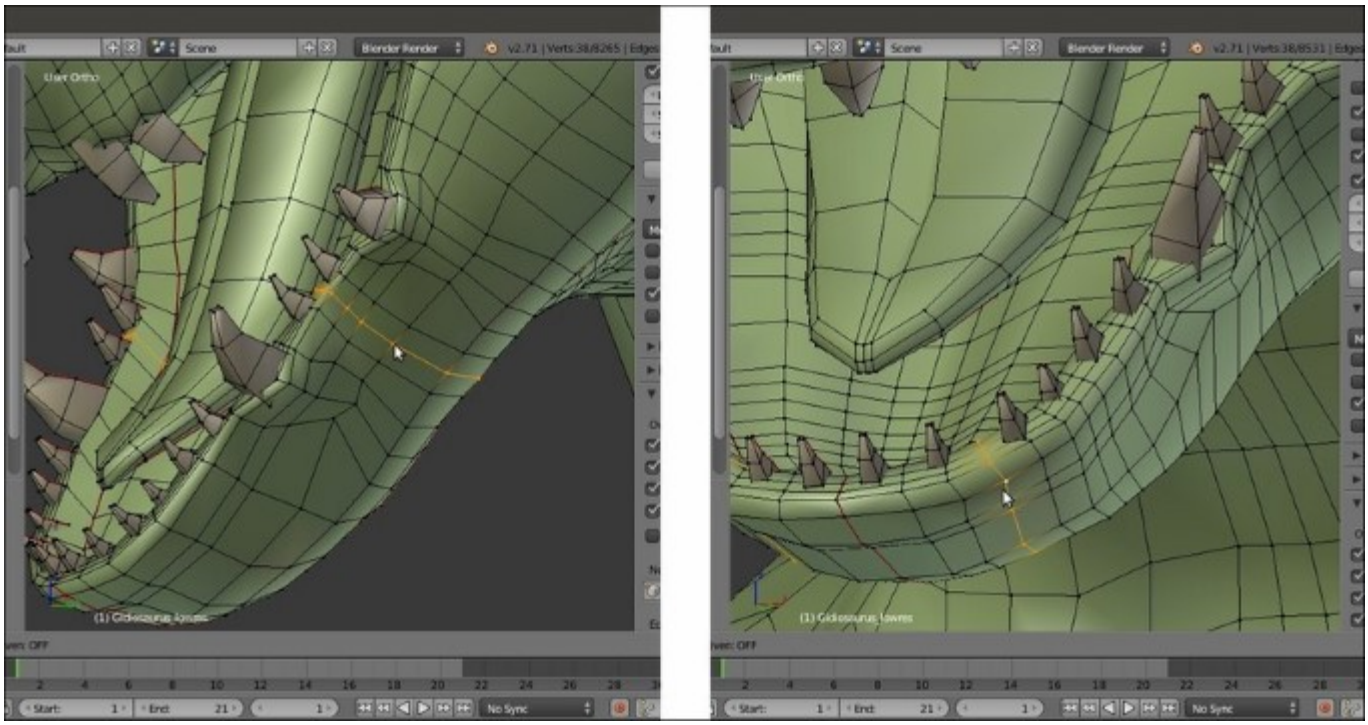
3. Go out of the **Edit Mode** and put the **3D Cursor** at the pivot point of the mesh (coincidentally, at the center of the scene), and then press the period key (.) to set the **Pivot Point** around the **3D Cursor**.
4. Press *1* on the numpad to go in the **Front** view, enter **Edit Mode** again, and select all the new **teeth**; press *Shift* + *D* to duplicate them, and then right-click; then, press *Ctrl* + *M* | *X* | *Enter* to mirror them on the *x* axis to the right-hand side of the **mandible**.
5. Press *Ctrl* + *N* to recalculate the normals and go out of the **Edit Mode**:



The new teeth mirrored on the x axis

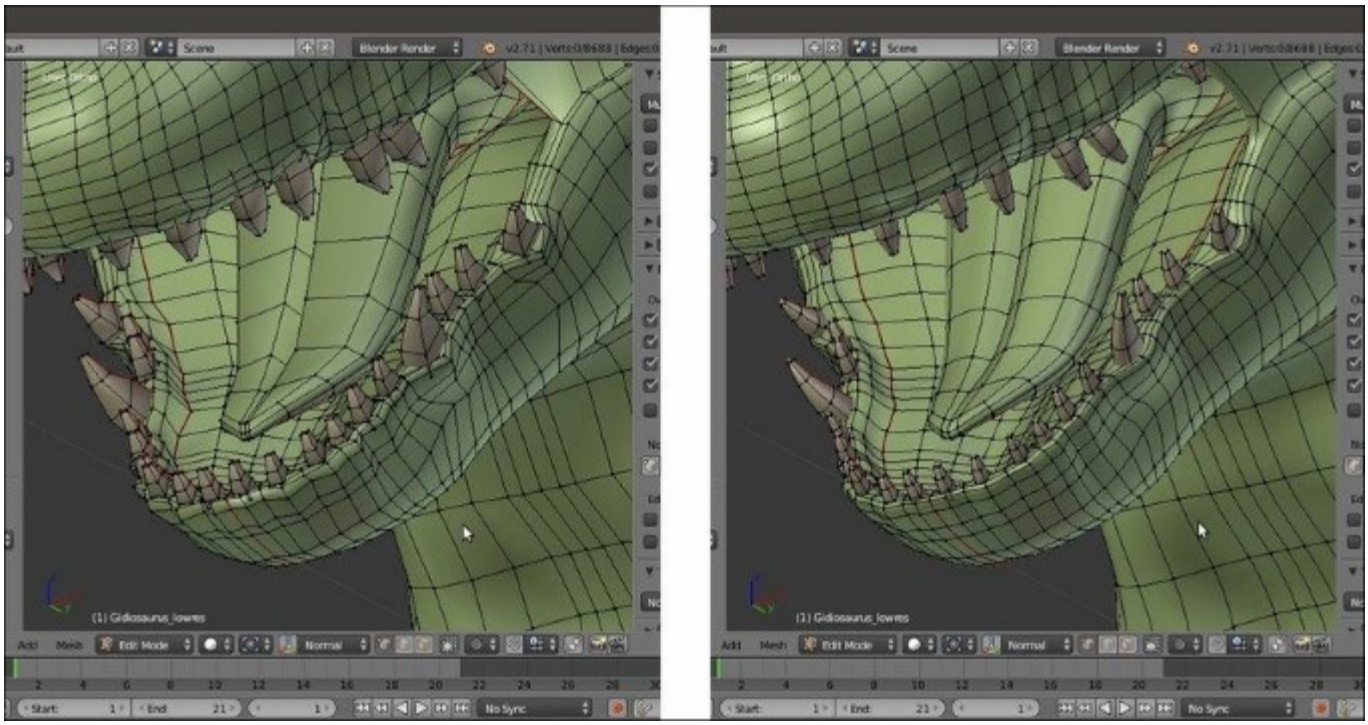
Now, we must adjust the **rim** of the **mandible** where we added the new **teeth**, in order to create the **alveoli**.

6. Go back into the **Edit Mode** and start to add vertical edge-loops on the **lower mouth rim**, in order to create more geometry for the **alveoli**:



Adding new edge-loops

7. Click on the **Options** tab under the **Tool Shelf** to the left-hand side of the 3D window and enable the **X Mirror** item under the **Mesh Options** subpanel.
8. Tweak the vertices to create the **alveoli** around the **new teeth**; enable the **Subdivision Surface** modifier visibility during **Edit Mode** in order to have better feedback:

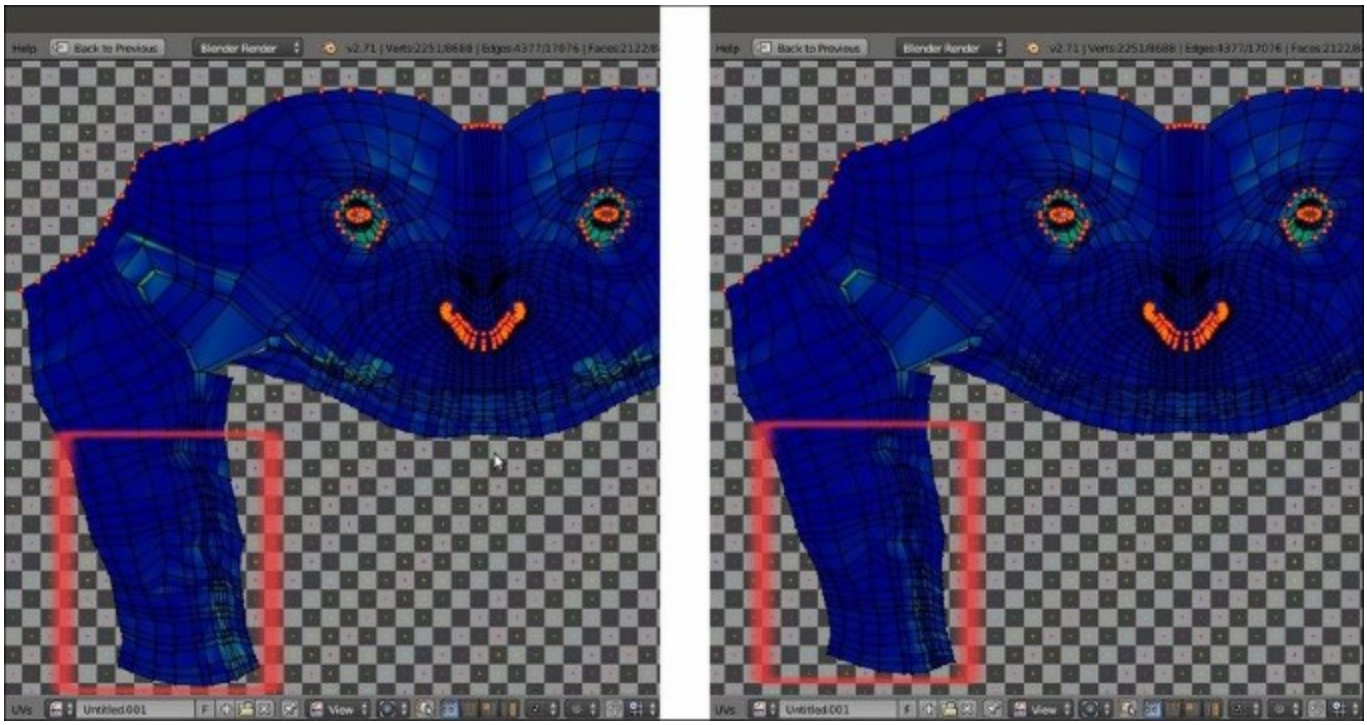


Modeling the alveoli

9. Press *Alt* + *S* to scale the **teeth** vertices on their normals, in order to thicken them, and add edge-loops where needed to make the transition from the **alveoli** to **inner mouth** as natural as possible.

   When you are done, it's time to update the unwrapped UV layer with the new modifications.
10. In **Edit Mode**, first select the vertices of the **new teeth**. Because we made them by duplicating one of the already unwrapped **fangs**, the **new teeth** will share the same UV island. In the **UV/Image Editor**, press *A* to select all their UV vertices and *Alt* + *P* to unpin them, and then press *E* for a new unwrap.
11. Scale the new **teeth** islands to **0.200**, and then select the original **teeth** on the mesh; adjust the size and position of the new islands based on the old ones and then pin them.
12. Now, switch to the **Face** selection mode and select the **Gidiosaurus** face; in the **UV/Image Editor** window, unpin all the vertices of its island (*Alt* + *P*), and then pin only the vertices of certain areas such as the **eyes**, **nose**, and upper outside edge-loop (*P* key).

With this method, the unwrap of all the new geometry gets recalculated together with the old one. Thanks to the pinned UV vertices, it will keep the previous size and position as much as possible. In the following image, you can see the face island before (left) and after (right):

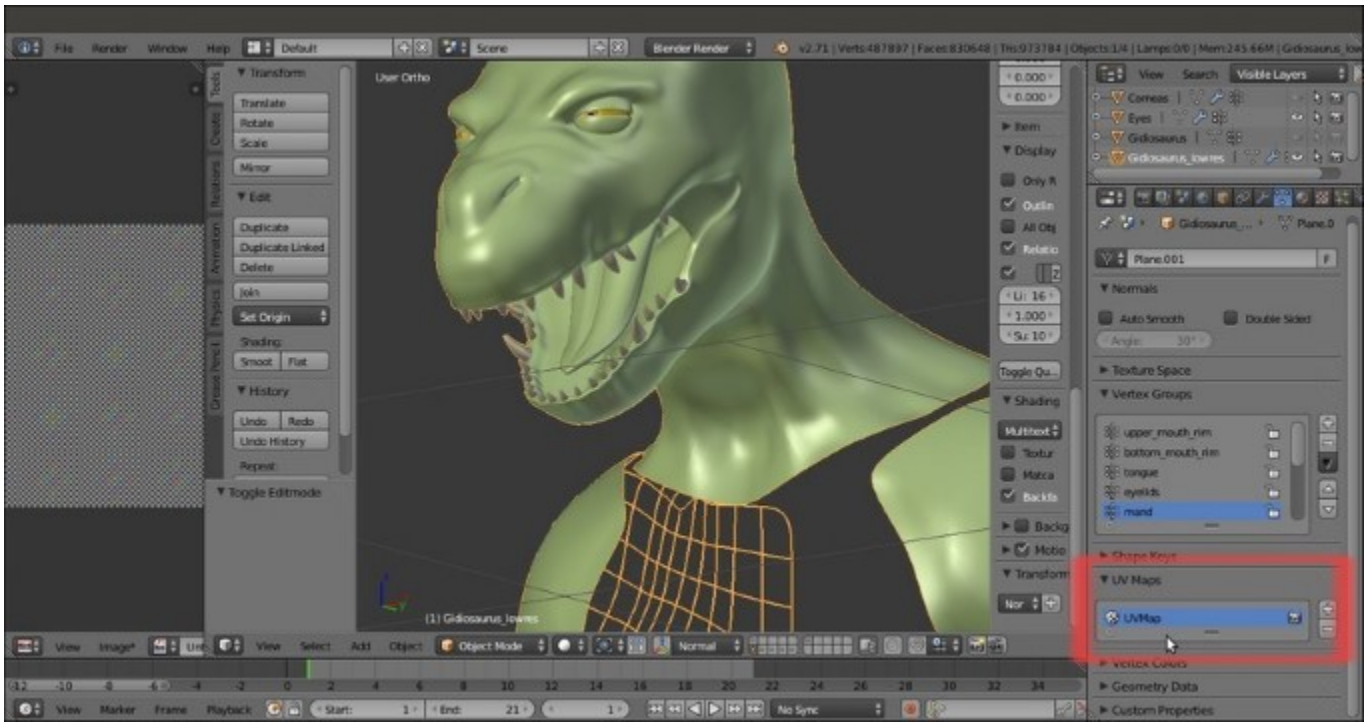


The updated unwrap

Note that you need to recalculate the unwrap for all the islands involved in the mesh's modification, and then save the file.

Setting up additional UV layers

Up until now, we have set just one UV layer whose name is, by default, **UVMap** (go to the **Object Data** window and look under the **UV Maps** subpanel):



The UV Maps subpanel with the UV Map coordinates layer

Actually, in Blender, it is possible to set more than one UV coordinates layer on the same object in order to mix different UV projections that can eventually also be baked into a single image map.

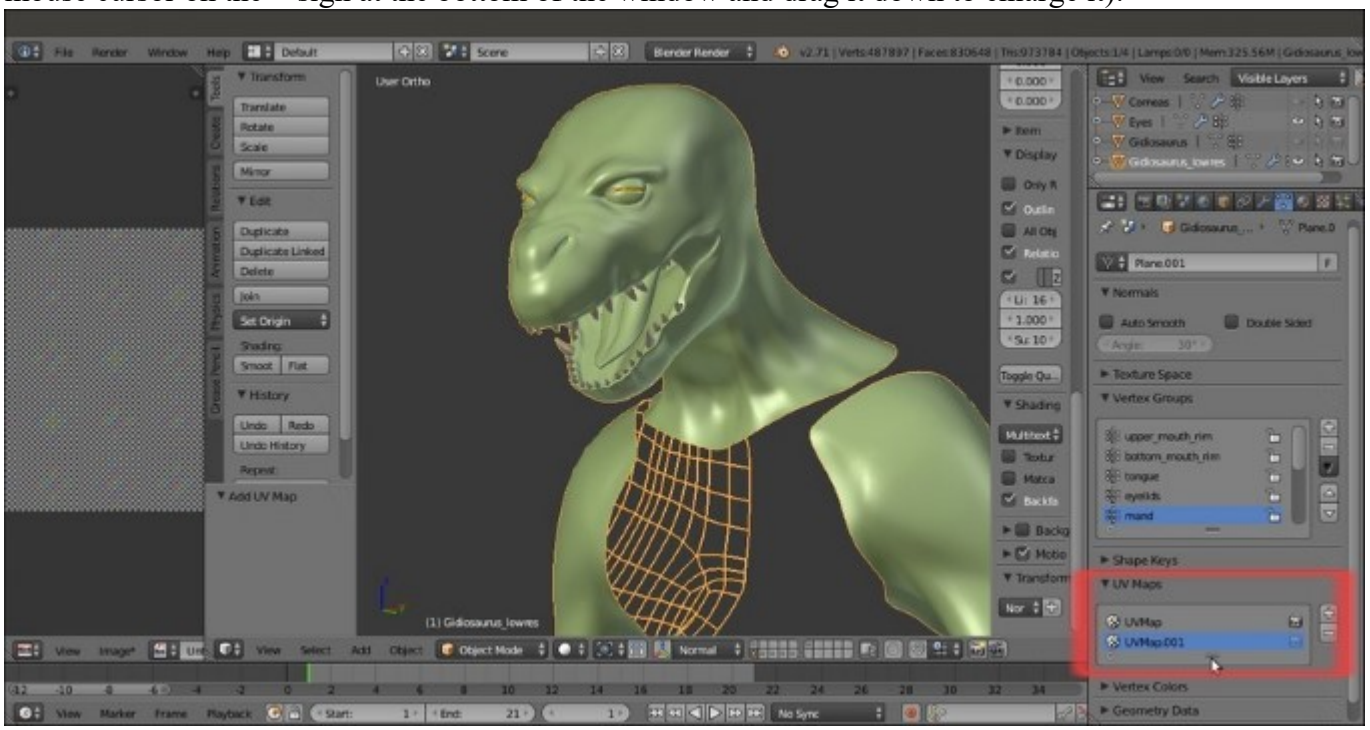
The names of the UV layers under the **UV Maps** subpanel are important, because they specify which one of the projections a material has to use for the mapping of a texture. By clicking on the + icon to the side of the **UV Maps** subpanel, it is possible to add a new UV layer (whose name, in this case, will be **UVMap.001** by default; of course it's possible to change these names by using *Ctrl* + clicking on them and typing the new ones).

Getting ready

We are now going to add a new UV layer to the **Gidiosaurus** object:

1. Ensure that the **Gidiosaurus** object is selected and go to the **Object Data** window under the main **Properties** panel to the right-hand of the screen.
2. Go to the **UV Maps** subpanel and click on the + icon to the right-hand side of the names window; a new UV layer is added to the list, right under the first one, and its name is

UVMap.001 (in case you don't see it, it may be because the window is too small; just put the mouse cursor on the = sign at the bottom of the window and drag it down to enlarge it):



The new UV coordinates layer

3. Use *Ctrl* + left-click on the **UVMap.001** item and rename it as **UVMap_scales**. Then, press *Enter* to confirm.

How to do it…

Now we must set the projection of the UV layer:

1. Go into **Edit Mode**, switch to the **Face** selection mode, put the mouse pointer on the mesh, and press the *L* key to select all the faces of the skin of the **Gidiosaurus** mesh.
2. Go to the **UV/Image Editor** window, select all the visible islands and unpin them (*Alt* + *P*).
3. Click on the **Image** item on the toolbar and select the **Open Image** item in the pop-up menu (or else, put the mouse cursor in the **UV/Image Editor** window and press *Alt* + *O*); browse to the `textures` folder and load the `scales_tiles.png` image.
4. With the mouse pointer in 3D view, press *U* and from the **UV Mapping** pop-up menu, select the **Cube Projection** item.
5. In the **UV/Image Editor** window, select all the islands and scale them **5** times bigger (*A* | *S* | *5* | *Enter*):

The Cube Projection mapping

6. Go out of the **Edit Mode** and into the **Properties** 3D view sidepanel, enable the **Textured Solid** item under the **Shading** subpanel to see the result of the unwrapping in the 3D viewport:

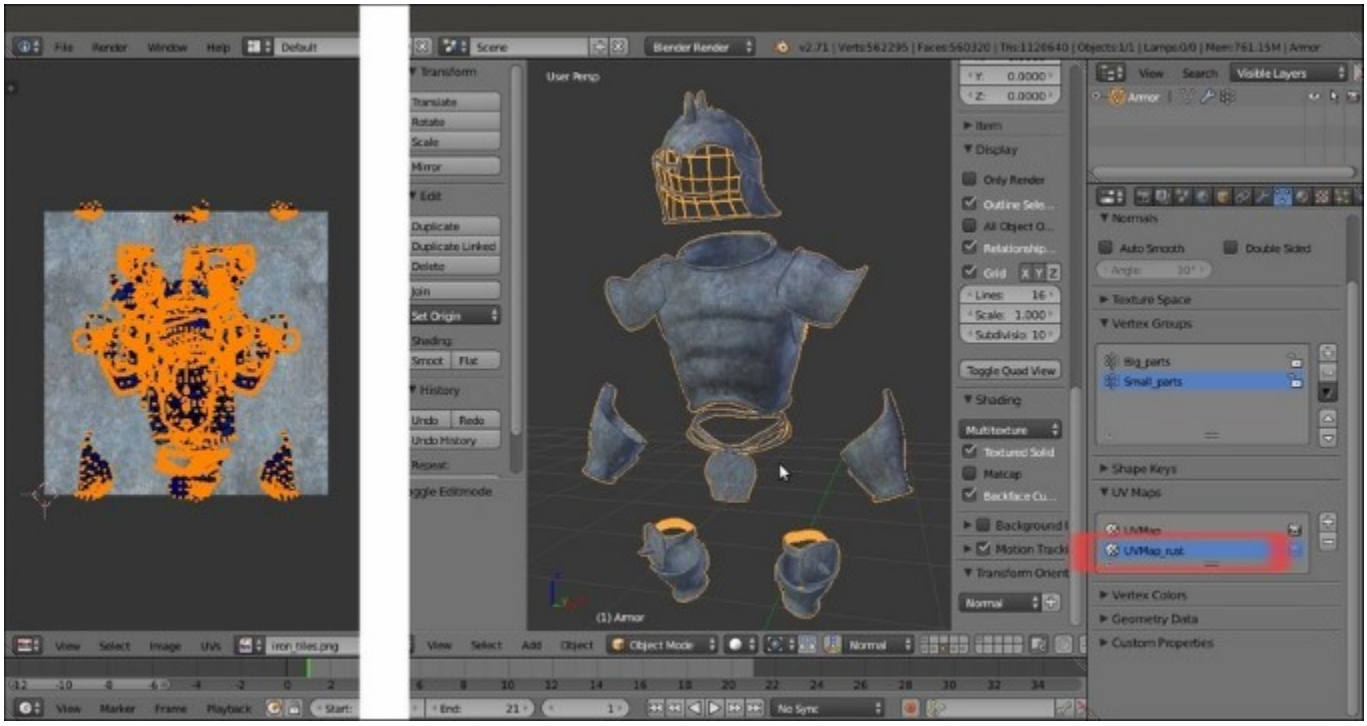
The scales_tiles.png image mapped on the model using the second UV coordinates layer

At this point, as you can see in the **UV Maps** subpanel, the **Gidiosaurus** object has **2** different UV coordinate layers, **UVMap** and **UVMap_scales**. We will use the **UVMap_scales** layer to map the scales image texture on the body and thereby to bake it on the first **UVMap** layer; this will be the one we'll use in the end for the rendering of the model. However, we'll see this in detail in the texturing and baking recipes.

Repeat the process for the **Armor**.

7. Add a new UV layer and rename it **UVMap_rust**; go into **Edit Mode**, select all the vertices and all the islands in the **UV/Image Editor** window, and load the `iron_tiles.png` image.
8. Switch to the **Face** selection mode, and in the 3D view, press *U* and select **Reset** (the last item) from the pop-up menu. Then press *U* again, and this time select the **Cube Projection** item.
9. Go out of **Edit Mode**.

As you can see, there are a few visible seams. This will be easily fixed during the texturing stage, but for the moment we are done:



The second UV coordinates layer for the Armor

Exporting the UV Map layout

In this last recipe, we are going to see how to export the UV coordinate layers outside Blender, in order to be used as a guide to paint textures inside any 2D image editing software.

Getting ready

We have seen that the **Gidiosaurus** object and also the **Armor** object have more than one UV coordinate layer, so the first thing to do is to be sure to have set the right layer as the *active* one.

To do this, simply click on the name of the chosen layer inside the **UV Maps** subpanel under the **Object Data** window; if you are in **Edit Mode**, by clicking on the different names, you can also see the different layers switch in real time in the **UV/Image Editor** window.

How to do it…

After you have selected the desired UV layer, do the following:

1. Click on the **UVs** item in the toolbar of the **UV/Image Editor** window, and from the menu, select the **Export UV Layout** item (the top item).
2. You can browse the directory where the `.blend` file is saved, as the directory opens, at the bottom-left side of the screen is the **Export UV Layout** option panel where you can decide on several items: the size and format of the exported image, and the opacity of the islands (by default, for mysterious reasons, it is set to **25 percent** rather than **100 percent**). Moreover, you can decide if you want to export all the islands of the selected object or only the visible ones, and also if you want the modifiers applied to the islands (for example, the **Subdivision Surface** modifier).
3. Browse to the folder where you decided to save the UV layout of your model, or click on the side of the path in the upper line after the slash, and write the name of a new directory. Press *Enter* and click on the pop-up panel with the **OK? Create New Directory** message to confirm (this actually creates a brand new directory).
4. Write the name of the UV layout in the second line and click on the **Export UV Layout** button at the top-right of the screen.

Note that if you want to export all the different tiles placed outside of the default **U0/V0** tile space, as illustrated in the *There's more…* section of the *Editing the UV islands* recipe, at least for the moment, you have to temporarily (using *Ctrl*) move each island at a time to the default **U0/V0** tile space and export it.

Chapter 6. Rigging the Low Resolution Mesh

In this chapter, we will cover the following recipes:

- Building the character's Armature from scratch
- Perfecting the Armature to also function as a rig for the Armor
- Building the character's Armature through the Human Meta-Rig
- Building the animation controls and the Inverse Kinematic
- Generating the character's Armature by using the Rigify add-on

Introduction

To be able to animate our character, we have to build the rig, which in Blender is commonly referred to as an **Armature**, and this is the *skeleton* that will deform the **Gidiosaurus** low resolution mesh.

The rigging process in Blender can be accomplished basically in two different ways:

- By building the **Armature** by hands from scratch
- By using the provided **Human Meta-Rig** or the **Rigify** add-on

Building the **Armature** manually by hand can be a lot of work, but in my opinion, is the only way to really learn and understand how a rig works; on the other hand, the **Rigify** add-on gives several tools to speed up and automate the rig creation process, and this in many occasions, can be very handy.

Building the character's Armature from scratch

So, the first recipe of this chapter is about the making of the **Armature** by hands for our **Gidiosaurus**.

Getting ready

In this first recipe, we are going to build by hands the **basic rig**, which is the skeleton made only by the **deforming bones**.

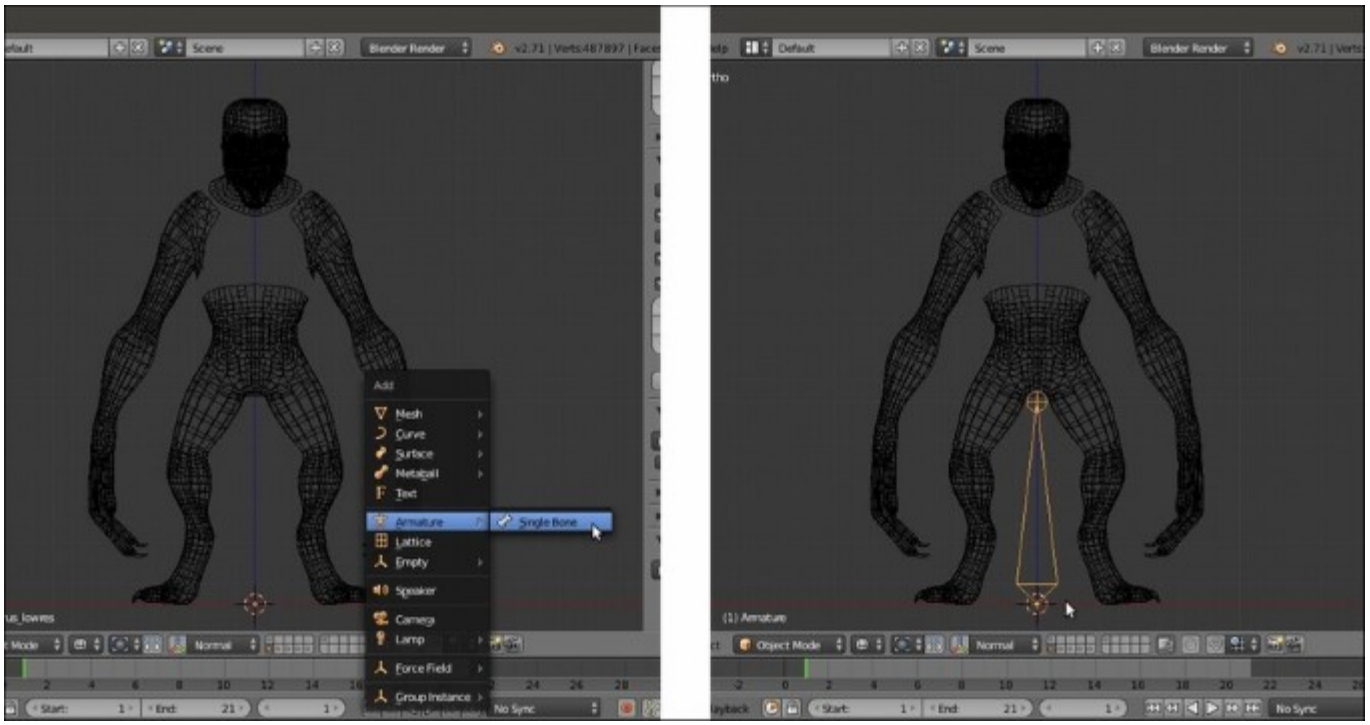
However, first, let's prepare a bit the file to be worked:

1. Start Blender and open the `Gidiosaurus_unwrap_final.blend` file.
2. Disable the **Textured Solid** and **Backface Culling** items in the 3D view **Properties** sidepanel, join the 3D window with the **UV/Image Editor** window, and click on the **11th** scene layer to have only the **Gidiosaurus** mesh visible in the viewport.
3. Go to the **Object** window under the **Display** subpanel and enable the **Wire** item. This will be useful in the process in order to have an idea of the mesh topology when in **Object Mode** and **Solid** viewport shading mode. However, for the moment, press the *Z* key to go in the **Wireframe** viewport shading mode.
4. Press *1* on the numpad to go in the **Front** view, and press *5* on the numpad again to switch to the **Ortho** view.
5. Save the file as `Gidiosaurus_rig_from_scratch_start.blend`.

How to do it…

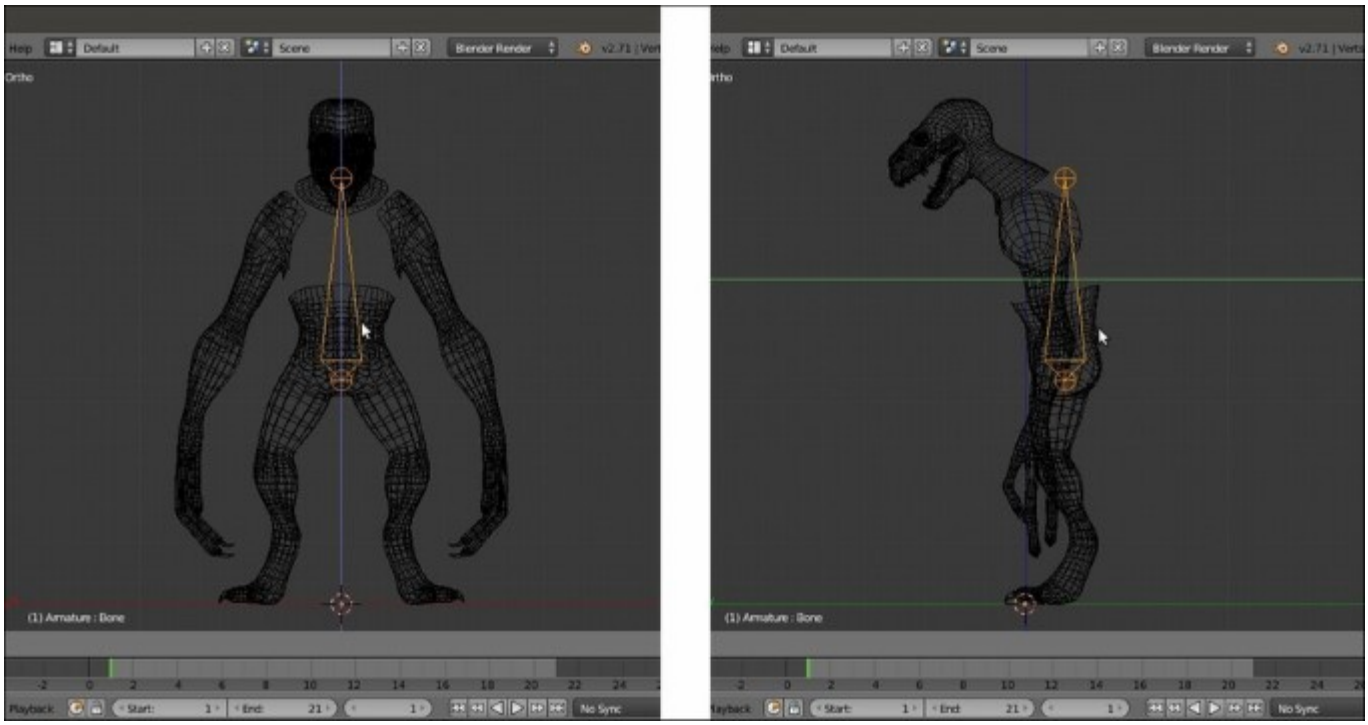
Let's start:

1. Be sure that the **3D Cursor** is at the origin pivot point of the **Gidiosaurus** mesh. Put the mouse pointer in the 3D view, press *Shift* + *A*, and in the **Add** pop-up menu, select **Armature** | **Single Bone**:



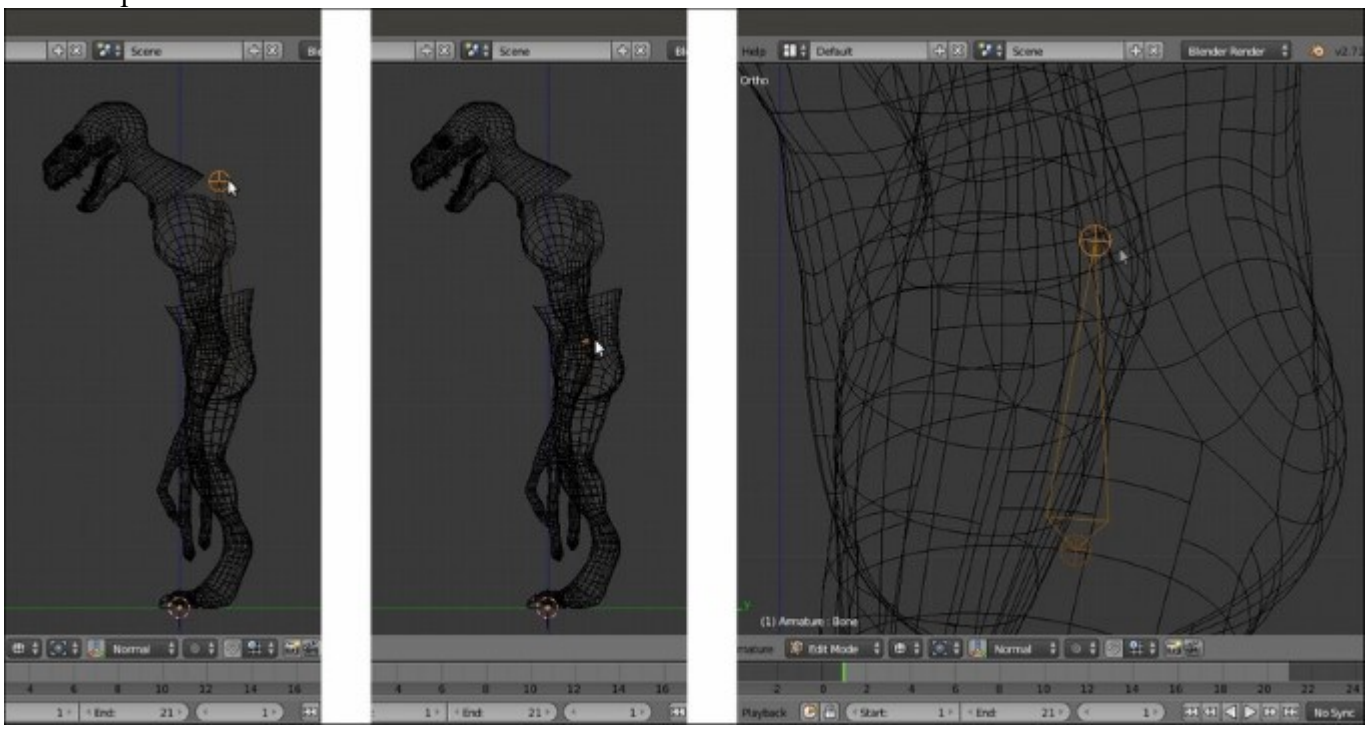
Adding the first Armature's bone

2. Press *Tab* to go into **Edit Mode** and select the whole bone by right-clicking on its **central part**; move the bone upwards to the **Gidiosaurus's** hips area (*G* | *Z* | *Enter* or left-click to confirm), and then go in the **Side** view (*3* key on the numpad) and center its position by moving it on the *y* axis:



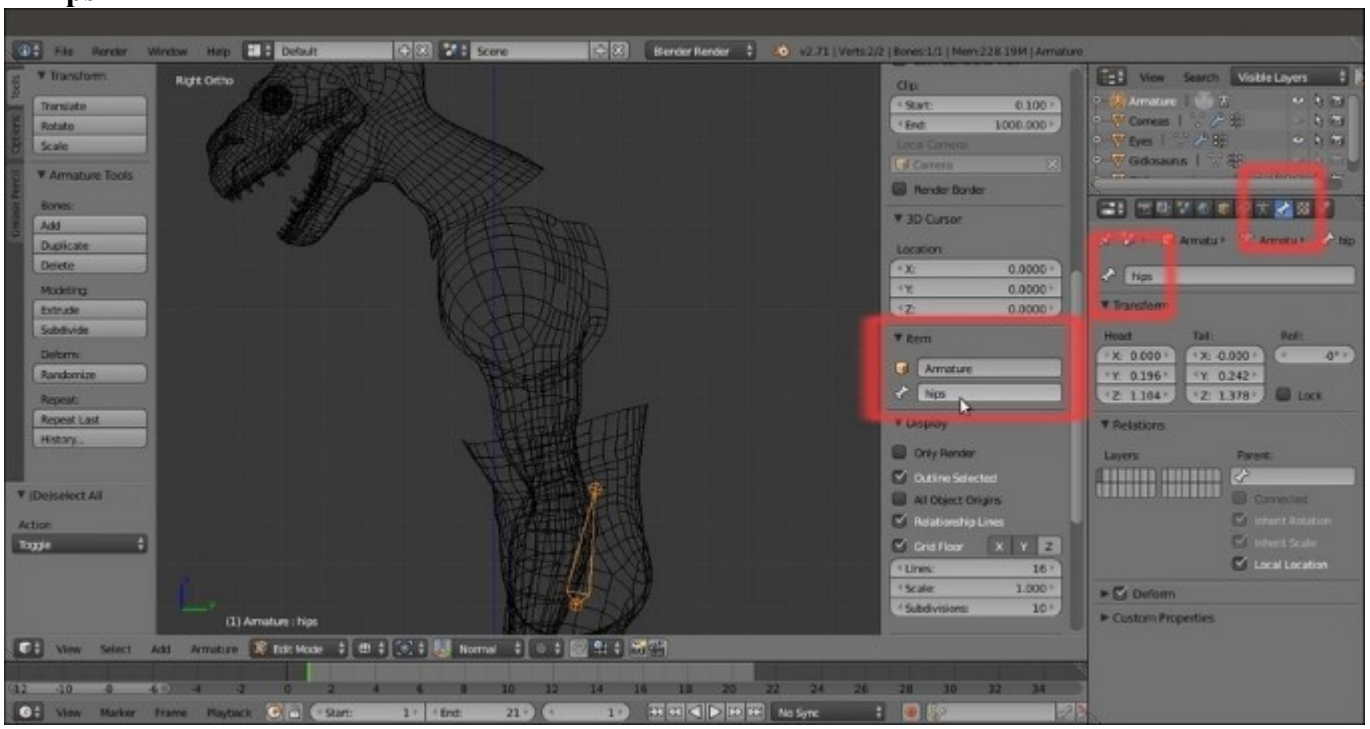
Positioning the bone in Edit Mode

3. Right-click on the **Head** of the bone to select it and by pressing *G* to move it, scale the bone size to fit the pelvis area:



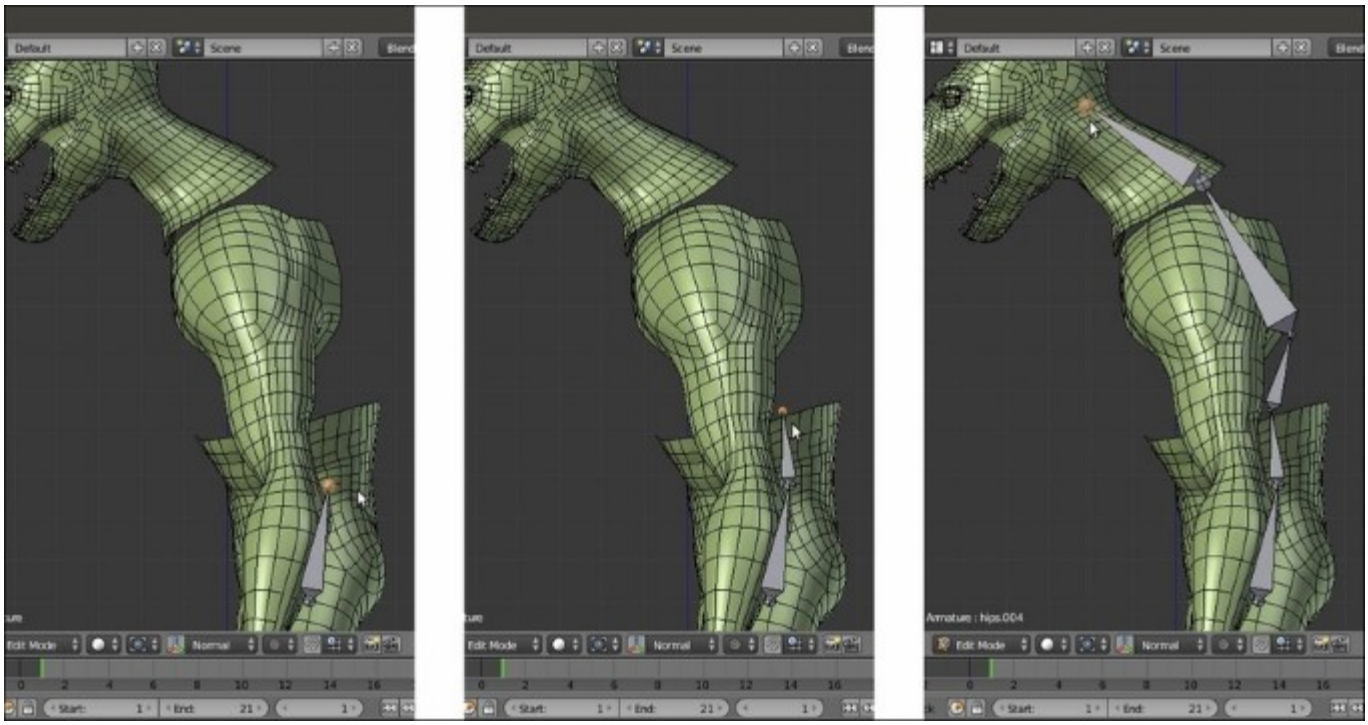
Scaling the bone in Edit Mode

4. Go to the **Item** subpanel under the 3D view **Properties** sidepanel, or in the **Bone** window under the main **Properties** panel to the right-hand side of the screen, and rename **Bone** (default name) as **hips**:



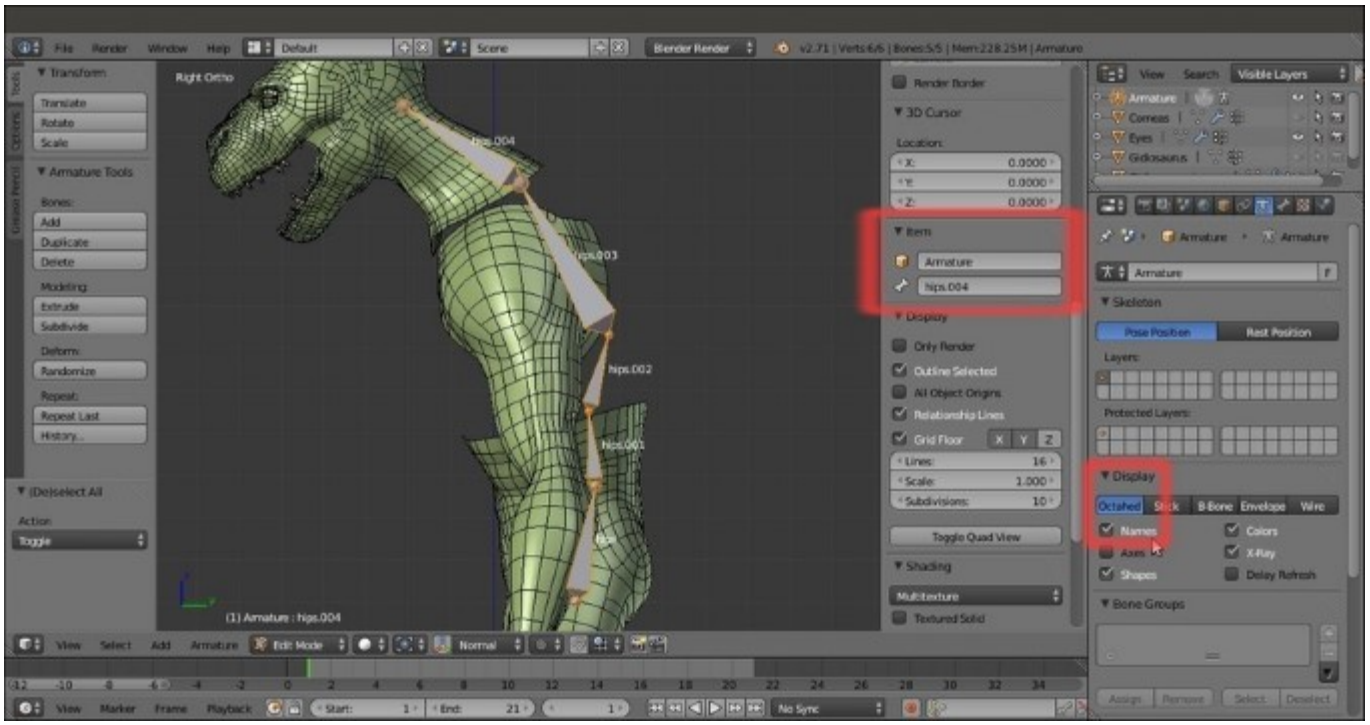
Renaming the bone

5. Press *Z* to go in the **Solid** viewport shading mode, and then go to the **Object Data** window and enable the **X-Ray** item under the **Display** subpanel.
6. With the tip of the bone selected (the **Head**), press the *E* key to extrude it. By this process, and by following the wire topology visible on the mesh as a guide, go upwards to build the **Gidiosaurus spine** (2 bones), **chest** (1 bone), and **neck** (1 bone); as much as possible, try to place the **Heads** (the tips/joints) of the bones aligned with the transversal edge-loops on the mesh's *articulation*:



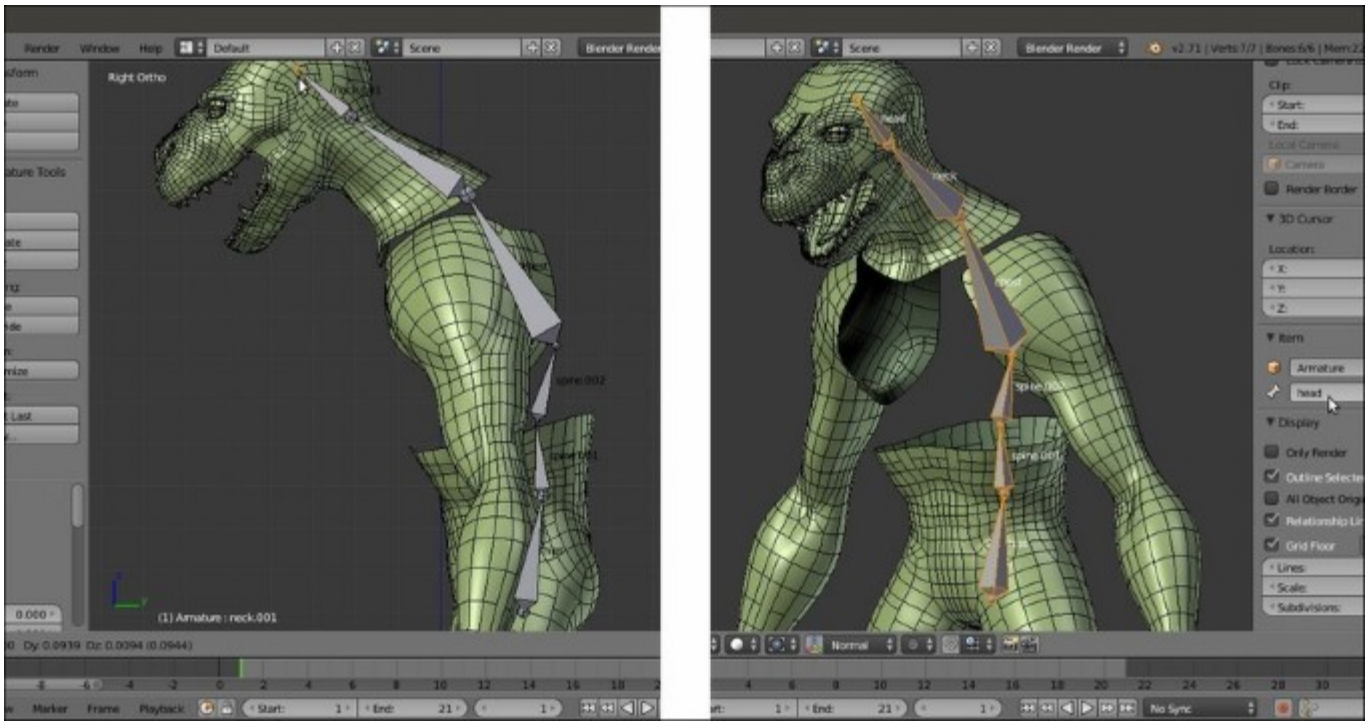
Extruding the bone to build the spine

7. Go again to the **Object Data** window under the **Display** subpanel, and enable the **Names** item (in the following screenshot, all the bones have been selected just to highlight them and their respective names). As you can see in the screenshot, the extruded bones get their names from the previous one, so we have **hips**, then **hips.001**, **hips.002**, and so on:



The bones' names

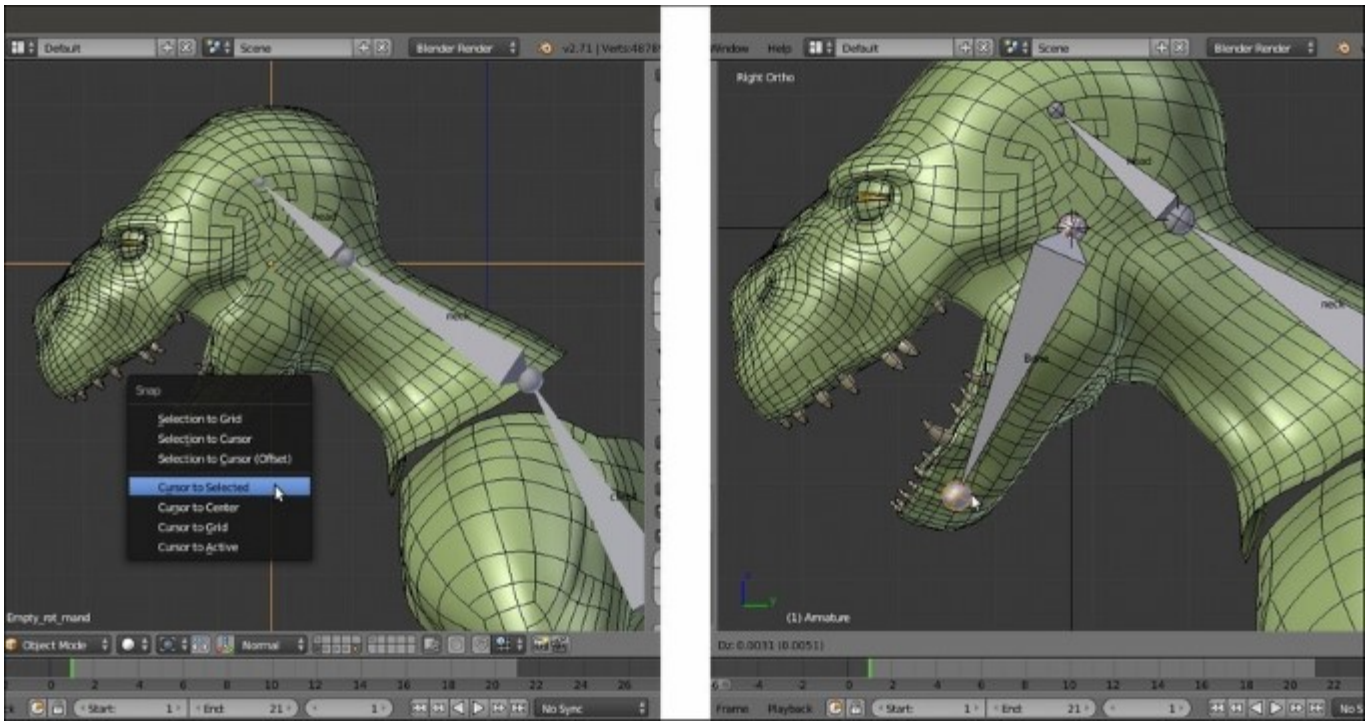
8. Select the **hips.001** bone and rename it **spine.001**; select the **hips.002** bone and rename it **spine.002**.
9. Select the **hips.003** bone and rename it **chest**; select the **hips.004** bone and rename it **neck**.
10. Select the tip of the **neck** bone and extrude it; rename the new bone (**neck.001**) as **head**:



The renamed bones and the head bone

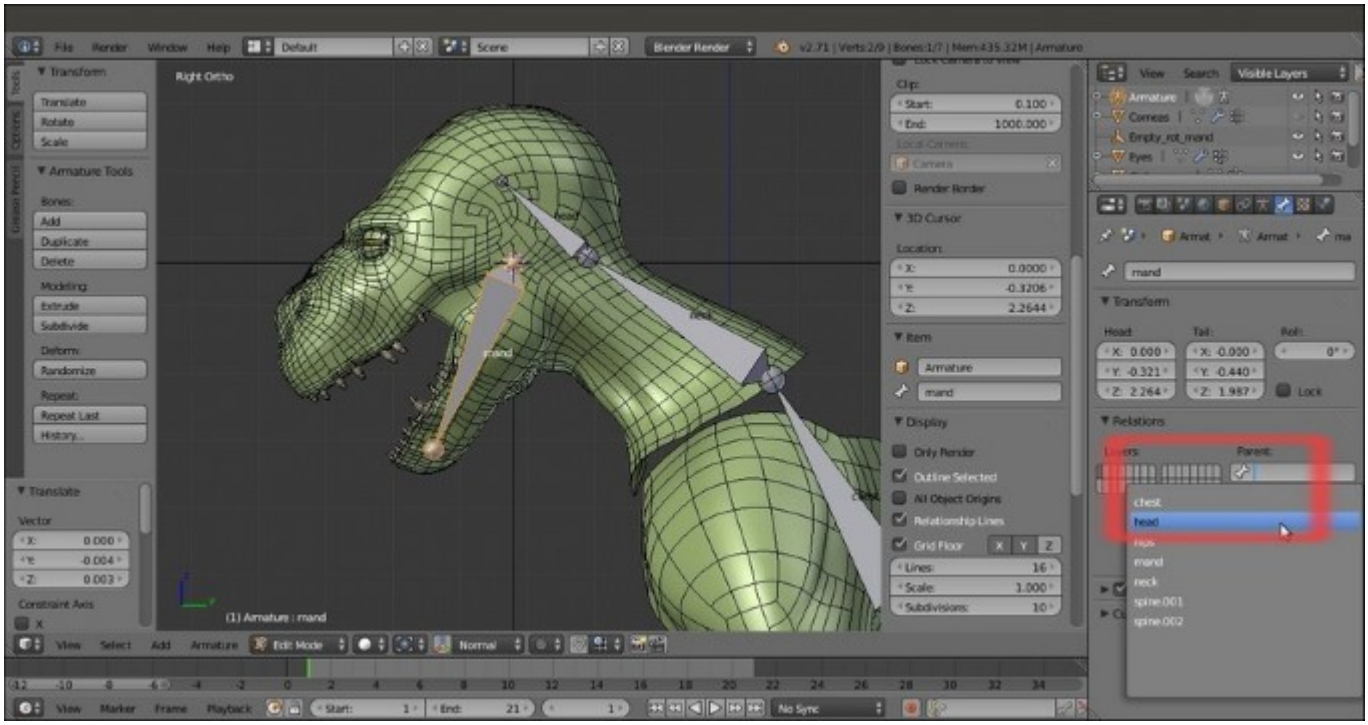
So, now we have built the **spine - neck** – **head** part of the **Armature**; actually, one thing is still missing: the bone to animate the **mandible**.

11. Press *Tab* to get out of **Edit Mode**. In the **Side** view, enable the **15th** scene layer on the 3D viewport toolbar, in order to show the **Empty_rot_mand** object; select it and press *Shift* + *S* to call the **Snap** pop-up menu. Then, select the **Cursor to Selected** item.
12. Reselect the **Armature** and go again into **Edit Mode**. Press *Shift* + *A* to add a new bone; move its **Head** to resize and fit it inside the **mandible** of the **Gidiosaurus**:



The mandible's bone

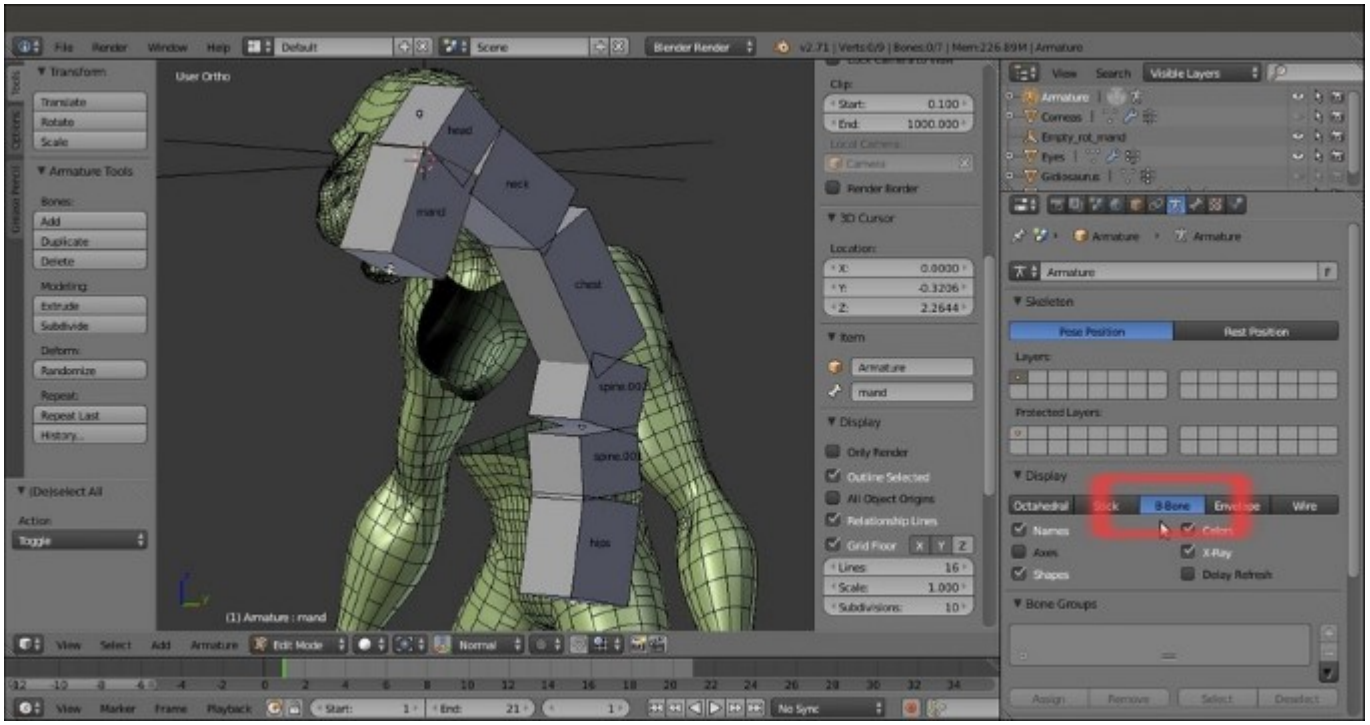
13. Rename it **mand** and in the **Bone** window under the main **Properties** panel, in the **Relations** subpanel, click on the **Parent** slot to select the **head** item from the pop-up menu with the bones list. Leave the **Connected** item **unchecked**:



The Parent slot and the pop-up menu to select the parent bone

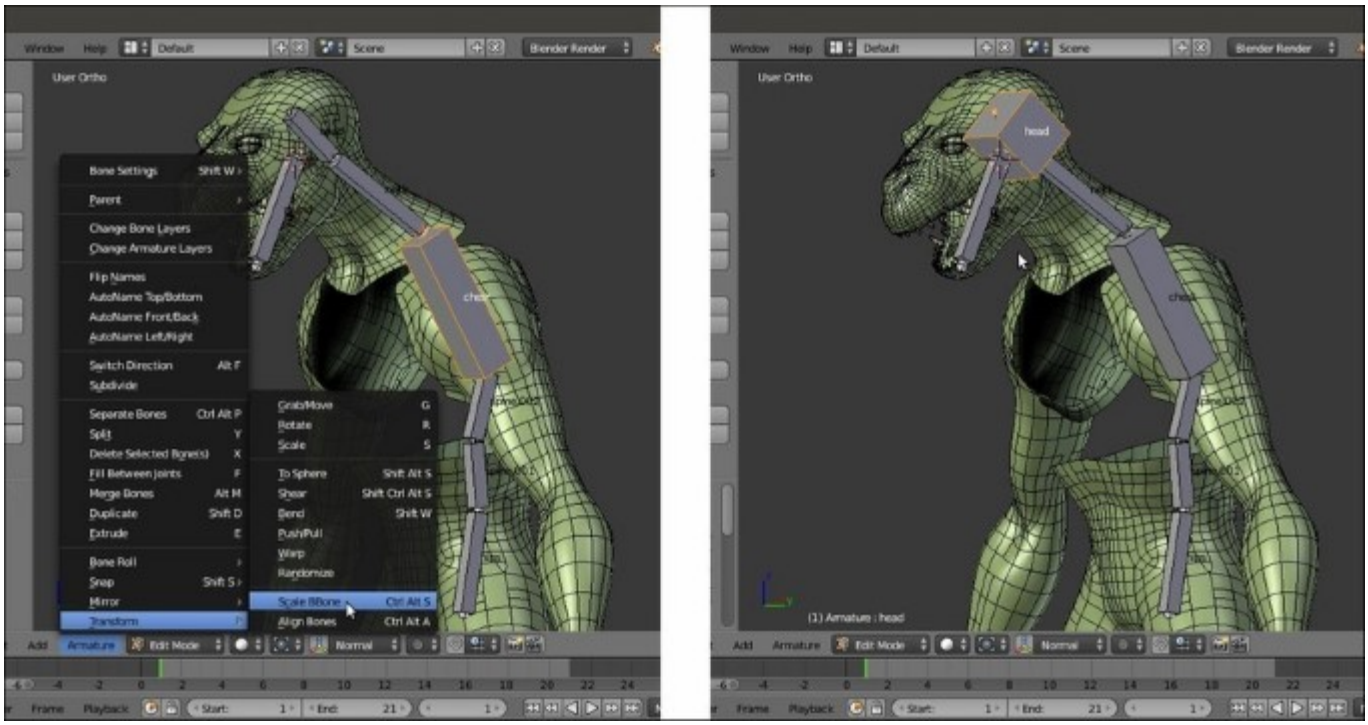
At this point, we can already see some particular setting to be applied to the bones.

14. Go to the **Object Data** window under the **Properties** panel and in the **Display** subpanel, switch from the default **Octahedral** to the **B-bone** button:



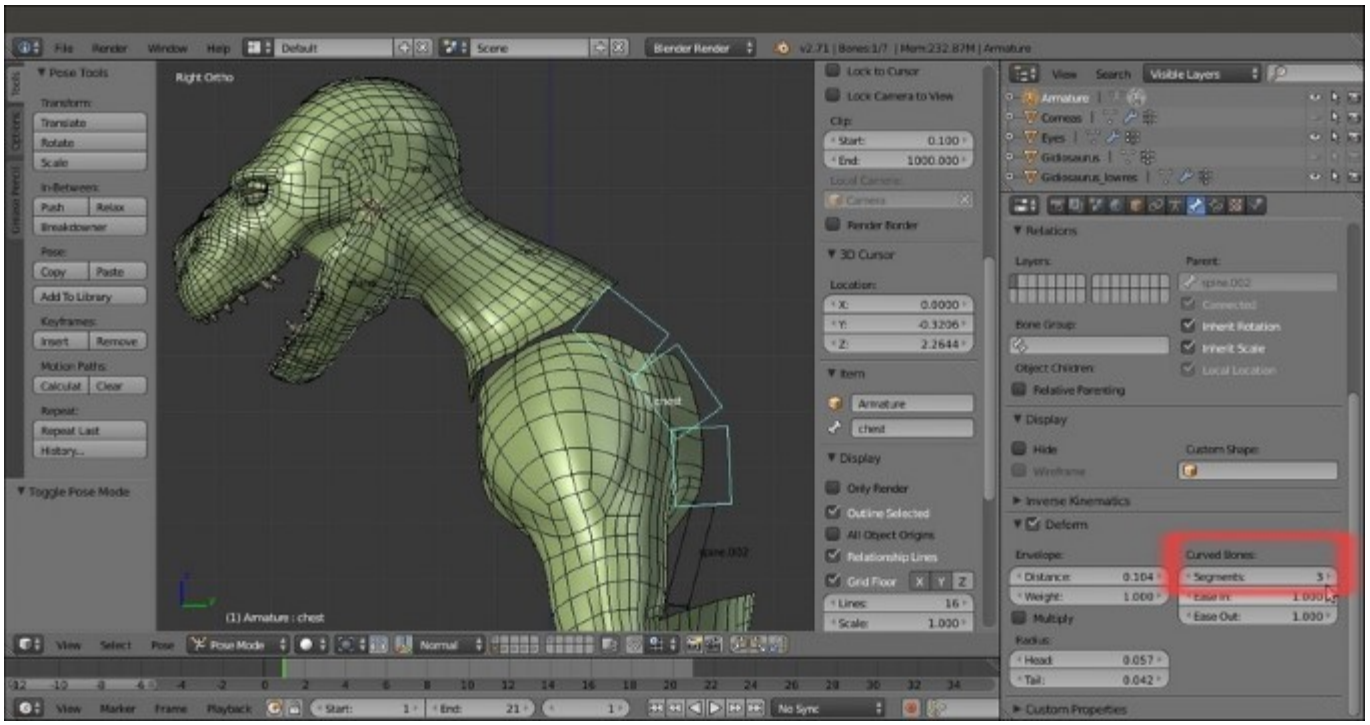
The bones visualized as B-bones

15. Press *A* to select all the bones, and then press *Ctrl* + *Alt* + *S* (or go to the **Armature** item in the window toolbar, and then go to **Transform** | **Scale Bbone**) and scale the **B-bones** to **0.200** (hold the *Ctrl* key to constrain the scaling values; the B-bones scaling works both in **Edit Mode** and **Pose Mode**).
16. Select only the **chest** bone and scale it bigger to **2.500**; select the **head** bone and scale it to **4.000**:



The B-bones scaled for better visualization

17. Go to the **Object** window and under the **Display** subpanel, click on the **Maximum Draw Type** slot (set to **Textured** by default) and switch it to **Wire**.
18. Press *Ctrl* + *Tab* to switch the **Armature** directly from **Edit Mode** to **Pose Mode**. Right-click on the **chest** bone to select it and go to the **Bone** window under the main **Properties** panel; in the **Deform** subpanel, set **Segments** under the **Curved Bones** item to **3**:

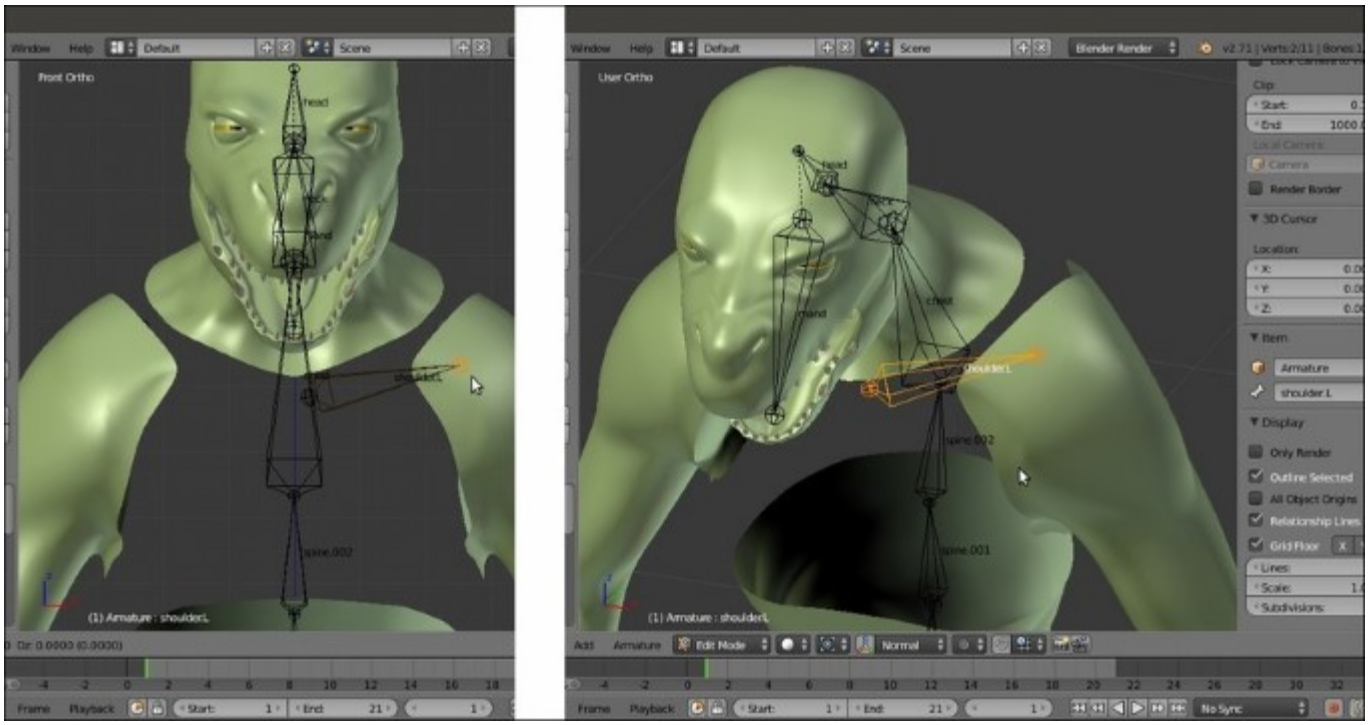


The chest B-bone with 3 curved segments

19. Select the **spine.002** and **spine.001** bones and set **Segments** to **2**. Select the **neck** bone and set **Segments** to **3**.
20. Select the **Gidiosaurus** mesh, go to the **Object** window, and disable the **Wire** item under the **Display** subpanel:

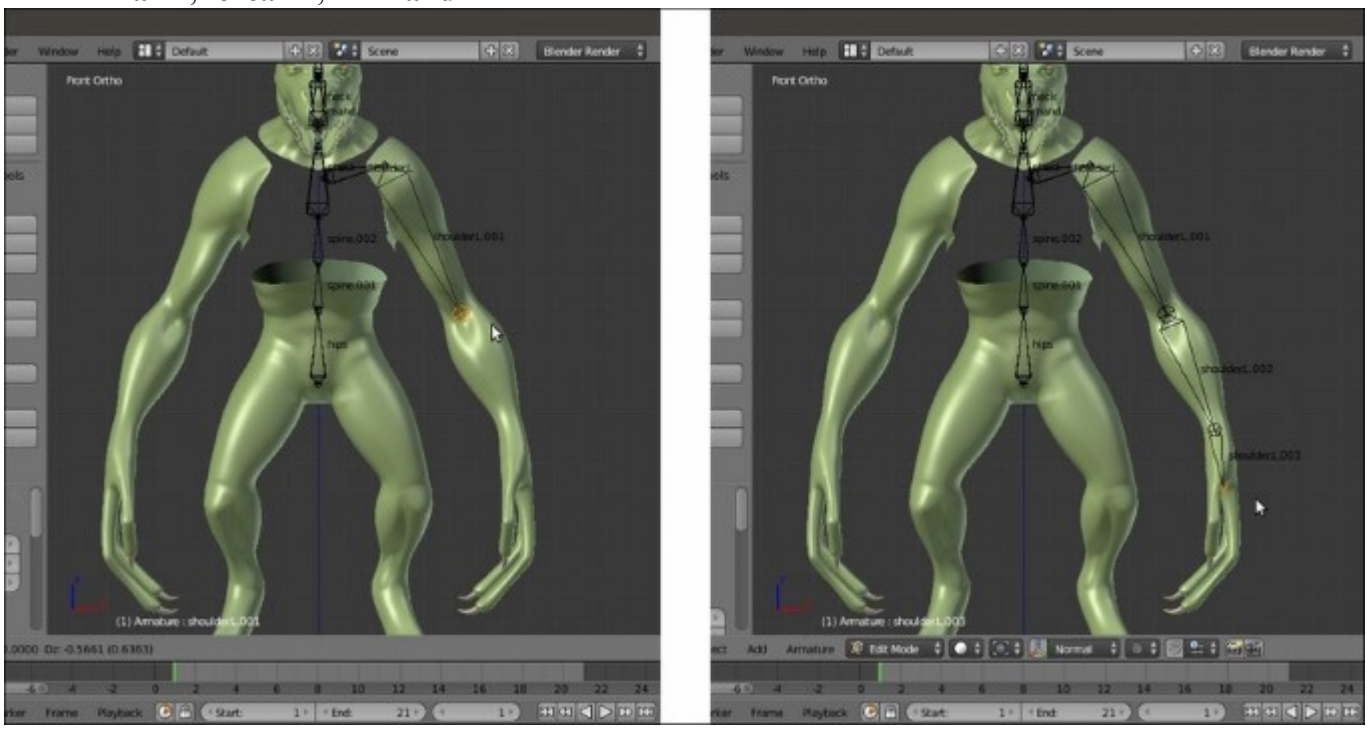
The rig so far

21. Press *Ctrl* + *Tab* to go out of the **Pose Mode**, and then *Shift* + *S* | **Cursor to Selected** to put the **3D Cursor** at the *rig/mesh/center of the scene* pivot point.
22. Press *Tab* to go into **Edit Mode** and press the *1* key on the numpad to go in the **Front** view; go to the **Object Data** window, under the **Display** subpanel, and switch back from **B-bone** to **Octahedral** (even if the visualization mode is different, the bones set as **B-Splines** still keep their *curved* properties in **Pose Mode**).
23. Press *Shift* + *A* to add a new bone at the cursor position. Move and resize it to put it as the **clavicle** bone—almost horizontal and slightly backward oriented, on the left-hand side of the rig. Rename it **shoulder.L** and in the **Parent** slot under the **Relations** subpanel, select the **chest** item:



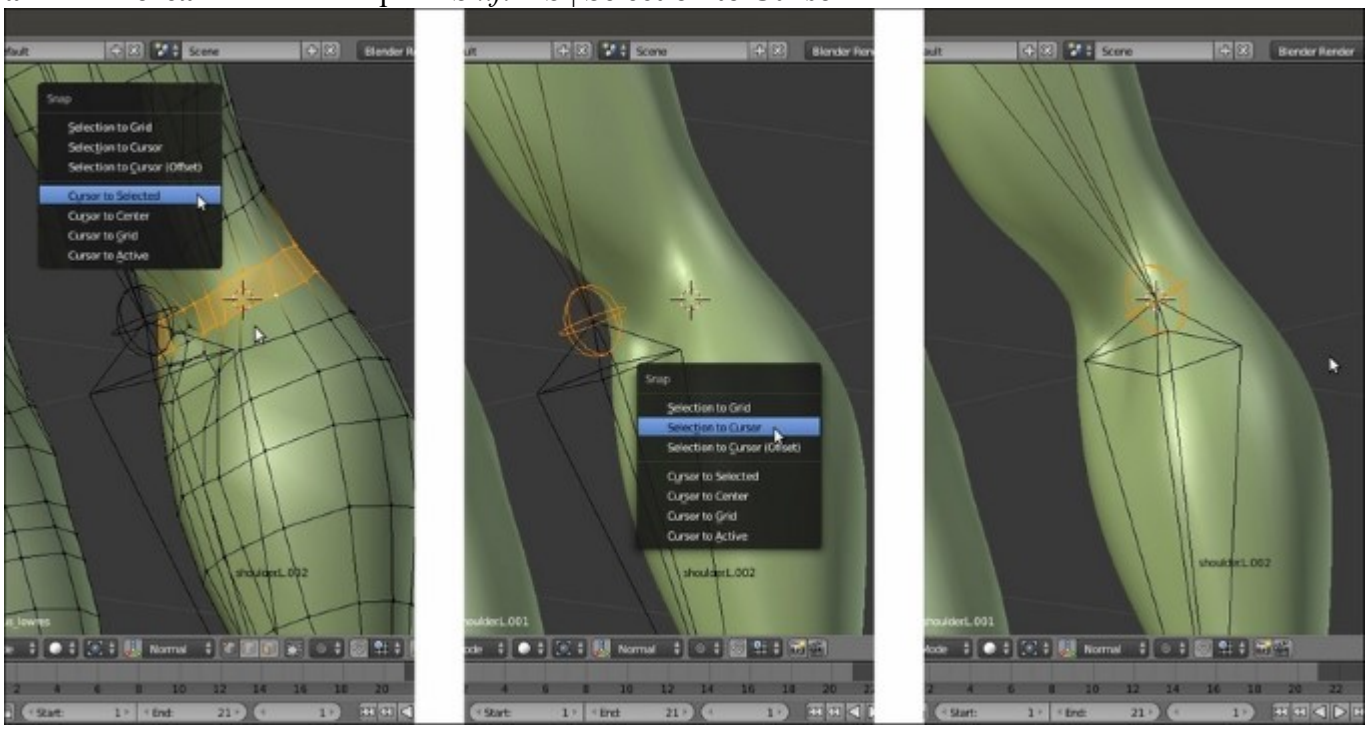
The shoulder.L bone

24. In the **Front** view, select the **Head** of the **shoulder.L** bone and extrude it **3** times to build the bones for **arm**, **forearm**, and **hand**:



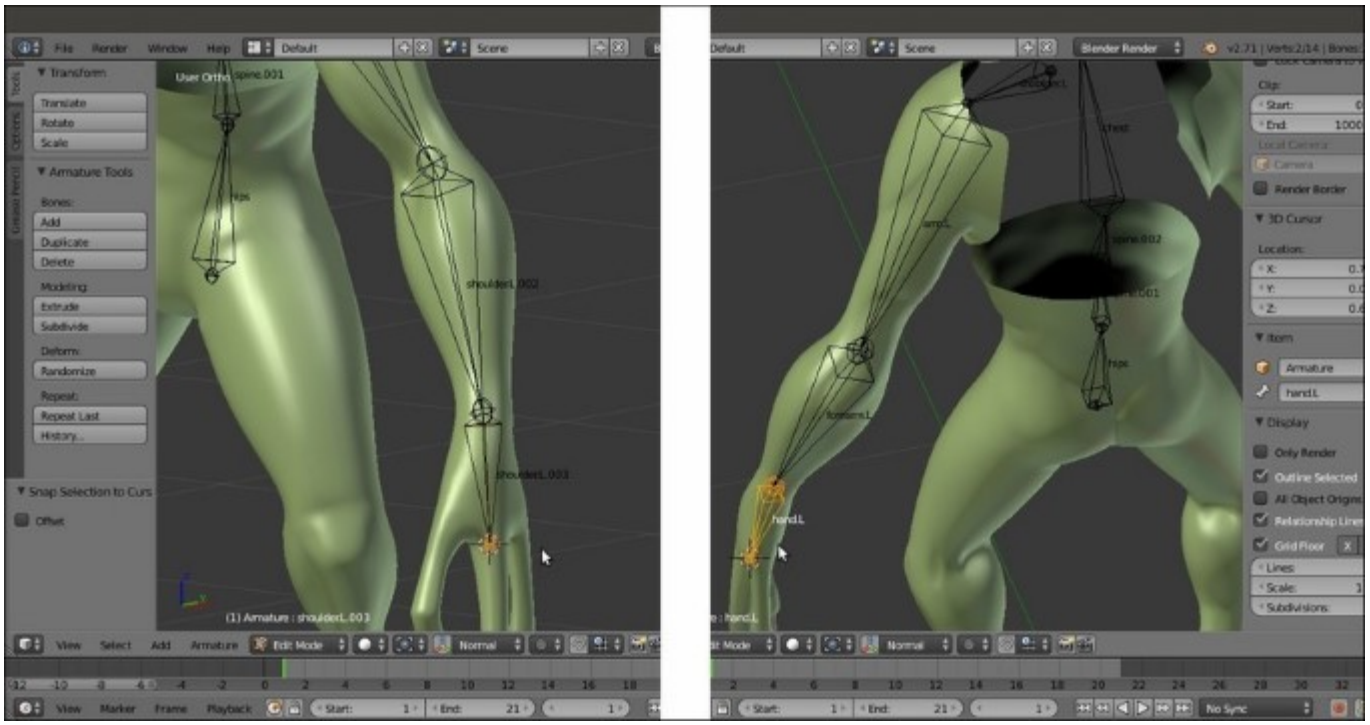
Extruding the shoulder.L bone to obtain the skeleton's bones for the arm

25. Now, exit **Edit Mode** and right-click to select the **Gidiosaurus** mesh; enter **Edit Mode**, select one or more edge-loops at the **elbow** level, and press *Shift* + *S* | **Cursor to Selected**.
26. Get out of **Edit Mode**, select the **Armature**; go into **Edit Mode**, select the joint between the **arm** and **forearm** bones and press *Shift* + *S* | **Selection to Cursor**:



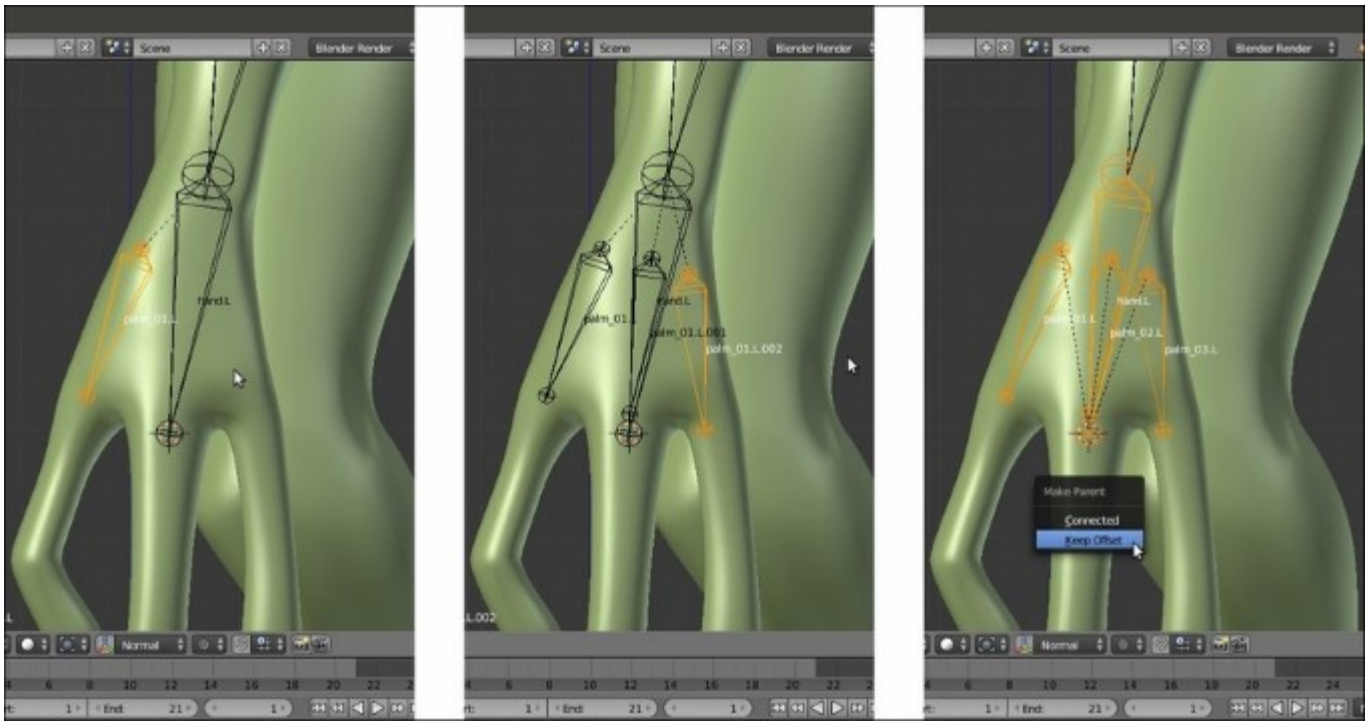
Placing the elbow joint

27. This is the easiest way to correctly align the rig joints with the mesh edge-loops. Do the same for the joint of the **wrist** and the bone of the **hand**; rename the bones as **arm.L**, **forearm.L**, and **hand.L**:



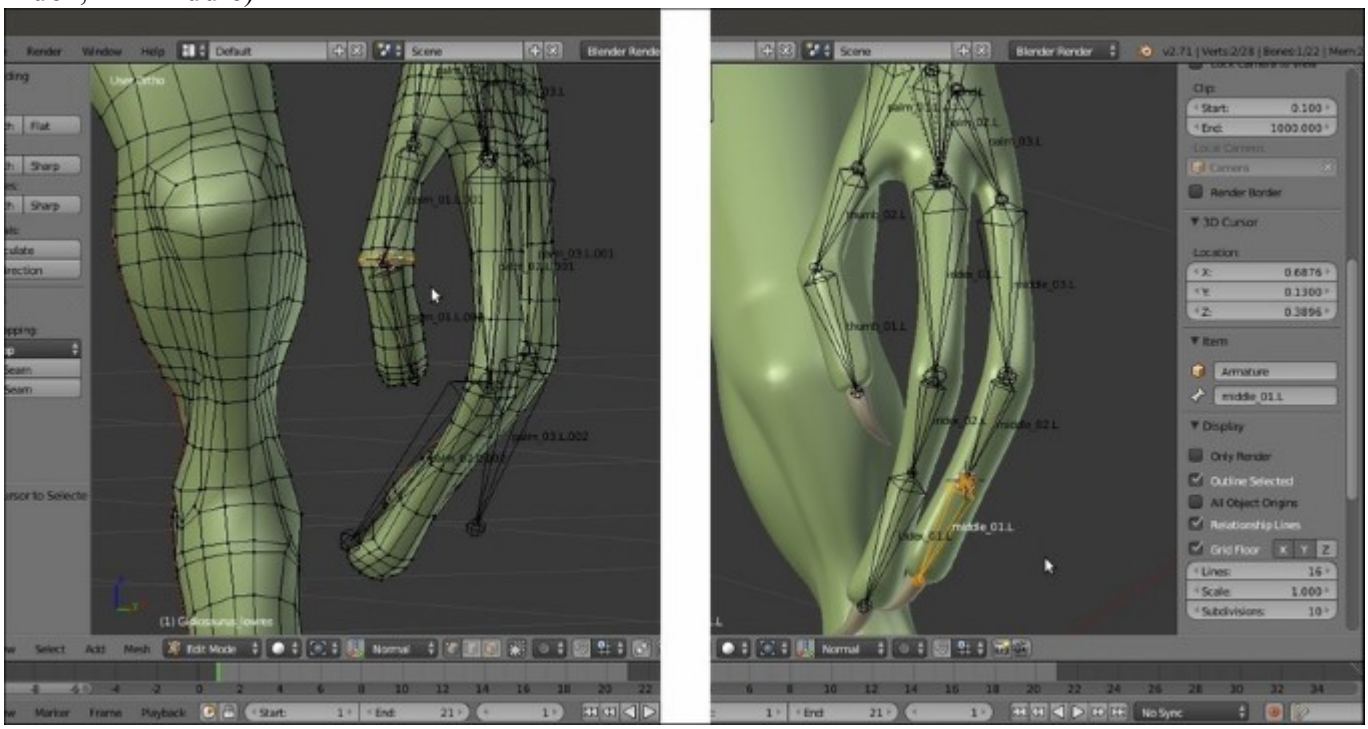
Fixing the position of the wrist joint and hand's bone

28. Select the **hand.L** bone and use *Shift* + *D* to duplicate it; scale it smaller (*S* | **0.600** | *Enter*), rename it **palm_01.L**, and move it above the joining of the **palm** with the **thumb**. Use *Shift* + *D* to duplicate it **2** more times and move the new bones above the joining of the other two **fingers**; rename them **palm_02.L** and **palm_03.L**.
29. Use *Shift* to select the three **palm** bones and, as the last one, the **hand.L** bone; press *Ctrl* + *P* | **Keep Offset** to parent them (**not connected**) to the latter one:



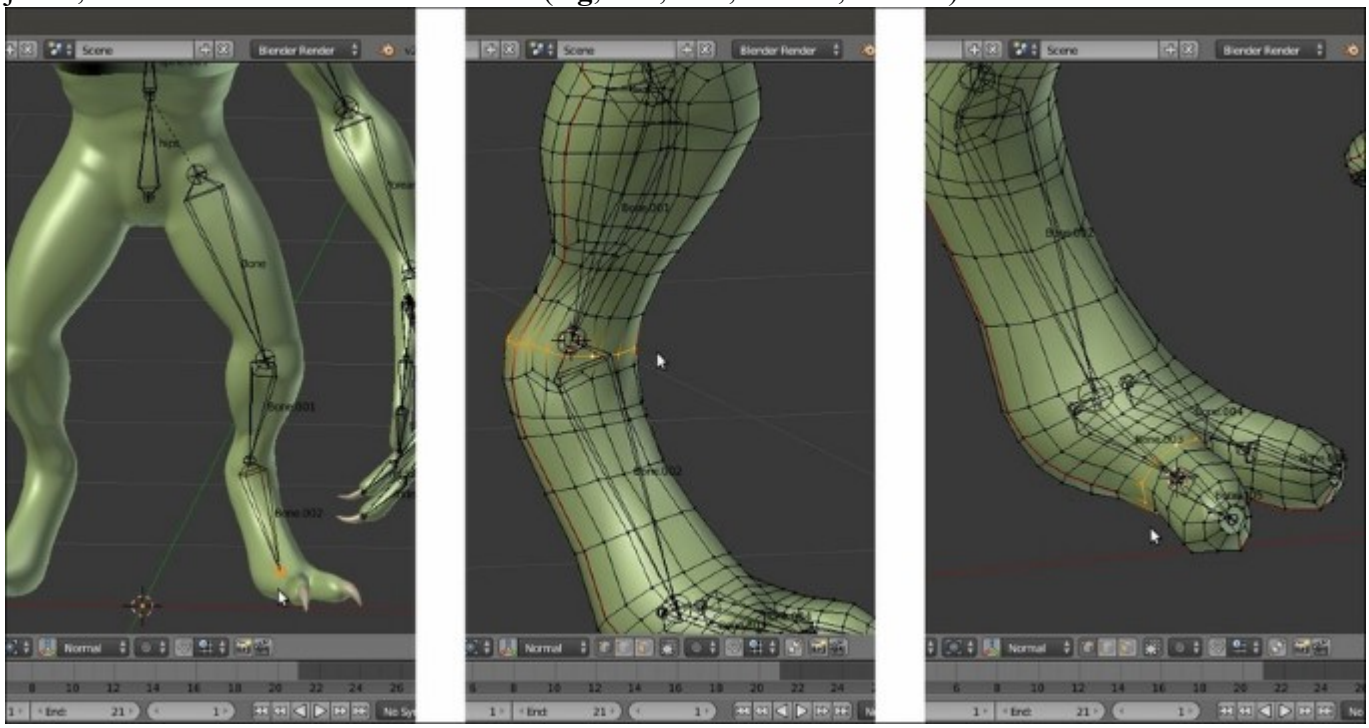
Adding the palm bones

30. Select (individually) the **Heads** of each **palm** bones and extrude the bones for the **fingers**; center their joints with the **3D Cursor**/**Snap** menu method and rename them properly (**thumb**, **index**, and **middle**):



The bones for the fingers

31. Again with the **3D Cursor** at the rig pivot point, add a new bone and shape it to fit inside the left **thigh**, the **Tail** at the top, close to the **hips** bone, and the **Head** at the **knee** location; select it and use *Shift* to select the **hips** bone, and then press *Ctrl* + *P* | **Keep Offset**. Extrude the bone's **Head** three times to build the **leg – foot** skeleton.
32. Extrude also the bones for the **toes** and repeat the previously described process to center the joints, and then rename all the new bones (**leg**, **calf**, **foot**, **toe inn**, and **ext**):



The bones for the leg and toes

In the preceding screenshots, you can see that we have hidden the **talons** vertices in **Edit Mode** (*H* key), in order to have the possibility to easily select the last edge-loops on **fingers** and **toes**.

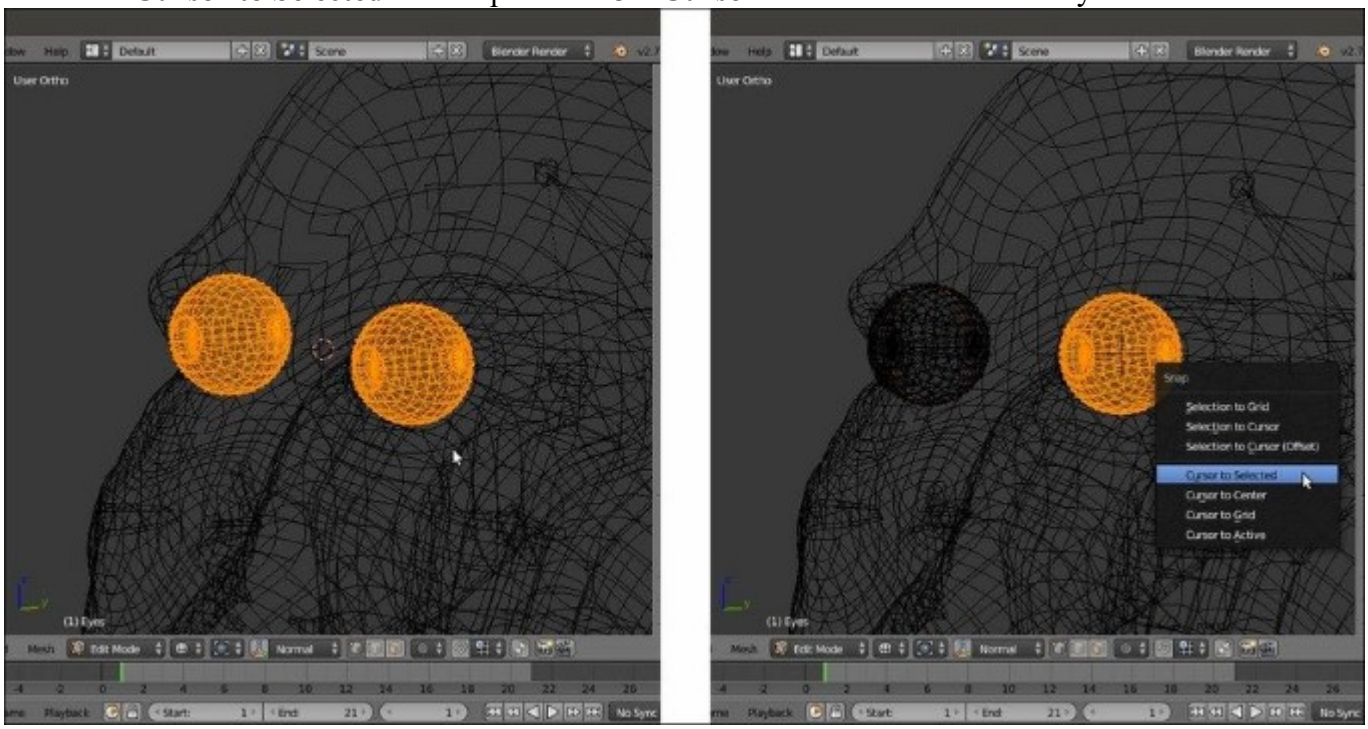
33. Save the file as `Gidiosaurus_rig_from_scratch_01.blend`.

Building the rig for the secondary parts

Now that we have completed the main body rigging system, it's time to build the rig for **eyes**, **eyelids**, and **tongue**:

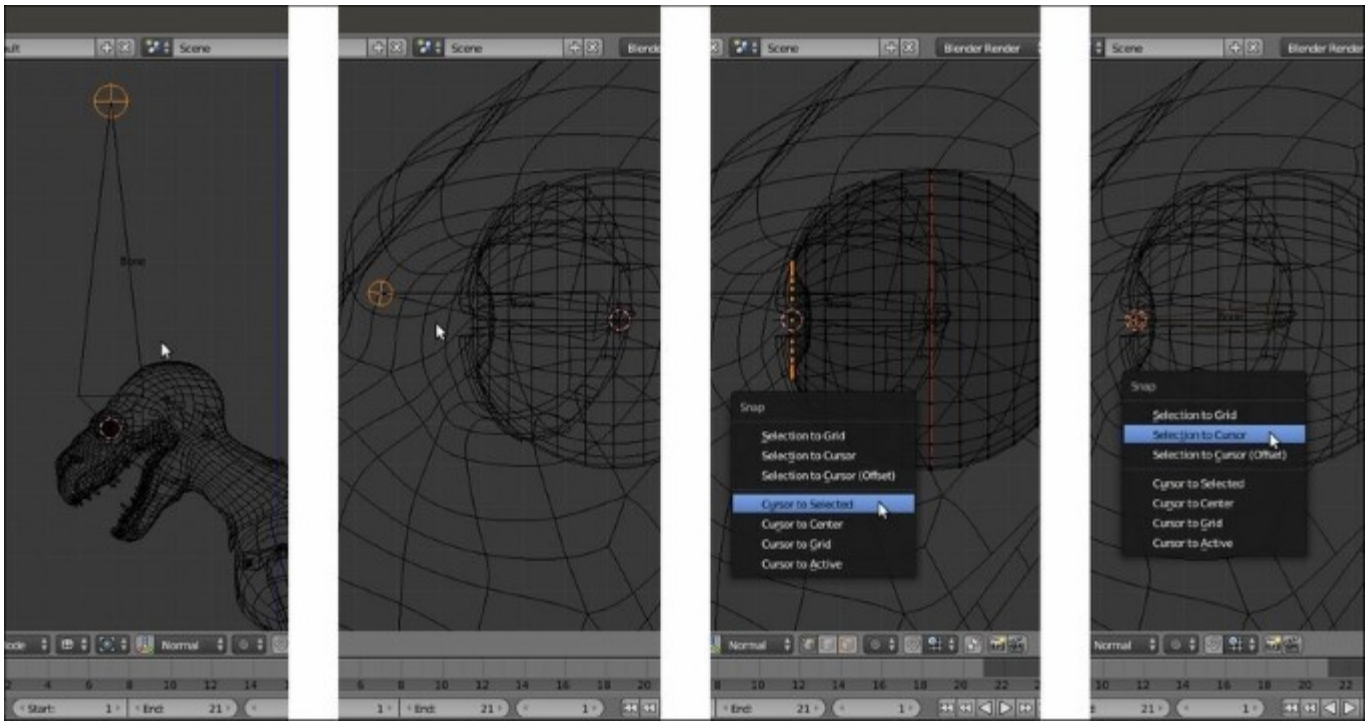
1. Get out of **Edit Mode** and select the **Eyes** item in the **Outliner**; press the dot (.) key on the numpad to center the view on the selected object, the *Z* key to go in **Wireframe** viewport shading mode, and *Tab* to go into **Edit Mode**.

2. Press the *A* key to select all the **eye** vertices and then box-deselect (the *B* key and the middle mouse button) the vertices of the **right eye**; use *Shift* + *S* to call the **Snap** pop-up menu and select the **Cursor to Selected** item to place the **3D Cursor** at the center of the left eye mesh:



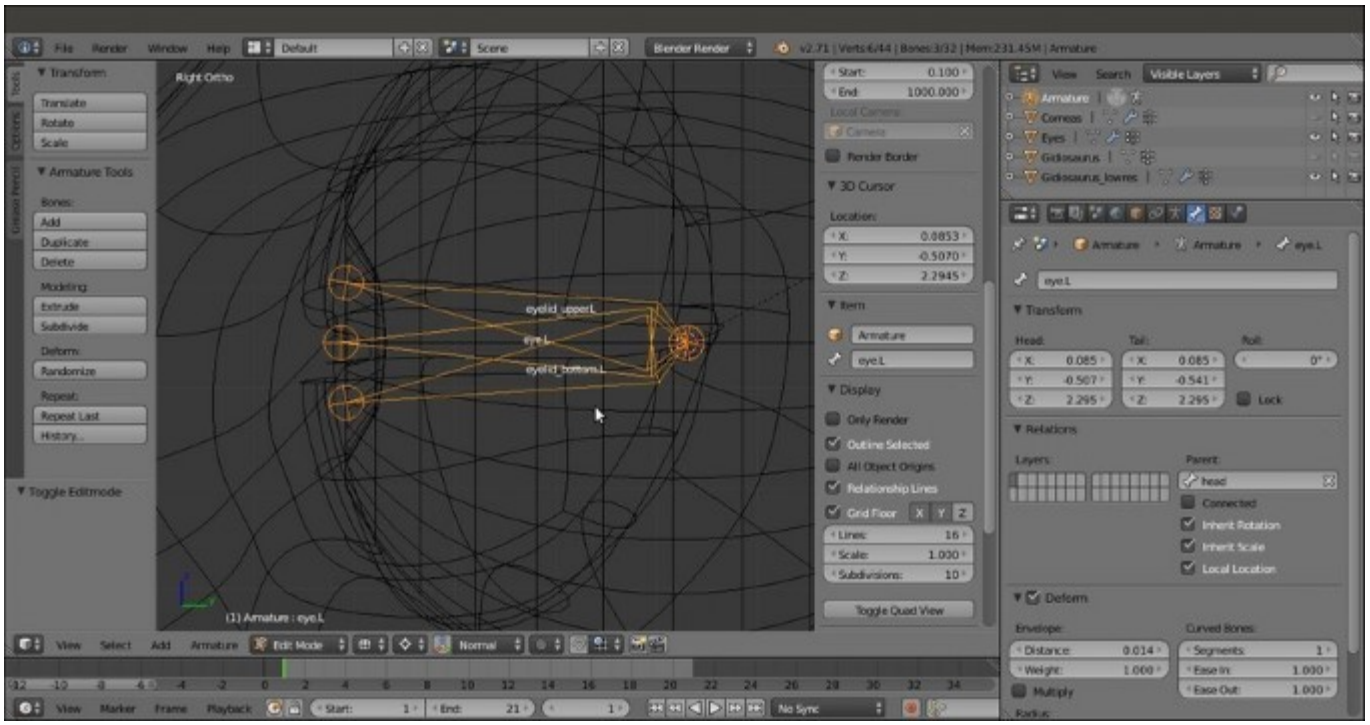
Placing the 3D Cursor at the center of the selected vertices

3. Get out of **Edit Mode**, press the *3* key on the numpad to go in the **Side** view and reselect the **Armature** item in the **Outliner**; press *Tab* to go into **Edit Mode**, and then use *Shift* + *A* to add a new bone at the cursor position. Press *G* to grab the already selected **Head** of the new bone and move it close to the center of the **eye** to resize it smaller.
4. Get out of **Edit Mode** and select the **Eyes** item; enter **Edit Mode** and deselect all the vertices except for the external last **iris** edge-loop. Then, press *Shift* + *S* | **Cursor to Selected** and get out of **Edit Mode**.
5. Again, select the **Armature**, go into **Edit Mode**, be sure that the **Head** of the new bone is still selected, and press *Shift* + *S* | **Selection to Cursor**:



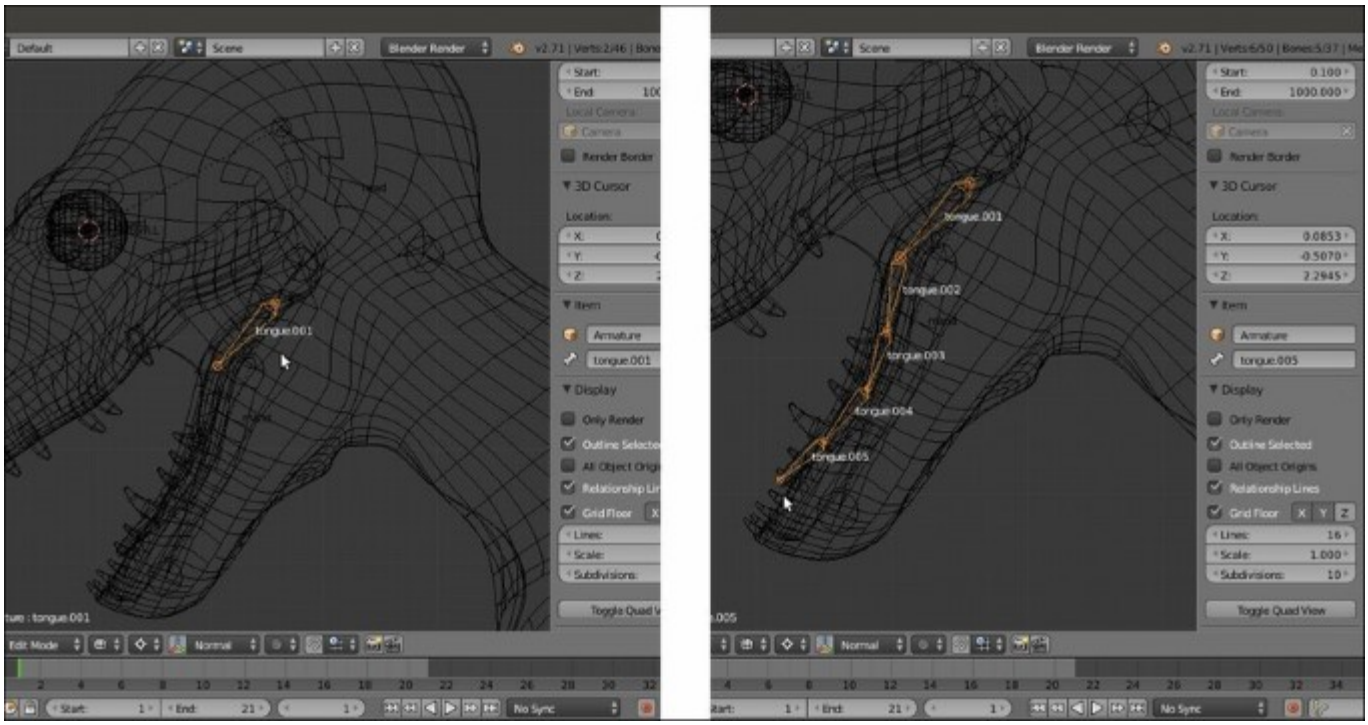
Placing the bone's head at the iris center location

6. Rename the new bone as **eye.L** and in the **Relations** subpanel under the **Bone** window, parent it to the **head** bone (not **Connected**), or use *Shift* to select the **eye.L** and **head** bones and press *Ctrl* + *P* | **Keep Offset**.
7. Now, select the **Tail** of the **eye.L** bone and press *Shift* + *S* | **Cursor to Selected** to put the **3D Cursor** on it, and then press the period (.) key to switch the **Pivot Point** around the **3D Cursor**; select the whole **eye.L** bone and use *Shift* + *D* to duplicate it, and soon after, click with the right mouse button to leave the duplicated bone untouched; rotate it **10** degrees clockwise on the cursor position (*Shift* + *D* | right-click | *R* | *X* | **10** | *Enter*).
8. Rename the new bone as **eyelid_upper.L**.
9. Reselect the whole **eye.L** bone and repeat the duplication procedure; rotate the new duplicate **10** degrees counterclockwise (*Shift* + *D* | right-click | *R* | *X* | **-10** | *Enter*).
10. Rename the new bone as **eyelid_bottom.L** (in the following screenshot, all the three new bones—**eyelid_upper.L**, **eye.L**, and **eyelid_bottom.L**—have been selected just to enhance their visibility):



The bones for the eye and eyelids

11. Now, duplicate the **head** bone, resize it smaller, and move it to the joining of the **tongue** with the **inner mouth**; rename it from **head.001** to **tongue.001** and in the **Relations** subpanel, change its parenting from **neck** to **mand**.
12. Select the **Head** of the **tongue.001** bone and press the *E* key to extrude **4** new bones:



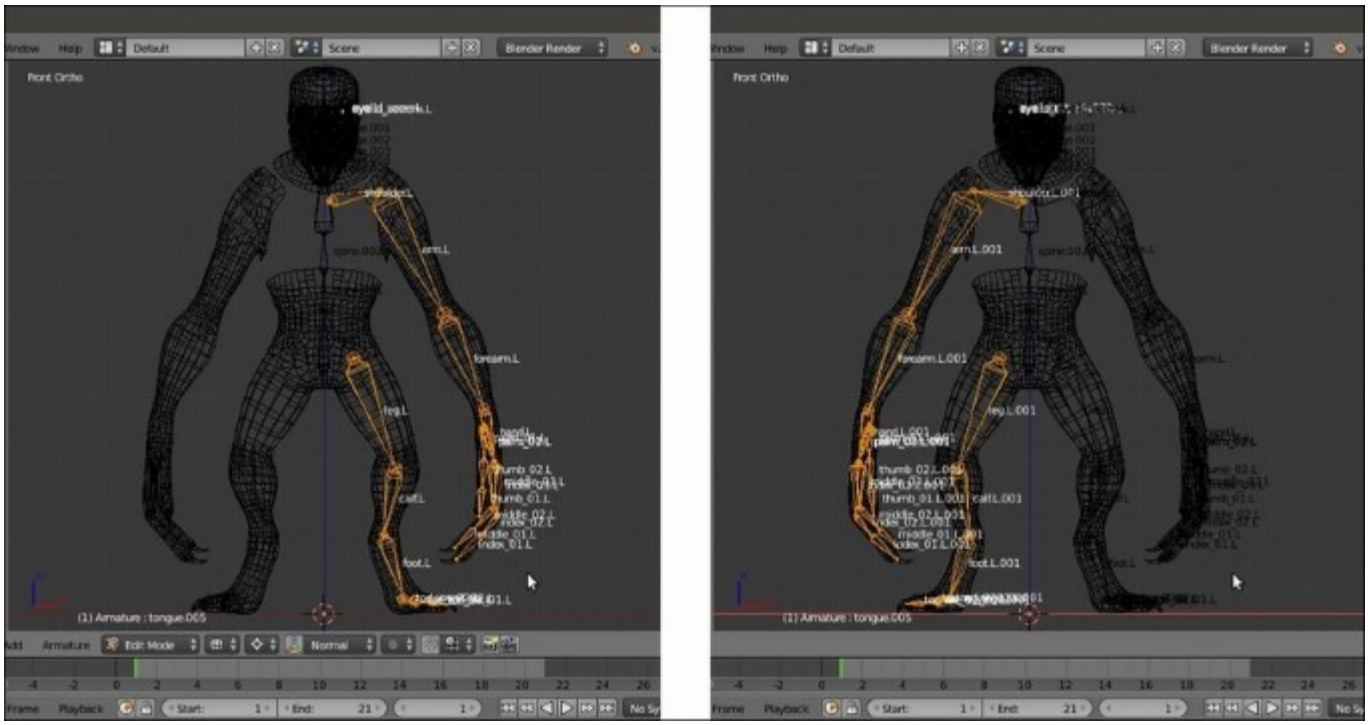
The tongue bones

13. Rename them accordingly, and then use *Shift* to select from the **tongue.001** to **tongue.005** bones and press *Ctrl* + *R*; move the mouse pointer horizontally to *roll* them on their *y* axis by **180°** (hold *Ctrl* to constrain the rolling to intervals of **5** degrees; alternatively, the roll value can also be set by typing it in the **Roll** button in the **Transform** subpanel under the 3D viewport **Properties** sidepanel).

Completing the rig

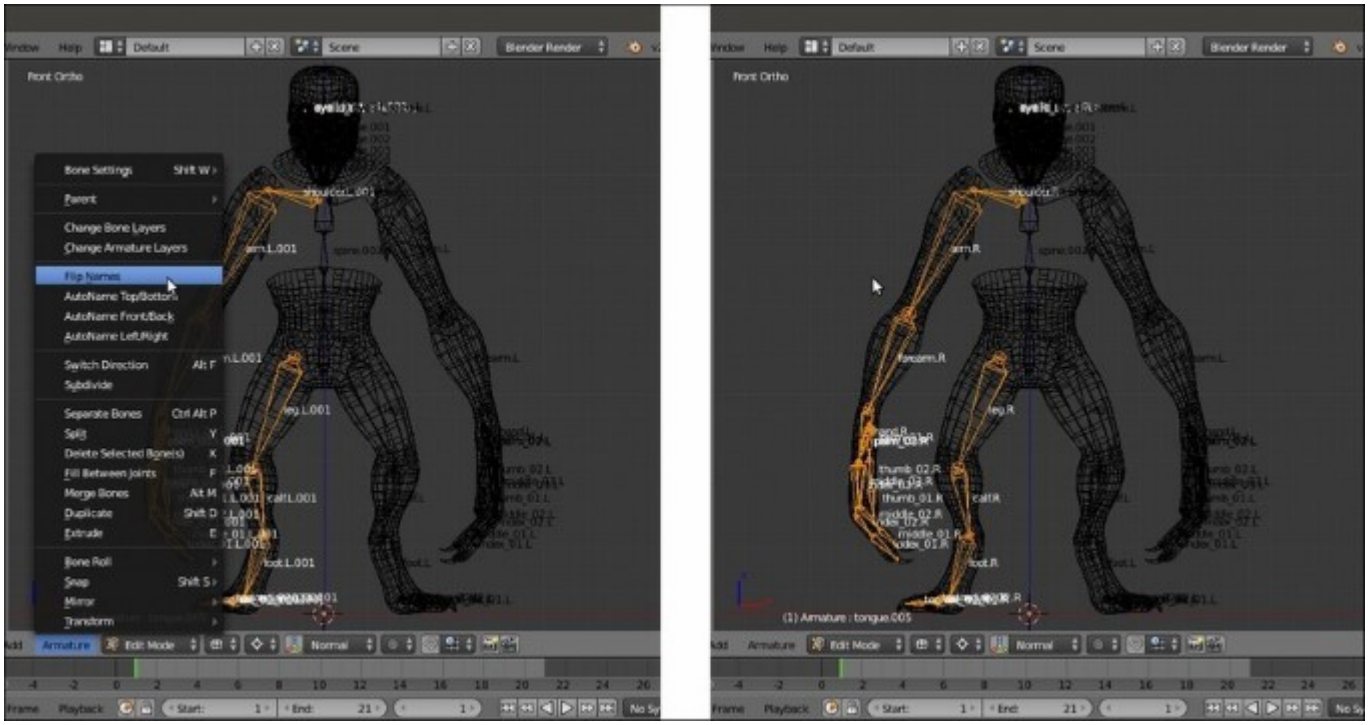
At this point, the basic rig building process is almost done, even if it is only for the left-half part of the mesh:

1. Get out of **Edit Mode** and press *Shift* + *S* | **Cursor to Selected** to place the **3D Cursor** at the median pivot point of the **Armature**.
2. Go back into **Edit Mode**, select only the left-half part bones and *not* the median ones (meaning: leave the **hips**, **spine**, **neck**, **head**, **mouth**, and **tongue** bones unselected), press *Shift* + *D* to duplicate them, and then right-click with the mouse button; press *Ctrl* + *M*, then the *X* key to mirror the duplicated bones on the *x* axis, in order to build the missing right-half part of the rig:



Mirroring the duplicated bones on the x axis

3. With the duplicated bones still selected, go to the **3D window toolbar** and click on the **Armature** item; in the pop-up menu, select the **Flip Names** item to automatically rename them with the correct **.R** suffix:



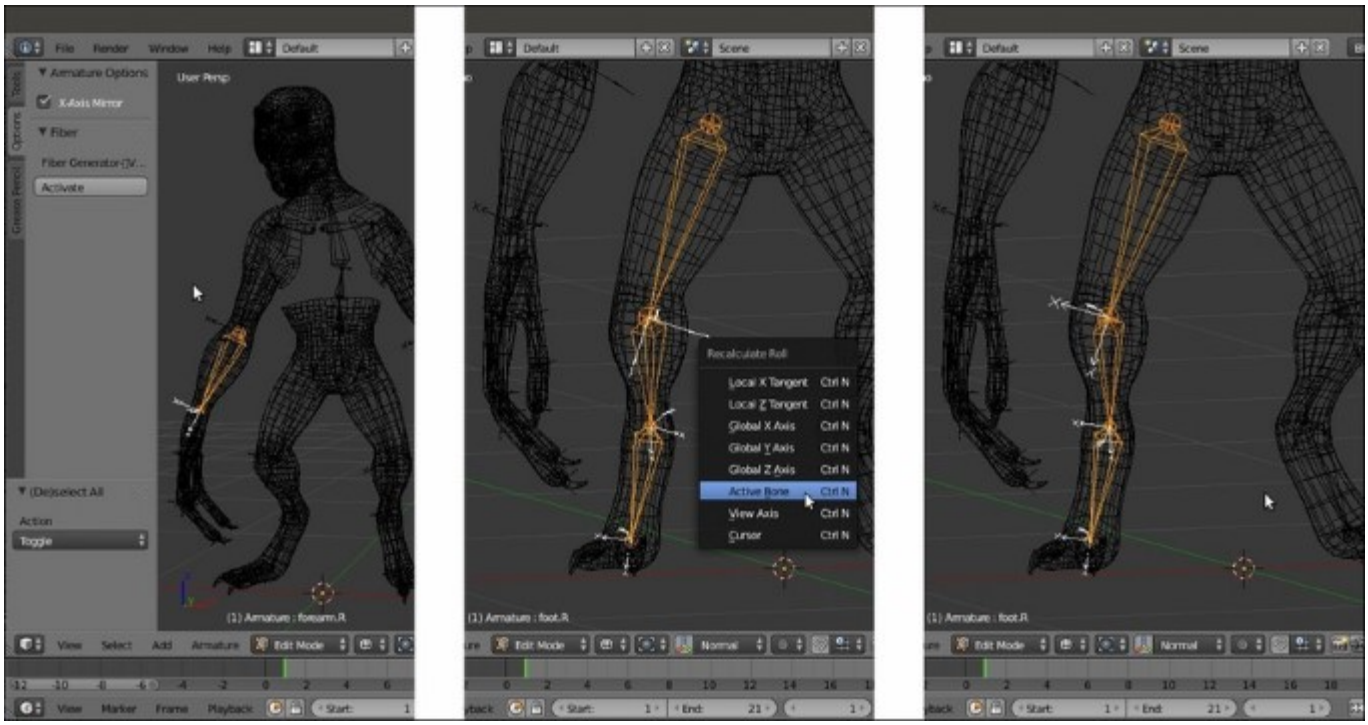
As a very last thing for this recipe, we must verify that the alignment of the bones, especially the last duplicated ones, is correct and, just in case, recalculate the roll rotation, that is, the rotation around the *y* axis of the bone itself.

4. In the **Object Data** window, under the **Display** subpanel, check the **Axes** item to make the bones orientation axes visible (only in **Edit Mode** and **Pose Mode**) in the 3D view.
5. Select all the bones and press *Ctrl* + *N* to recalculate the rolling of all of them; in the **Recalculate Roll** pop-up menu, there are several different options: because basically the *z* axis of the bones must match from the left to the right side of the whole rig, with the **Armature** (and the mesh) oriented along the *y* global axis, as in our **Gidiosaurus** case, the first top item, **Local X Tangent**, can be a good start.

   By the way, it is good practice to not trust this automated procedure alone, because sometimes it can give inconsistent results; so, do the following:
6. After the recalculation, check that the axes of each bone are actually correctly orientated in a consistent way; effectively, there are some bones that didn't get consistently oriented, meaning that their *x* and *z* local axes are oriented differently from the other bones.
7. In this case, select the incorrectly oriented bone, press *Ctrl* + *R*, and move the mouse to change the rolling; press the *Ctrl* key to constrain the rolling to intervals of **5** degrees. Alternatively, select the wrong bones, and then use *Shift* to select one bone that is correctly oriented and press *Ctrl* + *N* | **Active Bone** to copy the rolling from the last selected bone.

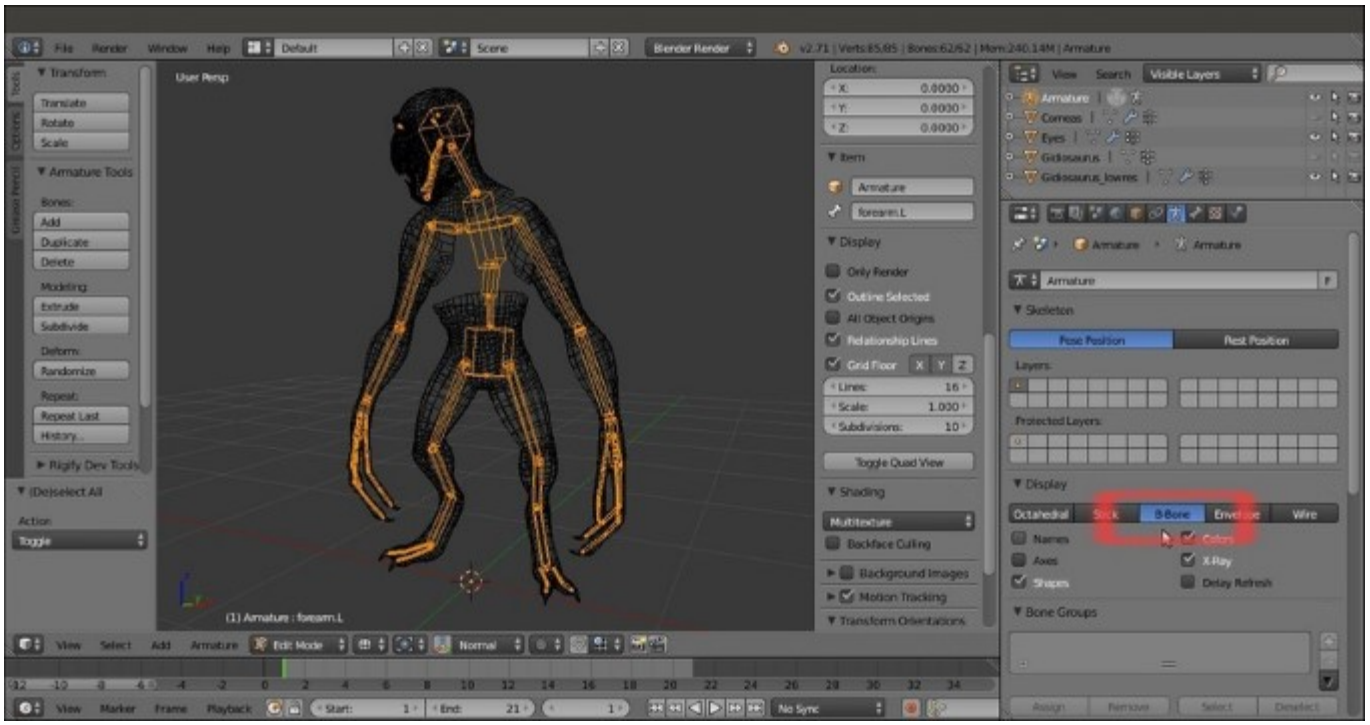
   By enabling the **X-Axis Mirror** item in the **Armature Options** tab under the **Tool Shelf**, you can recalculate only the bones of one side; the other side bones will follow automatically.

   If you want to make sure the bones' orientations are correct and everything is going to work in animation, just go into **Pose Mode** and rotate one bone, for example **leg.L**, and then click on the *Copies the current pose of the selected bones to copy/paste buffer* button (*Ctrl* + *C*), which is the first left one of the last three buttons to the right-hand side of the viewport toolbar; then, select the symmetrical bone, **leg.R**, and click on the last right button to paste the flipped pose (*Ctrl* + *Shift* + *V*); if the **leg.R** bone rotates correctly, then the orientation is OK:



Recalculating the roll of incorrectly oriented bones

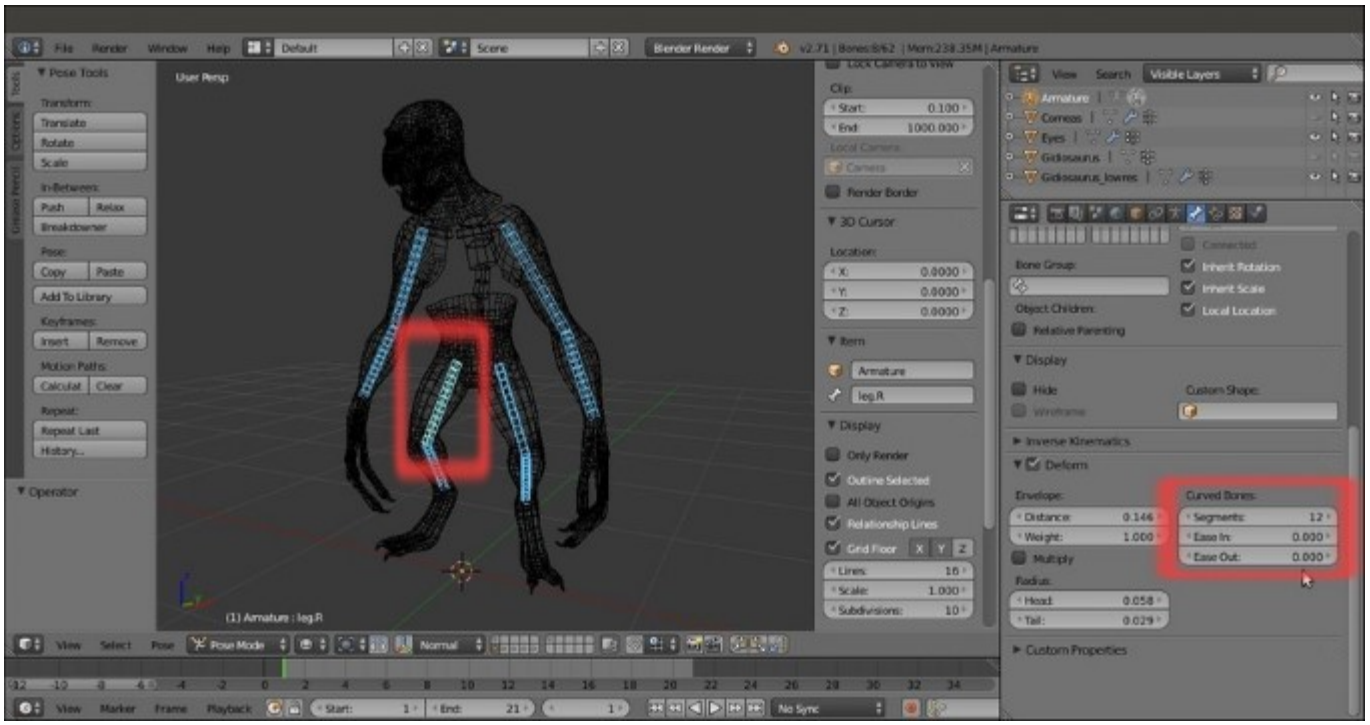
8. Now, go to the **Object Data** window under the main **Properties** panel and under **Display**, switch again the bones visualization from **Octahedral** to **B-bone**; select the bones and by pressing *Ctrl + Alt + S*, scale the **B-bones** smaller or bigger, depending on the visual effect you want to obtain:



The almost completed Armature in B-bones visualization

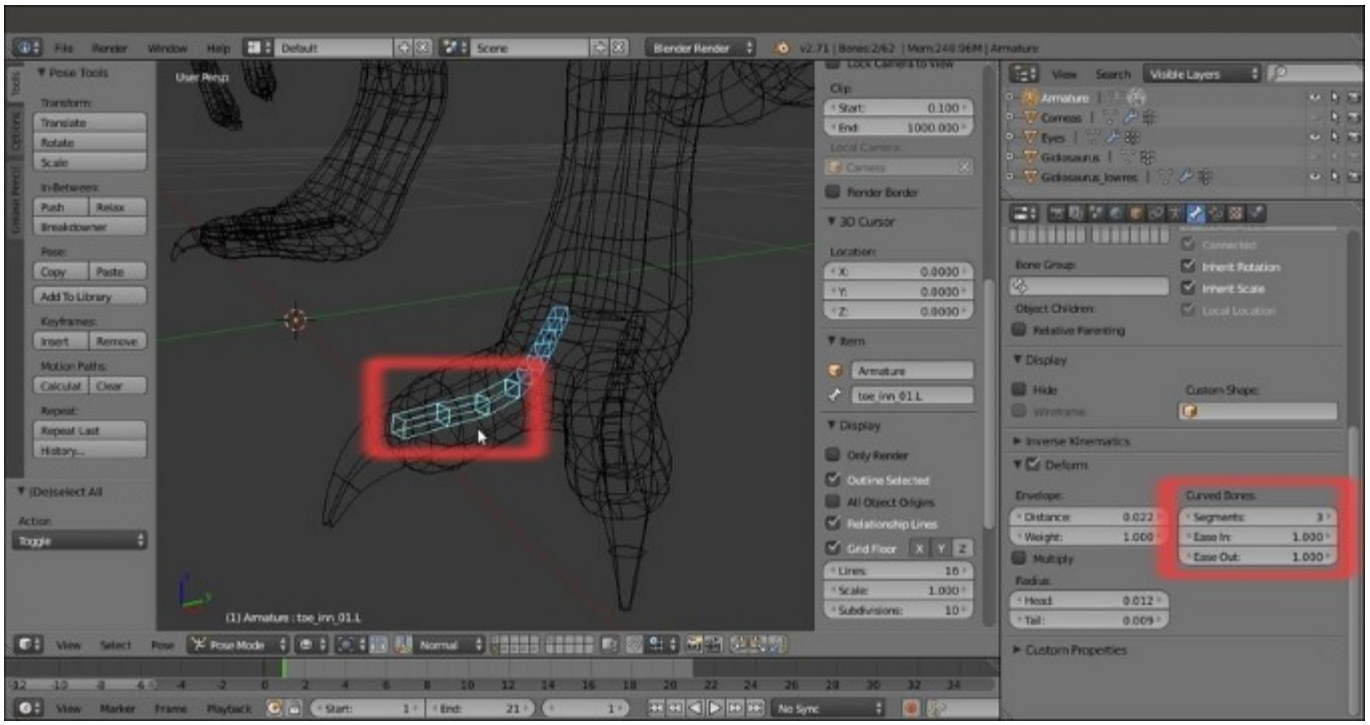
9. Press *Ctrl* + *Tab* to pass directly from **Edit Mode** to **Pose Mode** and select the **forearm.L** bone; in the **Deform** subpanel under the **Properties** panel, set the **Segments** for the **Curved Bones** to **12**, and then set the **Ease In** and **Ease Out** values to **0.000**.
10. Repeat this for the **forearm.R** bone and also for the **calf.L** and **calf.R** bones; repeat also for the **arm.L**, **arm.R**, **leg.L**, and **leg.R** bones.

    In the following screenshot, all the eight **B-bones** have been selected to make them more visible. By the way, the highlighted **leg.R** bone is the active one and shows the **Curved Bones** setting in the highlighted **Deform** subpanel to the right-hand side of the screen.



The Segments setting for the leg bone

11. Select the **toe_inn_02.L** bone and in the **Deform** subpanel under the **Properties** panel, set the **Segments** to **6** and leave the **Ease In** and **Ease Out** values to **1.000**.
12. Repeat this for the **toe_ext_02.L** bone; then, do the same also to the **toe_inn_02.R** and **toe_ext_02.R** bones.
13. Select the **toe_inn_01.L** bone and set the **Segments** to **3**; leave the **Ease In** and **Ease Out** values to **1.000**.
14. Repeat for the **toe_ext_01.L** bone; then, do the same also to the **toe_inn_01.R** and **toe_ext_01.R** bones:



The Segments setting for the toes bones

15. Save the file.

How it works…

Although it's often a really time consuming task, the handmade rigging is quite self-explicative; it is, however, better to explain some of the concepts behind this.

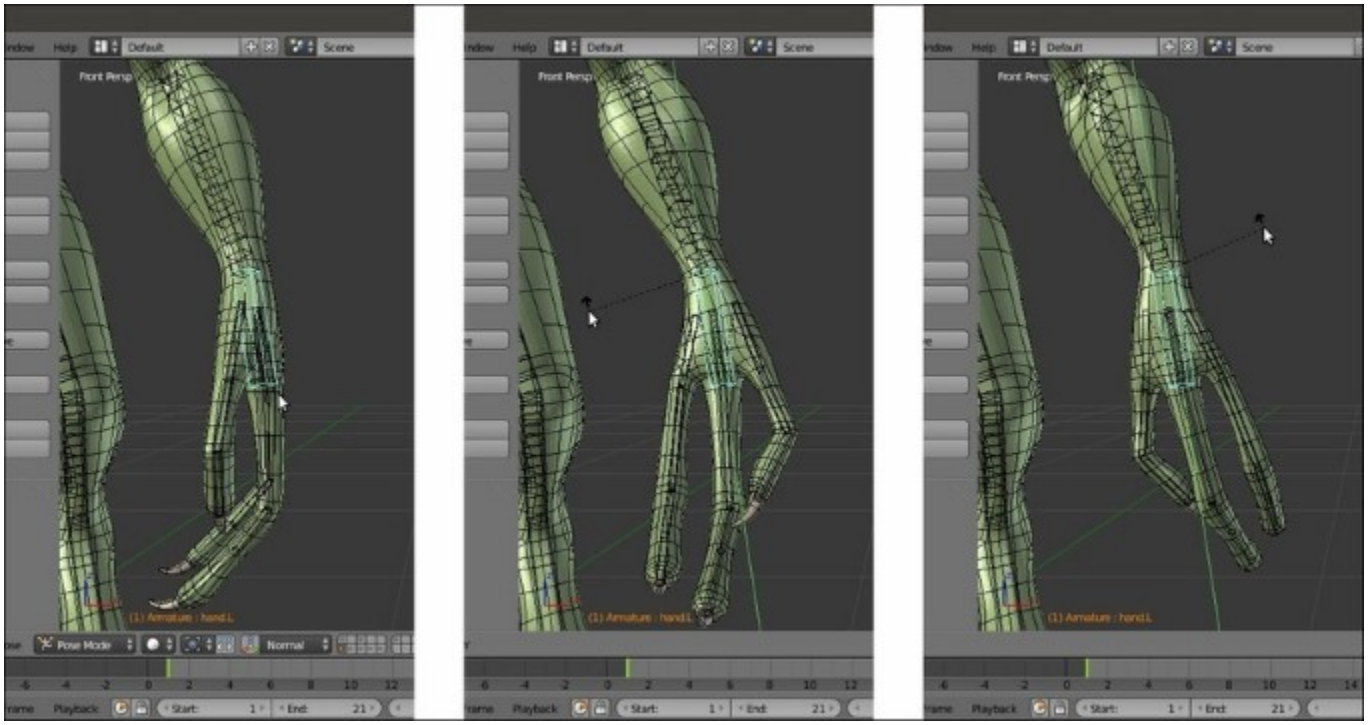
The proper renaming of the bones is important, considering that each deforming bone will affect a vertex group sharing the same name on the mesh; although in some cases, as for the **tongue** bones, the bone naming process can be automated in some way, usually it is better to spend time in giving meaningful names to each bone, in order to avoid mistakes in the following skinning process.

It's also very important to build the hierarchy of the bones so that a bone at a higher level can lead all of the children bones, as it would be in a real skeleton (that is, for example, the **hand** bone leads all the **fingers** bones, the **forearm** bone leads the **hand** bone, and so on).

Parenting a bone and then obtaining the others by extruding and/or duplicating simplifies the work because an extruded bone is automatically parented to the bone it has been extruded from, and a duplicated bone obviously inherits the parenting of the original one; in the case of the **tongue.001** bone, extruding the others has given us a chain with bones automatically parented and named as **tongue.002**, **tongue.003**, **tongue.004**, and **tongue.005**.

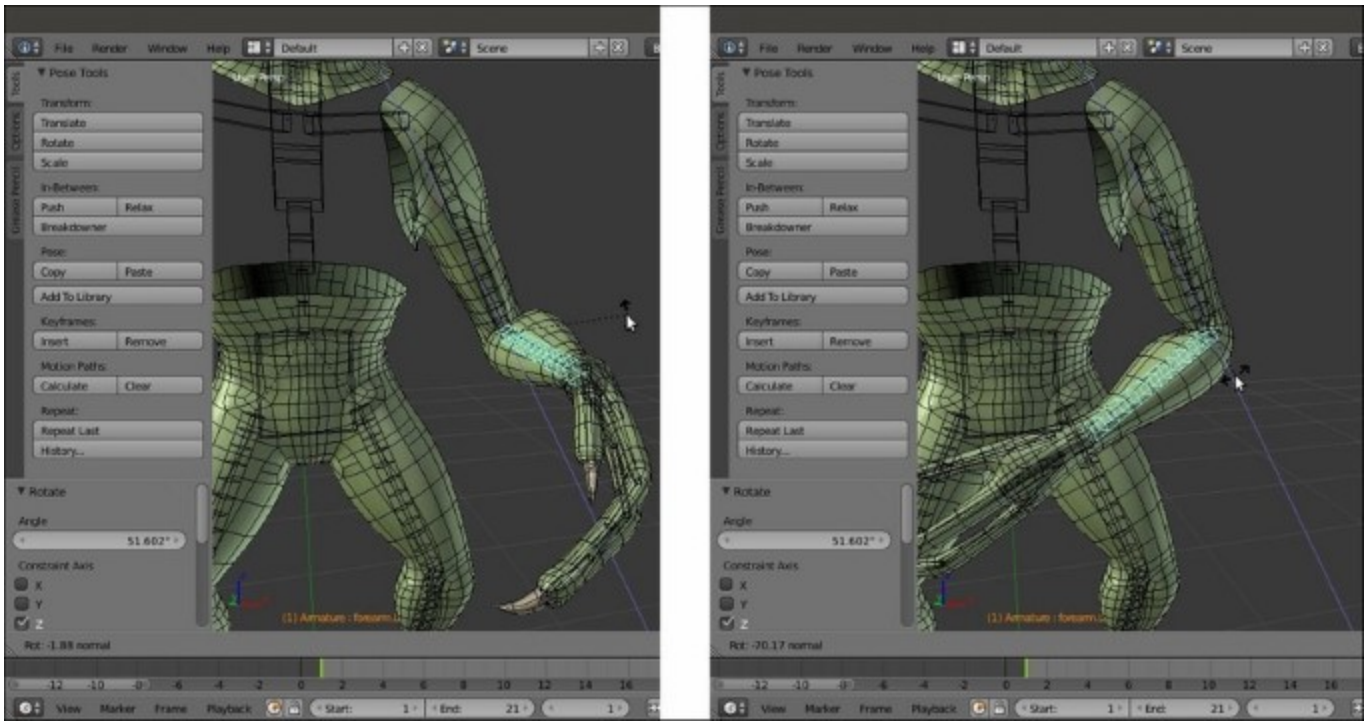
B-bones are both a visualization mode for the bones and a way of working; **B-bones**, in fact, can work inside a chain as splines, which means that the bones are curved according to the number of **Segments** and the values of the **Ease In** and **Ease Out** items. For the bones of the **arms** and **legs**, we have set the **Ease In** and **Ease Out** values to **0.000** (default is **1.000**; maximum is **2.000**), in order to have the B-bones rotating only on their **y** axis but remaining straight along their length, and hence, mimic the twisting by not only the rotation (*pronation* and *supination* of the lower arm) of both the *Ulna-Radius* and *Tibia-Fibula* articulation complexes, but also the (limited) rotation of *Femur* and *Humerus*.

In some way, **B-bones** can work as a kind of simulation for a very basic muscle system; in the following screenshot, you can see their effect on the skinned mesh for the **forearm** by rotating the **hand.L** bone on the local *y* axis (to enhance the visibility of the mesh surface's modifications, the *wireframe over solid drawing* item has been enabled in the **Display** subpanel under the **Object** window):



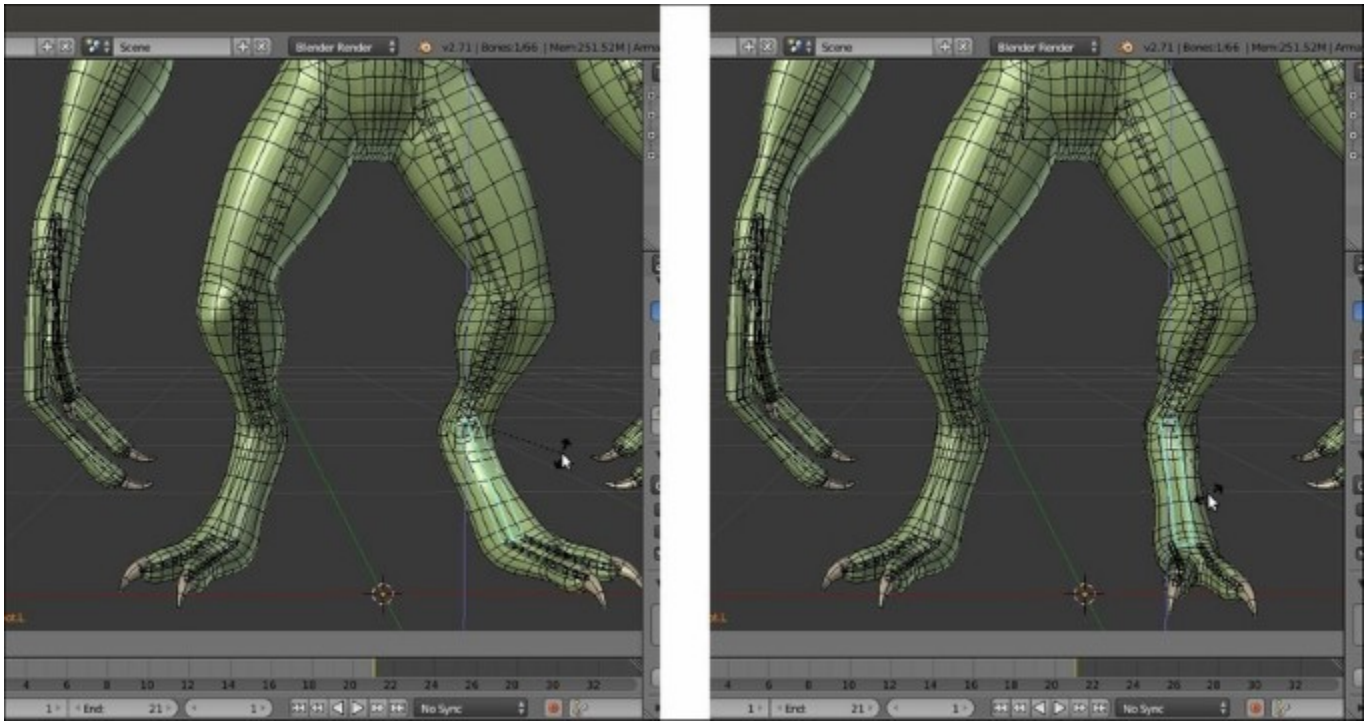
The effect of the rotation of the hand bone on the forearm B-bone and skinned mesh

Here is the effect of the rotation of the **forearm.L** bone on the **Gidiosaurus** high **arm**:



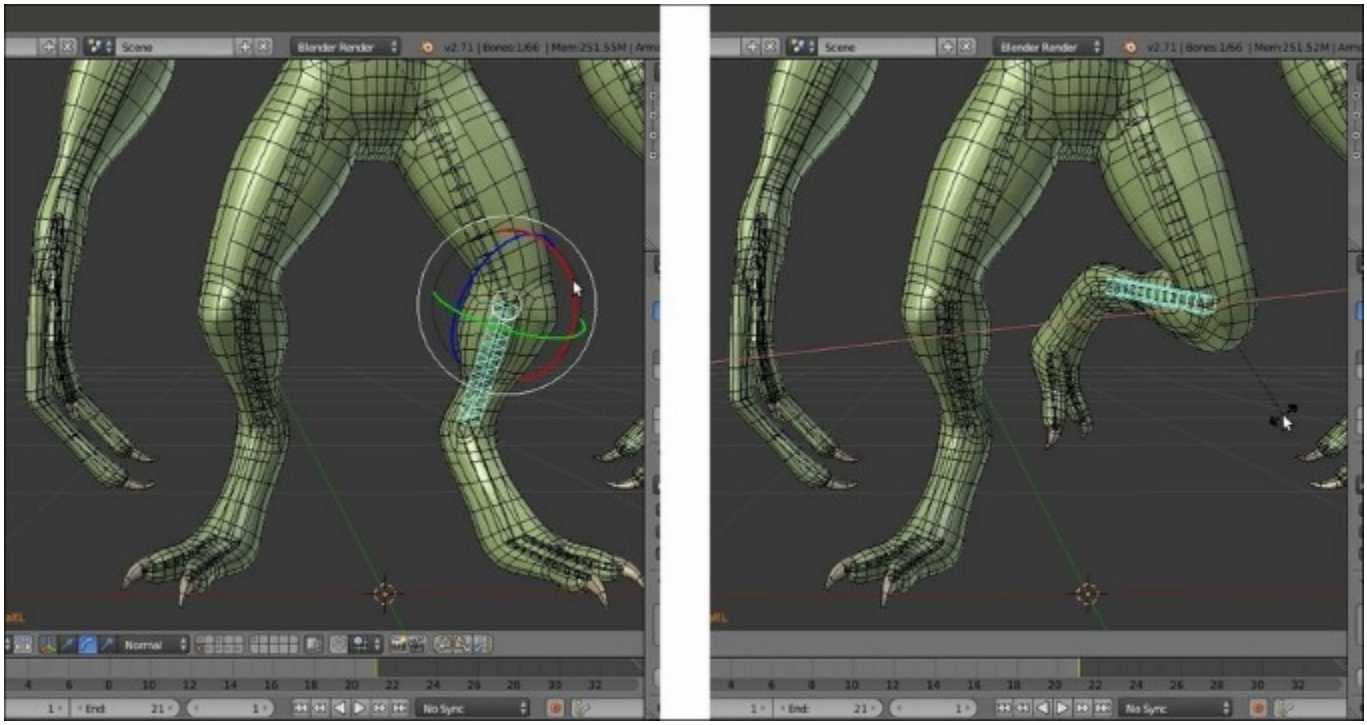
The effect of the rotation of the forearm bone on the upper arm b-bone and skinned mesh

The effect acts on the **shin** as well, by rotating the **foot.L** bone on the global *z* axis:



The same effect obtained on the calf b-bone by rotating the foot on its local y axis

Also, the same effect acts on the **thigh** by rotating the **calf.L** bone:



The same effect obtained on the leg b-bone by rotating the calf

Note that the **Gidiosaurus** is a **digitigrade biped humanoid**: the bones that, from our plantigrade point of view, look like the **foot** are actually the **toes**, while the almost vertical structure that we would call an **ankle** is the real **foot** (this is a very common condition among the majority of the terrestrial animals, both still alive and extinct).

Perfecting the Armature to also function as a rig for the Armor

So, in the previous recipe, we have built the body deforming **Armature** for the **Gidiosaurus** character.

However, the **Gidiosaurus** is an (almost) evolved and a civilized creature, and being also a warrior, it wears a metallic **Armor**; this armor will need to be later parented to the rig as well in order to be animated.

Some of the bones that we have already created will be perfect to skin the **Armor** object too, by assigning the right vertex group to the right mesh part (for example, the **head** vertex group for the **Helm** or the **chest** vertex group for the **Breastplate**). However, because the **Armor** is made also by different parts that cannot be simply driven by the already existing bones (for example, the **belts**, **Vambraces**, and especially **Groinguard**), some modification and/or addition to the rig must be done anyway.

Getting ready

Start from the previously saved `Gidiosaurus_rig_from_scratch_01.blend` file:

1. Enable the **13th** scene layer to show the **Armor** object.
2. Select it and go to the **Object Modifiers** window under the main **Properties** panel. Expand the **Subdivision Surface** modifier tab and click on the *Display modifier in viewport* button, the one with the eye icon, to disable it.
3. Go to the **Outliner** and click on the arrow icon to the side of the **Armor** item to make it unselectable.
4. Click on the arrow icon to the side of the **Gidiosaurus_lowres** item to make it unselectable as well.
5. Save the file as `Gidiosaurus_rig_from_scratch_02.blend`.

How to do it…

Let's start by adding bones dedicated to the **Armor**:

1. Go into **Edit Mode** and select the **forearm.L** bone; use *Shift* + *D* to duplicate it and rename it **vanbrace.L**. Press *M* and in the **Change Bone Layers** pop-up, click on the **2nd** button to move the duplicated bone to that bone layer.
2. Do the same for the **forearm.R** bone (**vanbrace.R**) and for the **calf.L** (**greave.L**) and **calf.R** bones (**greave.R**).
3. Now, go to the **Object Data** window and click on the **2nd** button under the **Layers** item in the **Skeleton** subpanel, in order to show only the four duplicated bones in the 3D viewport; press *Tab* to get out of **Edit Mode**.
4. Select the **vanbrace.L** bone and go to the **Bone** window under the **Deform** subpanel; under the **Curved Bones** item, set back the **Segments** and **Ease In** and **Ease Out** values to default, that is, **1**, **1.000** and **1.000**.
5. Go back into **Edit Mode** and click on the **Connected** item under the **Relations** subpanel.

6. Get out of **Edit Mode** and go to the **Bone Constraints** window under the main **Properties** panel (not to be confused with the **Object Constraints** window); click on the **Add Bone Constraint** button and select a **Copy Rotation** constraint from the pop-up menu (the bone turns light green, in order to show that it has a constraint assigned now).
7. In the **Target** field, select **Armature**; in the **Bone** field, select the **forearm.L** item; in the **Space** fields, select **Pose Space** for both.

   Alternatively, for steps 6 and 7, select the **forearm.L** bone and then use *Shift* to select the **vanbrace.L** bone. Hence, press *Shift* + *Ctrl* + *C* to call the **Add Constraint (with Targets)** pop-up menu and select the **Copy Rotation** item. This will automatically add the **Copy Rotation** constraint to the **vanbrace.L** bone, with the first select bone (**forearm.L**) as a target; the other setting must be enabled and/or tweaked in the constraint subpanel instead.
8. Click again on the **Add Bone Constraint** button and this time, select an **Inverse Kinematics** constraint (the bone turns yellow, in order to show that an **IK solver** has been assigned). In the **Target** field, select the **Armature** item, in the **Bone** field, select the **hand.L** bone, and set the **Chain Length** to **1**; deselect **Stretch** and select **Rotation**, lowering the weight to the minimum (that is **0.010**):

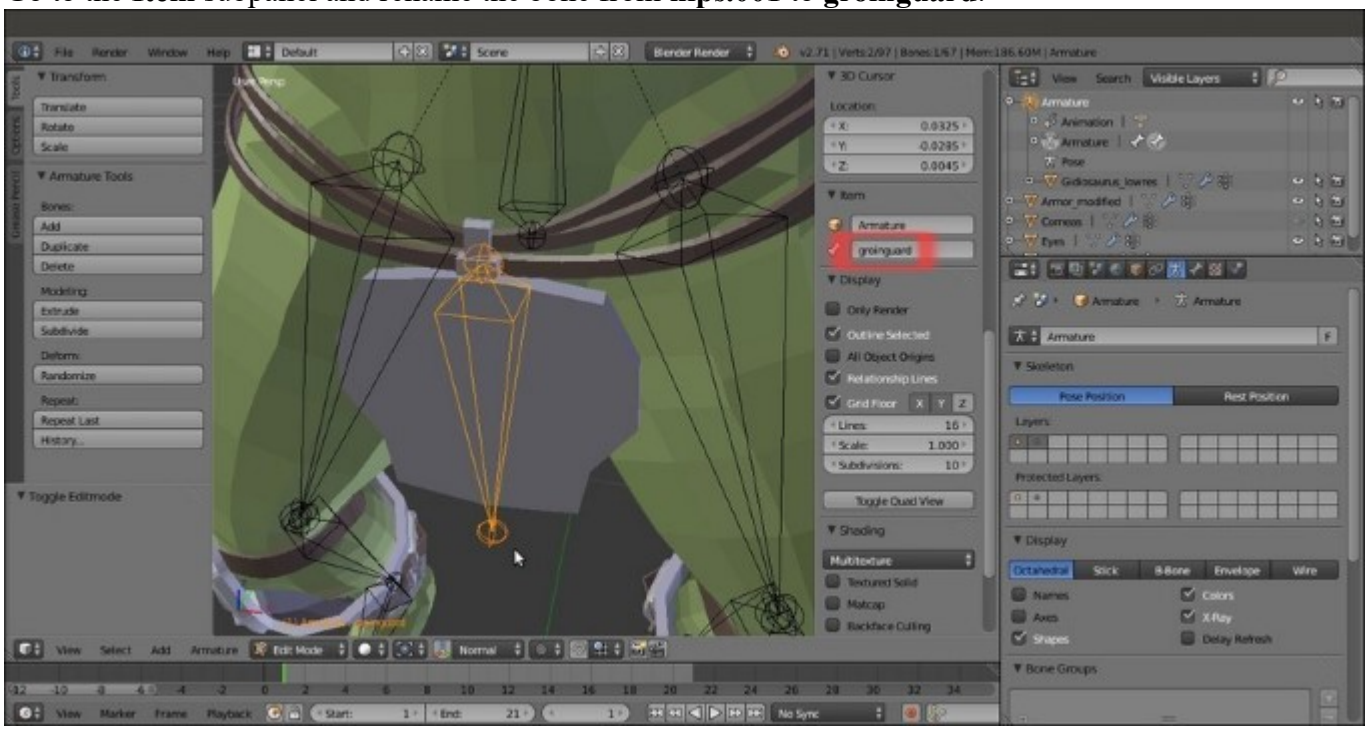
The constraints assigned to the forearm.L b-bone

9. Repeat the steps from 4 to 8 for the other three duplicated bones (obviously, setting the appropriate bones as targets for each pair of constraints; the target bone for the **IK** constraint assigned to the **greave** bones is the respective foot bone).

   The rig can now drive the **vambraces** and **greaves**; let's see the **knee guards** and **Groinguard**.
10. First, switch the **Armature** visualization back to **Octahedral**, then go into **Edit Mode**, select the **hips** bone and use *Shift* + *D* to duplicate it; in the **Side** view, rotate the duplicate **170**

degrees, then move it on the **Groinguard** part of the armor, in order to have the **Head** of the bone placed to the joint of the **plate** with the **ties**; select the **Tail** of the **groinguard** bone and scale it smaller to fit the part.

11. To position the bone more precisely, go to the **Transform** subpanel under the **Properties** 3D view sidepanel and set the following values for the **Head** (of the bone): **X** = **0.001**, **Y** = **0.020**, and **Z** = **1.147**; for the **Tail** set the following values: **X** = **0.001**, **Y** = **0.022**, and **Z** = **0.873**.
12. Go to the **Item** subpanel and rename the bone from **hips.001** to **groinguard**:

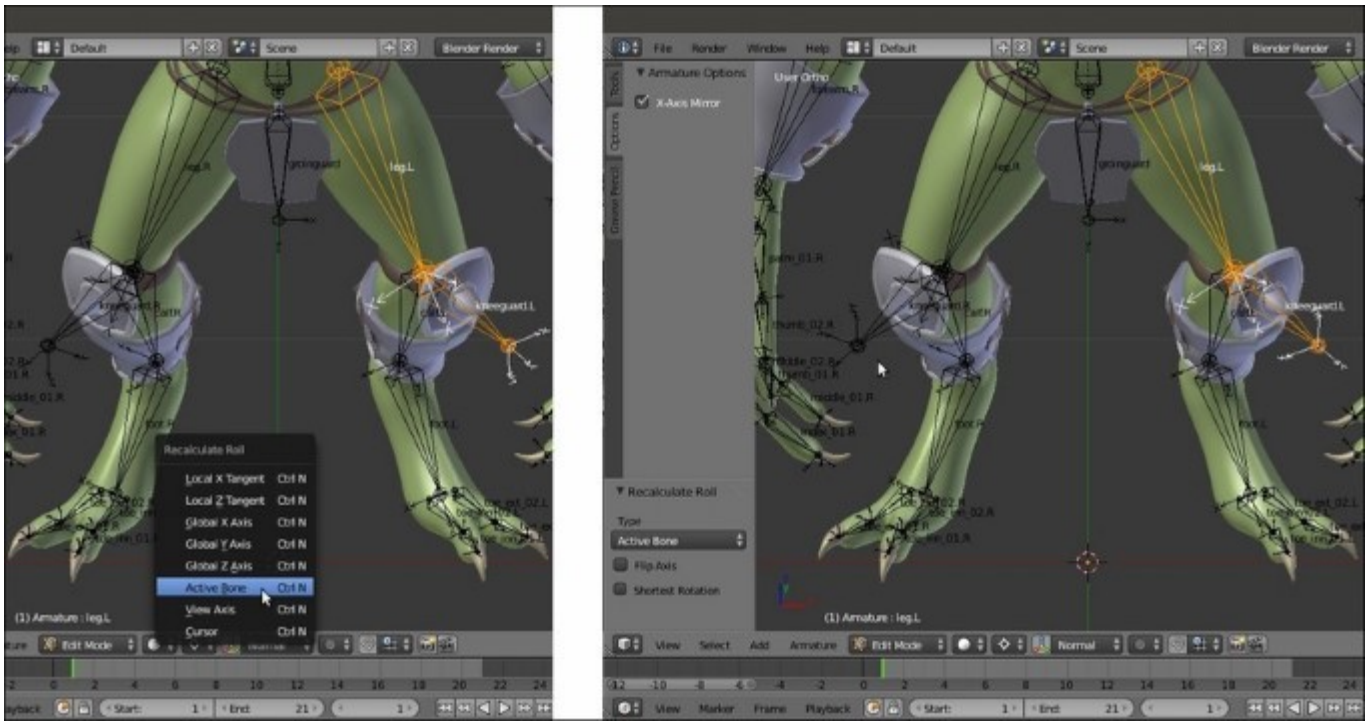


The groinguard bone

13. Go to the **Bone** window in the main **Properties** panel, and under the **Relations** subpanel, click on the **Parent** empty slot to select the **hips** item.
14. Now, select the joint of the **leg.L** bone with the **calf.L** bone and press *Shift* + *S* | **Cursor to Selected**; press *Shift* + *A* to add a new bone and rescale it smaller.
15. Select the whole new bone and use *Shift* to select the **calf.L** bone. Then, go in the 3D view toolbar and click on the **Armature** item; go to **Transform** | **Align Bones** (or else, press the *Ctrl* + *Alt* + *A* keys) to align the new bone as the **calf.L** one.
16. Enable the widget (*Ctrl* + spacebar), set the **Transform Orientation** to **Normal**, and the rotation pivot on the **3D Cursor**. Then, rotate the new bone **110** degrees on the normal *x* axis (the red wheel of the widget, or else *R* | *X* | *X* | **110** | *Enter*):

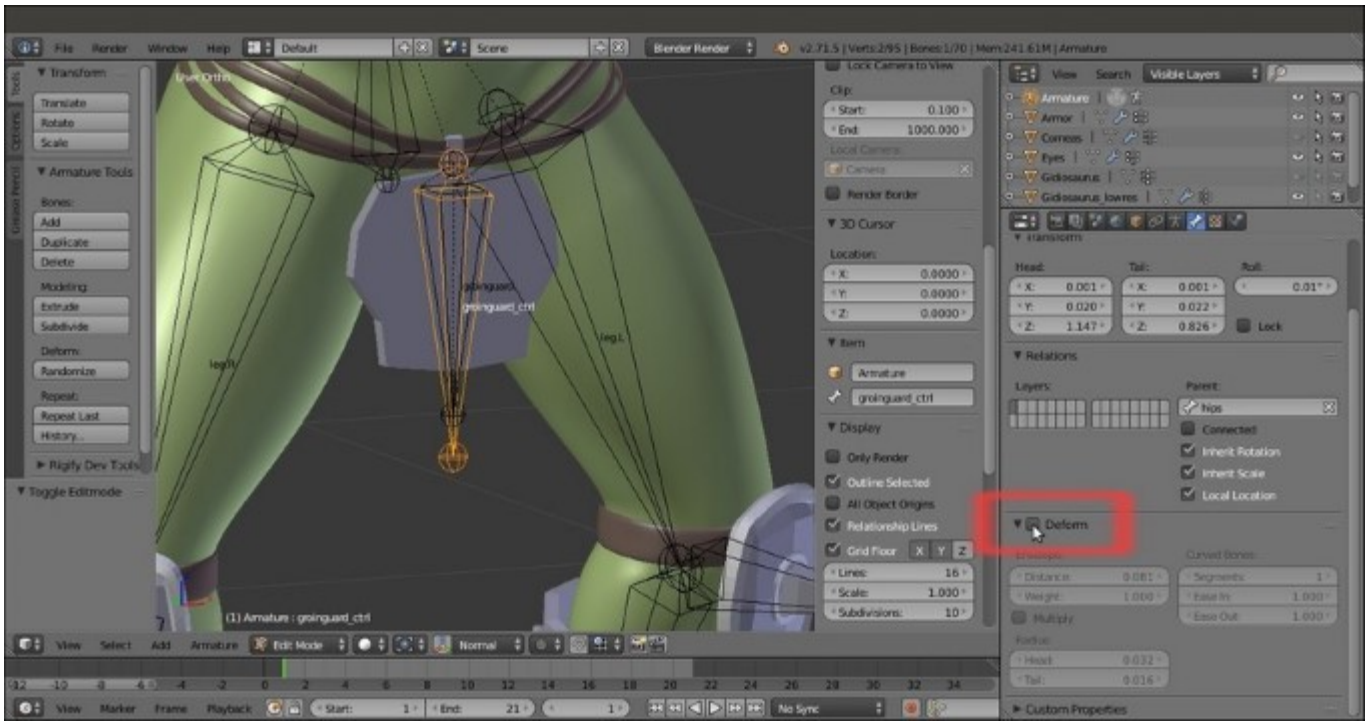
Aligning the new bone

17. Go into **Object Mode** and press *Shift* + *S* | **Cursor to Selected** to place the **3D Cursor** at the median pivot point of the **Armature**; go back into **Edit Mode**, press *Shift* + *D* to duplicate the new bone, then *Ctrl* + *M* | *X* to mirror it on the other side.
18. Rename the new bones as **kneeguard.L** and **kneeguard.R**; enable the axis visibility and recalculate the roll by the *Ctrl* + *N* | **Active Bone** tool:



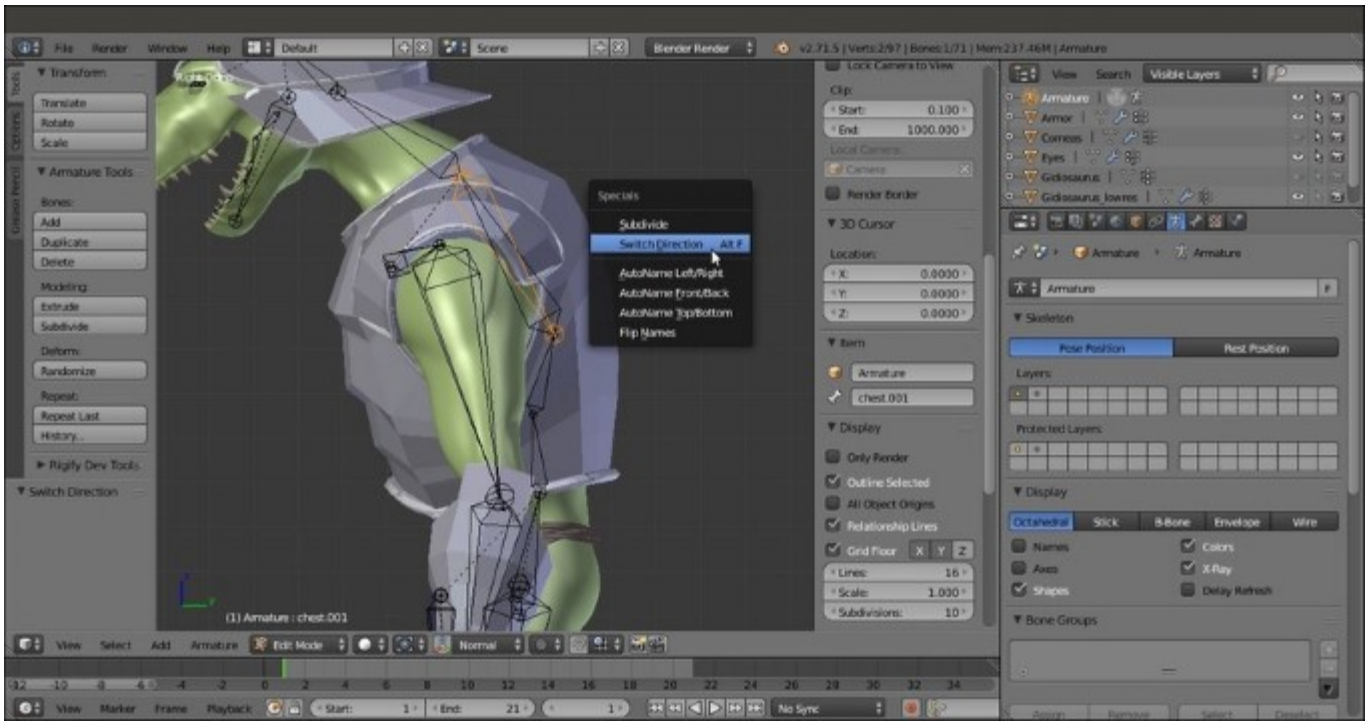
Recalculating the roll angle of the kneeguard bones

19. Parent the **kneeguard.L** bone to the **leg.L** bone and the **kneeguard.R** bone to the **leg.R** one (not connected).
20. Select the **groinguard** bone and use *Shift + D* to duplicate it, and then scale the duplicated bone a little bit bigger and rename it as **groinguard_ctrl**; uncheck the box of the **Deform** subpanel under the **Bone** window:



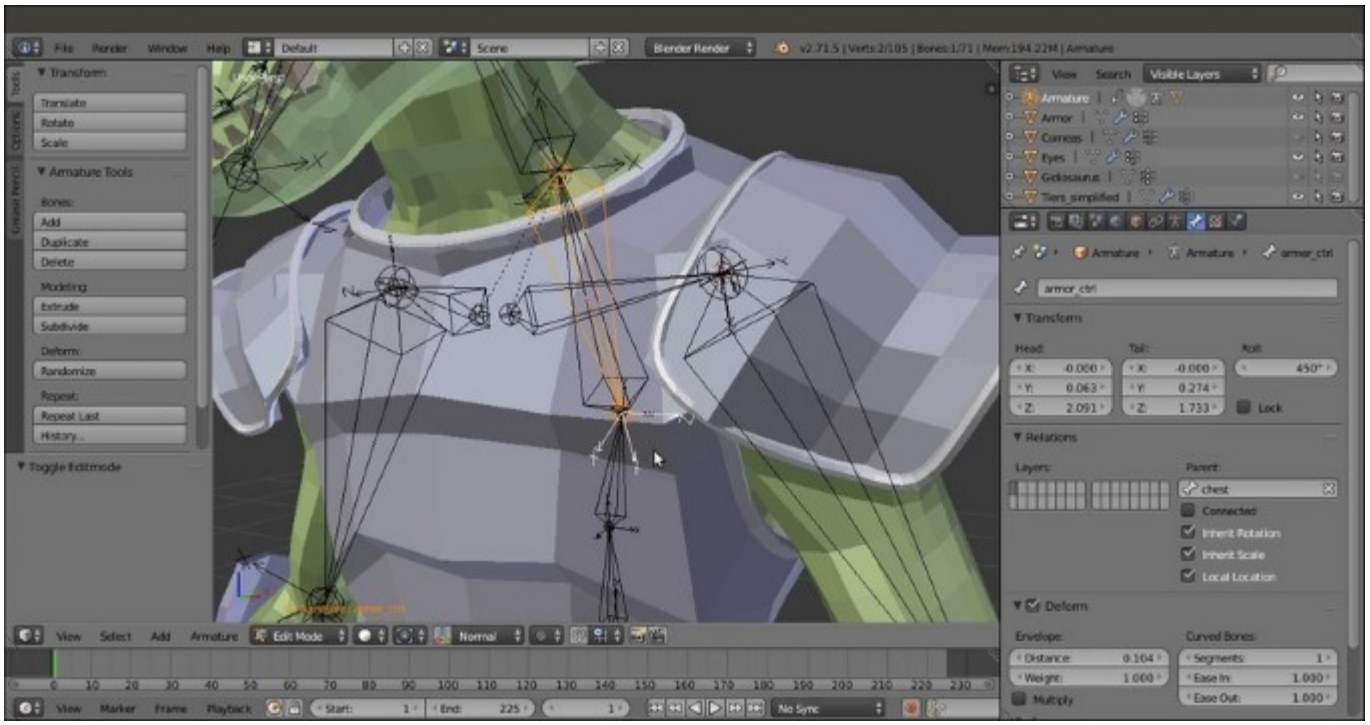
Creating a control bone for the groinguard bone

21. Select the **groinguard** bone, go to the **Relations** subpanel, and click in the **Parent** field to select the **groinguard_ctrl** bone.
22. Get out of **Edit Mode** and in **Pose Mode**, select the **groinguard_ctrl** bone.
23. Go to the **Bone Constraints** window under the main **Properties** panel; click on the **Add Bone Constraint** button and select a **Locked Track** constraint from the pop-up menu.
24. In the **Target** field, select the **Armature** item; in the **Bone** field, select the **kneeguard.L** item. Set the **Head/Tail** value to **0.500**: **To** (*Axis that points to the target object*) = **-X** and **Lock** (*Axis that points upward*) = **Y**. In the **Constraint Name** field, rename it as **Locked Track.L**.
25. Add a new **Locked Track** constraint and repeat everything as in the previous one, except in the **Bone** field, select the **kneeguard.R** item; rename it as **Locked Track.R**.
26. Add a **Damped Track** constraint: **Target** = **Armature**, **Bone** = **kneeguard.L**, **Head/Tail** = **0.728**, **To** = **Y**, and **Influence** = **0.263**. Rename it as **Damped Track.L**.
27. Add a new **Damped Track** constraint and repeat everything as in the previous one, except again in the **Bone** field, select the **kneeguard.R** item; rename it as **Damped Track.R**.
28. Just to be sure, save the file!
29. Go back into **Edit Mode** and in the **Side** view, select the **chest** bone and use *Shift* + *D* to duplicate it. Press *W* to call the **Specials** pop-up menu and select the **Switch_Direction** item, or else press *Alt* + *F* directly:



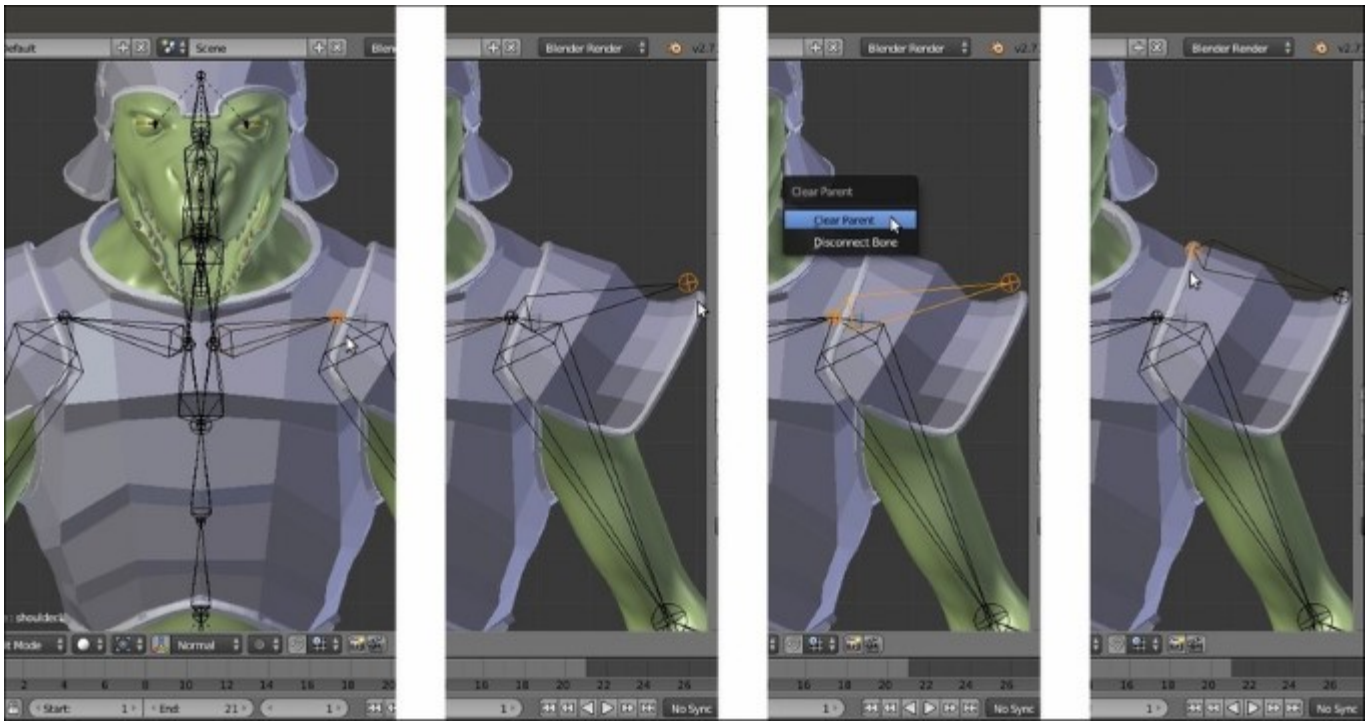
The Specials pop-up menu for the bones

30. Go to the **Bone** window and click on the **Parent** slot under the **Relations** subpanel to select the **chest** item (not connected); then, go to the Deform subpanel and set the **Segments** under **Curved Bones** to **1**. Rename the new bone as **armor_ctrl**.
31. Press *Ctrl* + *R* to roll the **armor_ctrl** bone, in order to be sure that its local *x* axis is pointing towards the front of the model; this is important to make the **Transformation** constraints, which we'll add later, work properly:



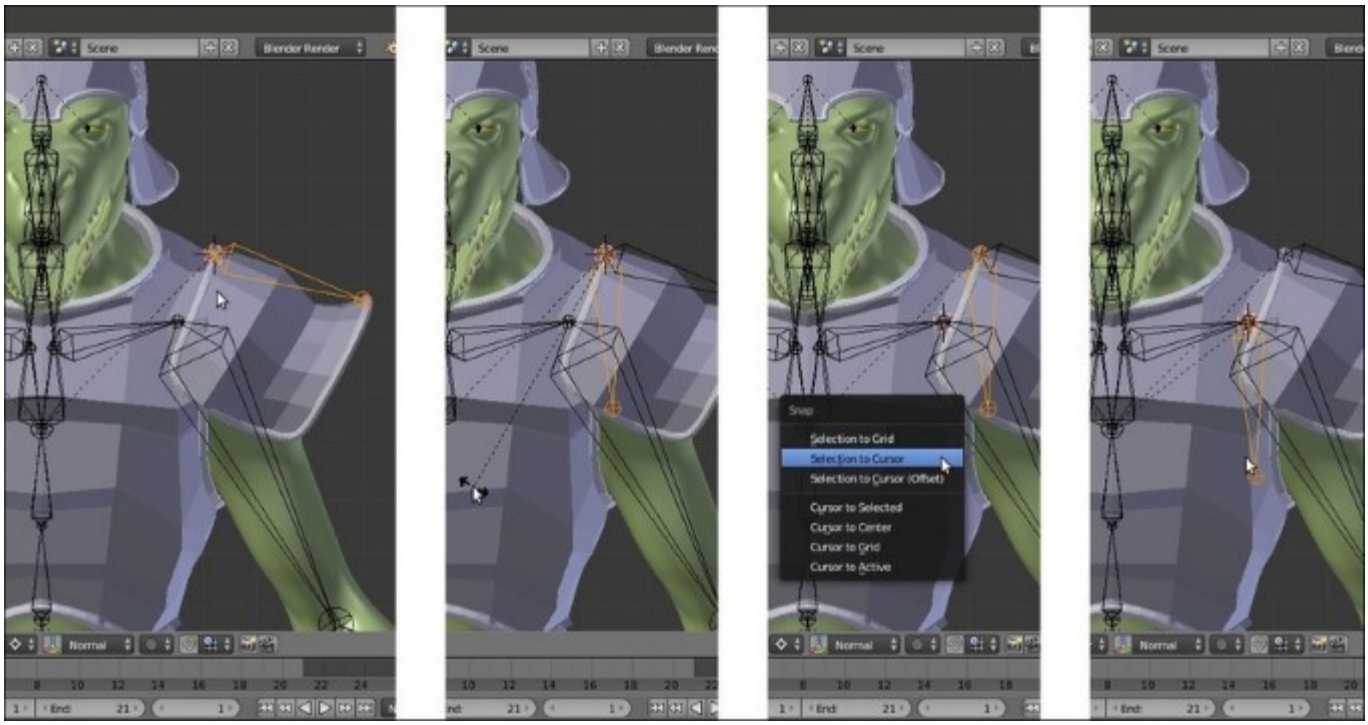
The armor control bone

32. Go in the **Front** view. Note that the **X-Axis Mirror** item in the **Armature Options** panel under the **Tool Shelf** is still enabled; select the **Tail** of the **shoulder.L** bone and extrude a new bone going towards the external edge of the armor **spaulder**. Then, select the extruded bone, press *Alt* + *P* | **Clear Parent**, and move its **Head** to be positioned above the *joint* of the **spaulder** with the **chest** plate.



Creating the bone for the spaulder

33. Rename the extruded bone and the corresponding mirrored one as **spaulder.L** and **spaulder.R**; parent them to the **armor_ctrl** bone (enable the **Keep Offset** item).
34. Use *Shift* to select the **spaulder.L** and **arm.L** bones and press *Ctrl* + *N* | **Active Bone**; do the same with the **spaulder.R** and **arm.R** bones.
35. Now, put the **3D Cursor** at the **spaulder.L** bone's **Head** location, and then set the **Pivot Point** to the **3D Cursor** in the 3D window toolbar. Use *Shift* + *D* to duplicate the **spaulder.L** bone and rotate the duplicate **70** degrees (in the **Front** view, *R* | **70** | *Enter*).
36. Place the **3D Cursor** at the **shoulder.L** bone's **Tail** location, select the duplicated bone, and press *Shift* + *S* | **Selected to Cursor**. Rename the duplicated bone and the mirrored one as **rotarmor.L** and **rotarmor.R**. Go to the **Relations** subpanel and set the **rotarmor.L** bone as the child of the **arm.L** bone and the **rotarmor.R** bone as the child of the **arm.R** bone. Disable the **Deform** item for both of them:



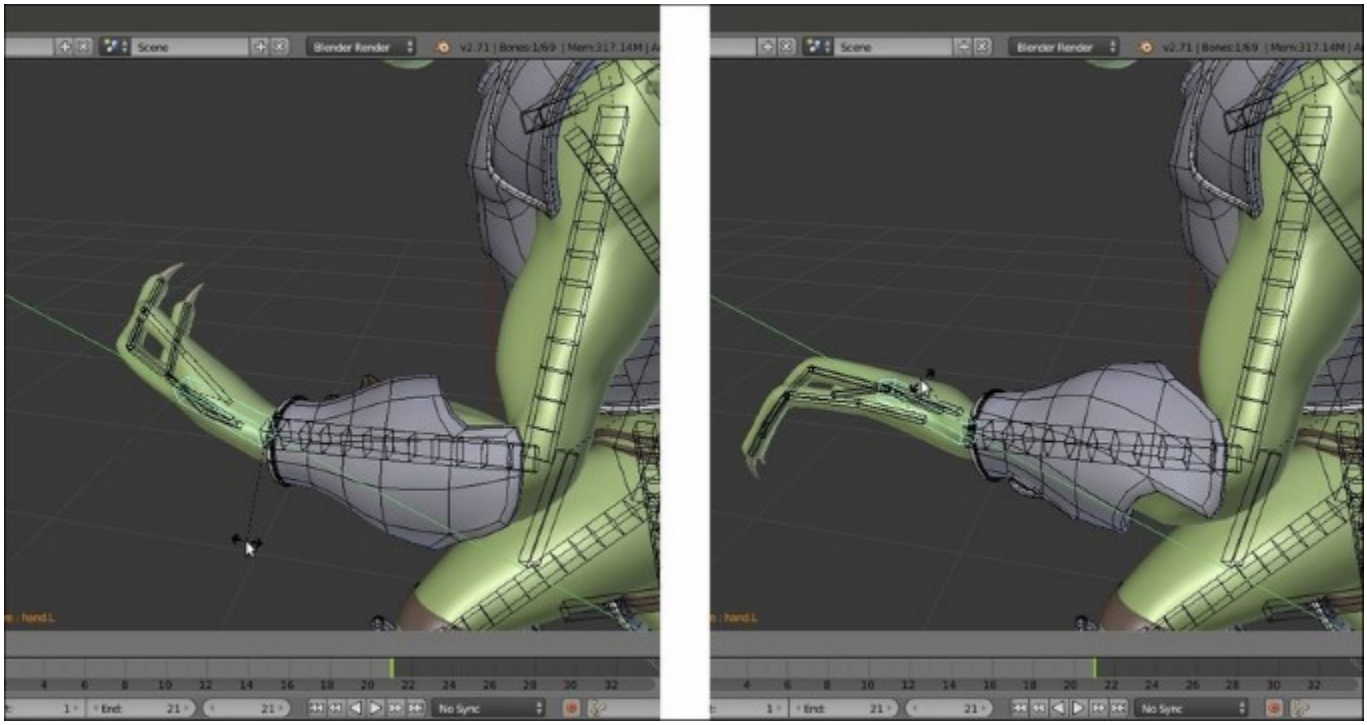
Using the 3D Cursor and the Snap menu to exactly place the bones

37. Go into **Pose Mode**. Select the **spaulder.L** bone and in the **Bone Constraints** window, assign a **Copy Rotation** constraint: **Target** = **Armature**, **Bone** = **arm.L**, **Space** = **Pose Space** to **Pose Space**, and **Influence** = **0.200**.
38. Select the **spaulder.R** bone and repeat with the **Bone** = **arm.R** target.
39. Now, select the **armor_ctrl** bone and assign a **Transformation** constraint. Set **Target** = **Armature**, **Bone** = **rotarmor.L**, **Source** = **Rot**, and **Z Max** = 20°; **Source To Destination Mapping** = switch **X** with **Z**; **Destination** = **Rot**, **X Max** = 4°, and **Space** = **Pose Space** to **Pose Space**. Rename the constraint as **Transformation_rot.L** and collapse the panel.
40. Assign a second **Transformation** constraint; set everything as in the previous one, except for the target **Bone** = **rotarmor.R**, **Source** = **Rot**, **Z Min** = **-20°**, and **Destination X Min** = **-4°**. Rename the constraint as **Transformation_rot.R** and collapse it.
41. Assign a third **Transformation** constraint; set everything as in the first one, except do not switch **X** with **Z**, set **Destination** = **Loc** and **Z Max** = **0.050**. Rename the constraint as **Transformation_move.L** and collapse it.
42. Assign a fourth **Transformation** constraint; set everything as in the second one, except do not switch **X** with **Z**; set **Destination** = **Loc** and **Z Min** = **0.050**. Rename the constraint as **Transformation_move.R** and collapse it.
43. Save the file.

How it works…

We couldn't directly use the **forearm** and **calf** bones to rig the **vanbraces** and **greaves** parts because being subdivided **B-bones**, they would *curve* these **armor** parts along the length as they actually do by

deforming organic parts as the **forearms** and **shins**, and this would look awkward, as you can see in the following screenshot:

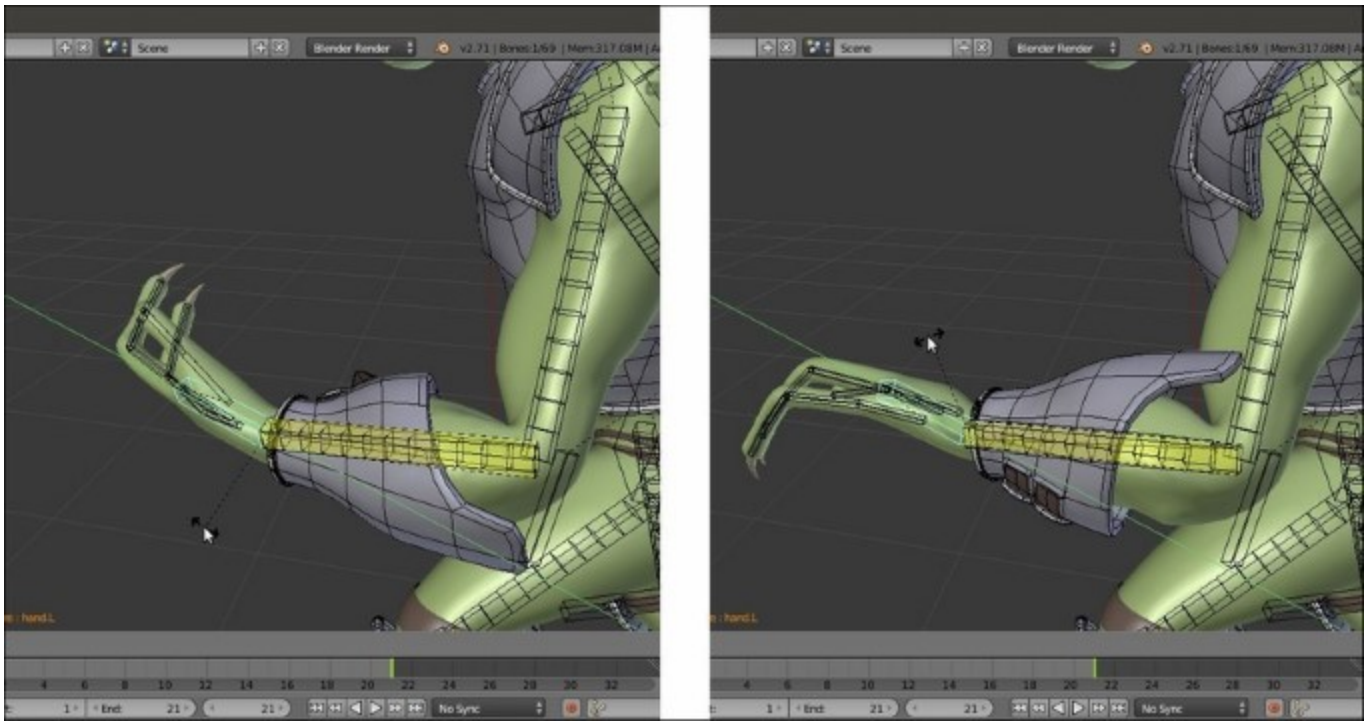


B-bones erroneously deforming stiff objects

Instead, we just duplicated the bones, restored **Segments** and **Ease In** and **Ease Out** to default values, and assigned **2** bone constraints (note that, as already mentioned, the bones have a **Bone Constraints** panel of their own, which is different from the **Object Constraints** one).

The **Copy Rotation** constraint, as the name itself explains, copies the rotation in space of the target **B-bone**; the position inside the chain is granted because the duplicated bones, although not connected, are children of the same bones as the original ones.

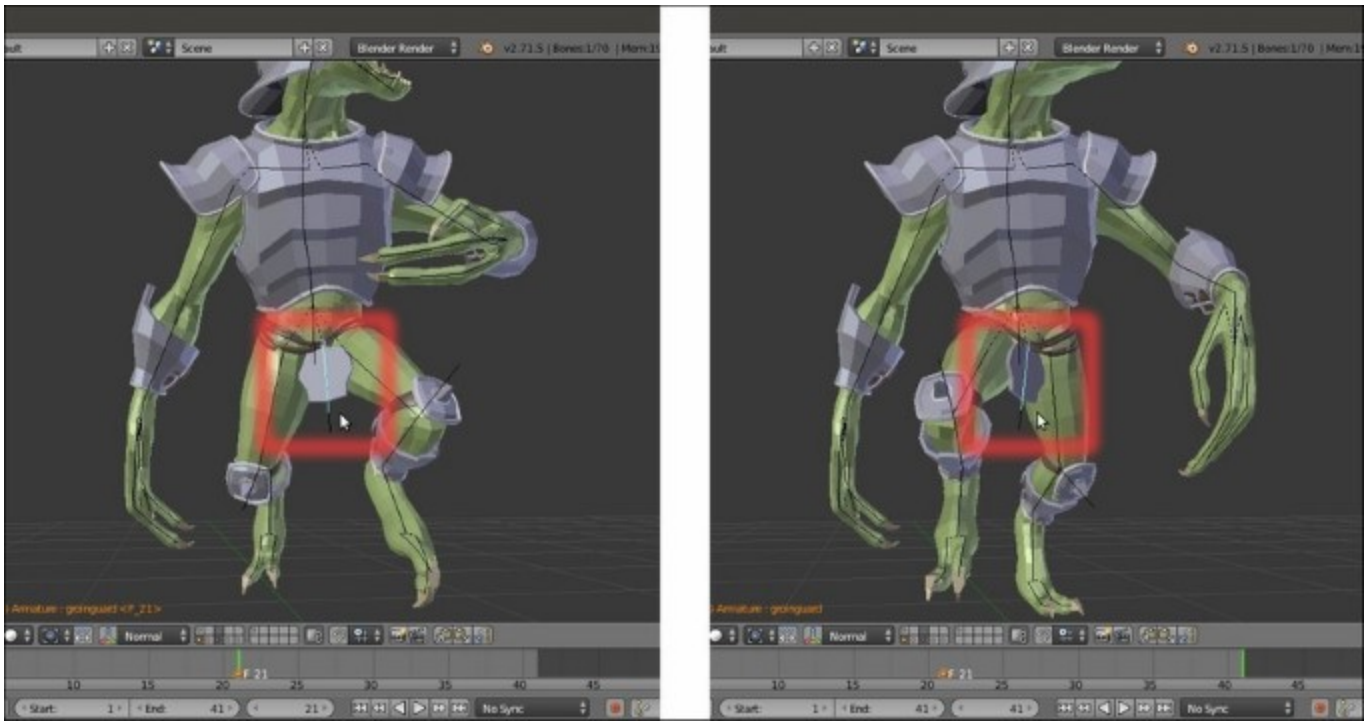
The **Inverse Kinematics** constraint—in this case, is used simply to track the **local y** rotation of the **hand** bone in order to rotate correctly on its *y* axis— is necessary because the **Copy Rotation** constraint doesn't seem to read the **local y** rotation of a subdivided **B-bone** (besides the technical details, it makes sense because that's actually not a rotation in space):



The correct rotation of the stiff armor parts

The constraints assigned to the **groinguard_ctrl** bone are a cheap, but quite an effective, way to fake a rigid body simulation for the **plate** that—in actions, for example, a walk cycle—should interact by colliding with the **Gidiosaurus thighs**. The **Locked Track** constraints, targeted to the **leg** bones, automatically rotate the **plate** according to the **thighs** movements, and the **Dumped Track** constraints, targeted to the **leg** bones as well but with a low influence, add a swinging movement.

The **groinguard** bone, actually the one affecting the **armor plate**, is the child of the **groinguard_ctrl** bone, and so it inherits the constraint's movements but can be used to refine, tweak, or modify the final animation of the plate by hands:



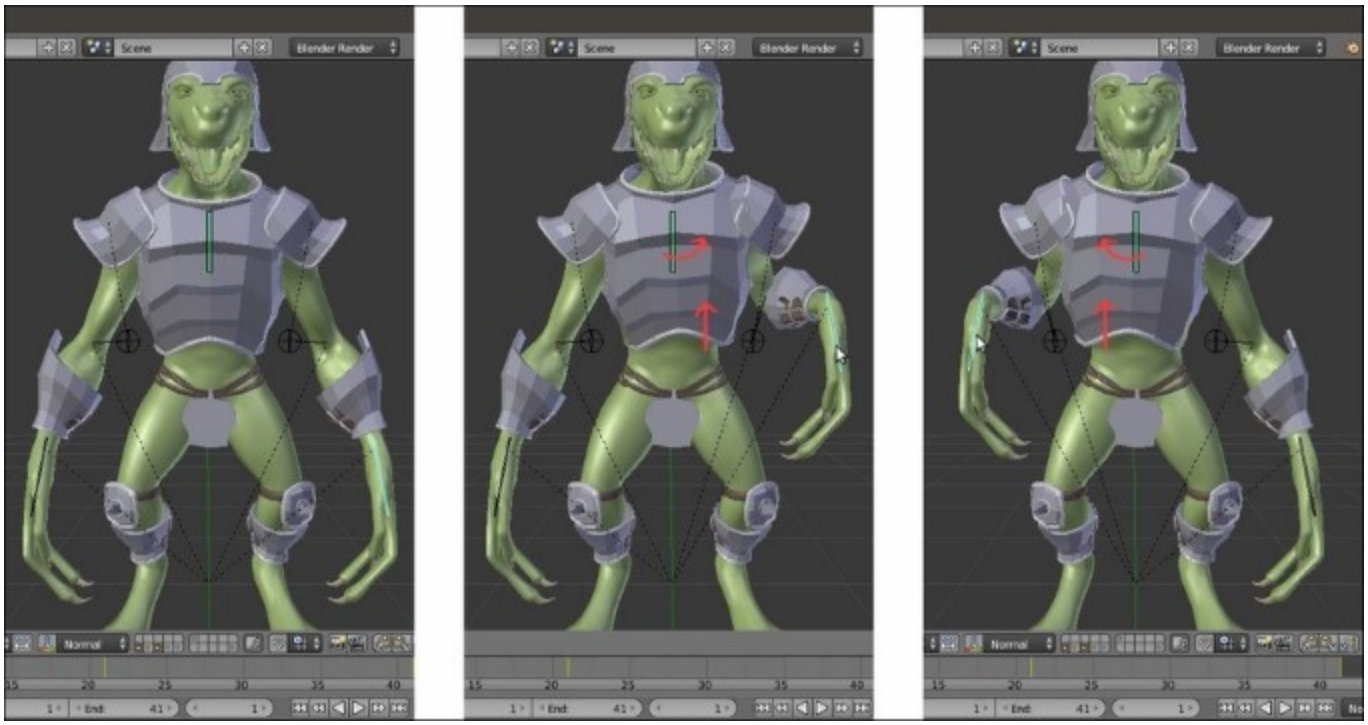
The groinguard bone (and plate) automatically rotating during the walk cycle

The **armor_ctrl** bone is the bone controlling the **armor's Breastplate**; it's the child of the **chest** bone, so it inherits the rotation of the **chest,** but has four **Transformation** constraints.

By using as an input the rotation angle of the **rotarmor.L** and **rotarmor.R** bones (which are children themselves of the **arm.L** and **arm.R** bones), the constraints give to the **Armor chest plate** a slight rotation on the vertical axis and a lateral swinging, driven by the oscillations of the **Gidiosaurus arms**, and simulating of the character's **shoulders** colliding with the **armor plate** during the walk.

Also, the **spaulders** are, in turn, partially rotated by bones with the **Copy Rotation** constraints targeted to the **arms,** but with quite a low influence.

Although better appreciated in motion, the following screenshot will show you the effects as the **arms** rotate backward:



The rotation and swinging of the armor chest plate according to the arms' movements

Building the character's Armature through the Human Meta-Rig

In the previous long and quite complex recipe, we hand-built the deforming elements of an average basic rig for the **Gidiosaurus** character; actually, in Blender, there are other tools to build rigs, particularly meant to facilitate the task, and we'll see them in this recipe and in the following ones.

Now, we are going to take a look at the **Human Meta-Rig** tool.

Getting ready

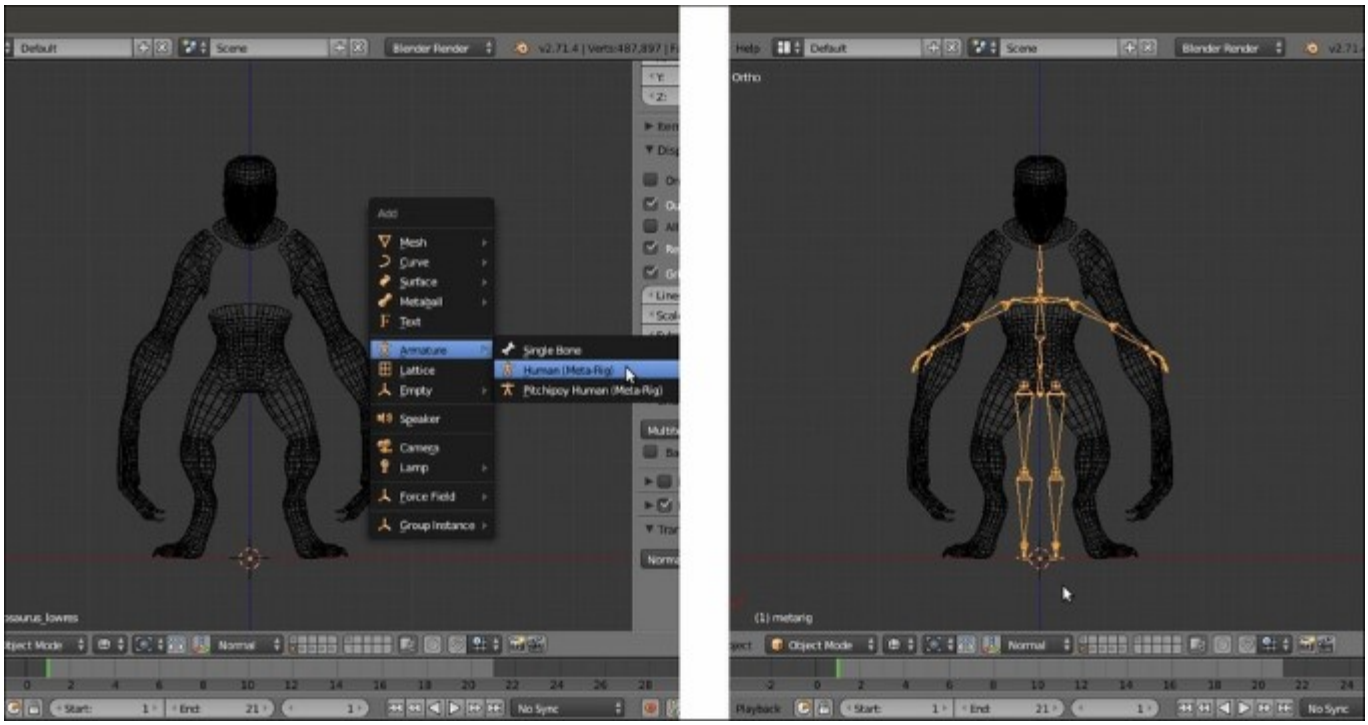
To be able to use the **Human Meta-Rig** tool, we must first enable the proper add-on:

1. Start Blender and press *Ctrl* + *Alt* + *U* to call the **User Preferences** panel. Go to the **Add-ons** tab and under **Categories** on the left-hand side, click on the **Rigging** item. Go to the right-hand side of the panel and check the box to the side of the **Rigging: Rigify** add-on to enable it.
2. Click on the **Save User Settings** button at the bottom-left of the panel and then close it. Because we are starting a rig from scratch again, load the `Gidiosaurus_unwrap_final.blend` file.
3. Disable the **Textured Solid** and **Backface Culling** items in the 3D view **Properties** panel, join the 3D window with the **UV/Image Editor** window, and click on the **11th** scene layer to have only the **Gidiosaurus** mesh visible.
4. Go to the **Object** window and under the **Display** subpanel, enable the **Wire** item; this will be useful in the process to have an idea of the mesh topology when in **Object Mode** and in **Solid** viewport shading mode. However, for the moment, press *Z* to go in the **Wireframe** viewport shading mode.
5. Press *1* on the numpad to go in the **Front** view and *5* on the numpad again to switch to the **Ortho** view.
6. Save the file as `Gidiosaurus_meta_rigging.blend`.

How to do it…

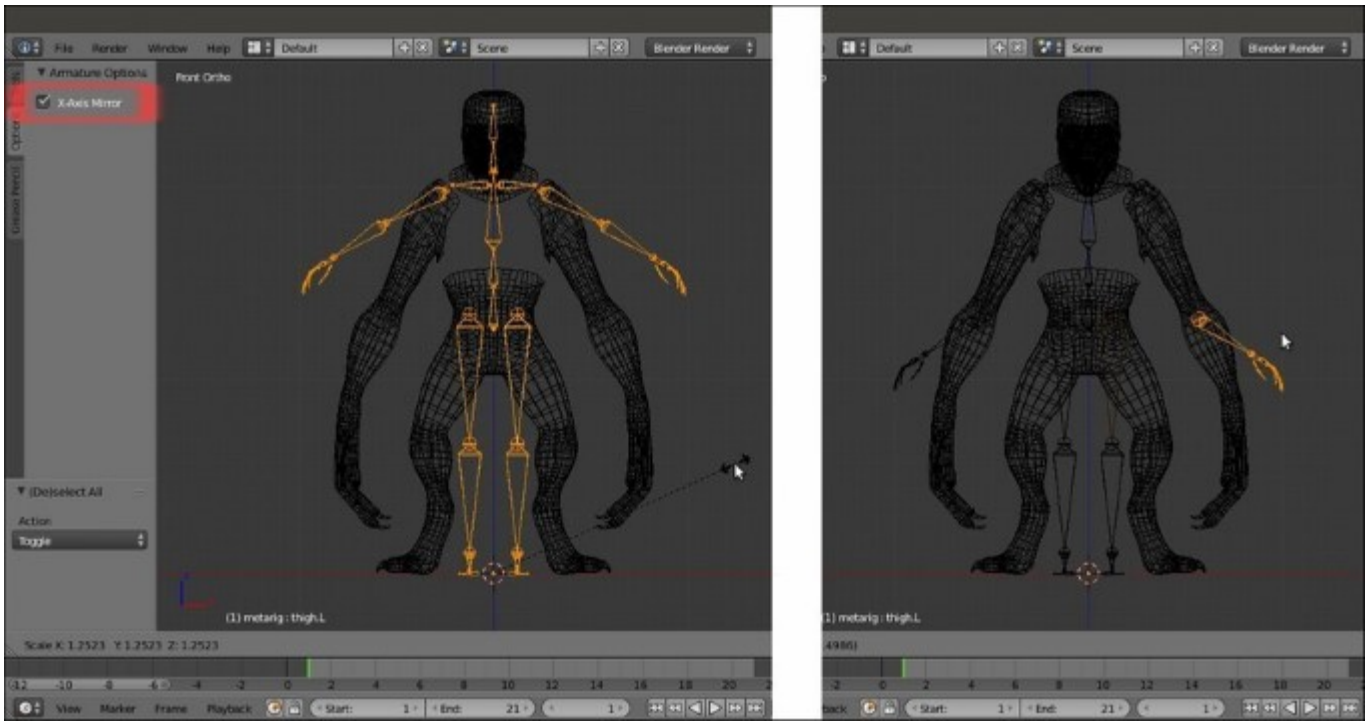
Let's go with the **metarig** itself:

1. Ensure that the **3D Cursor** is at the origin pivot point of the **Gidiosaurus** mesh. Put the mouse cursor in the 3D viewport, press *Shift* + *A*, and in the pop-up menu, select **Armature** | **Human (Meta-Rig)**; a biped **Armature**, automatically named **metarig** in the **Outliner**, appears at the **3D Cursor** location:



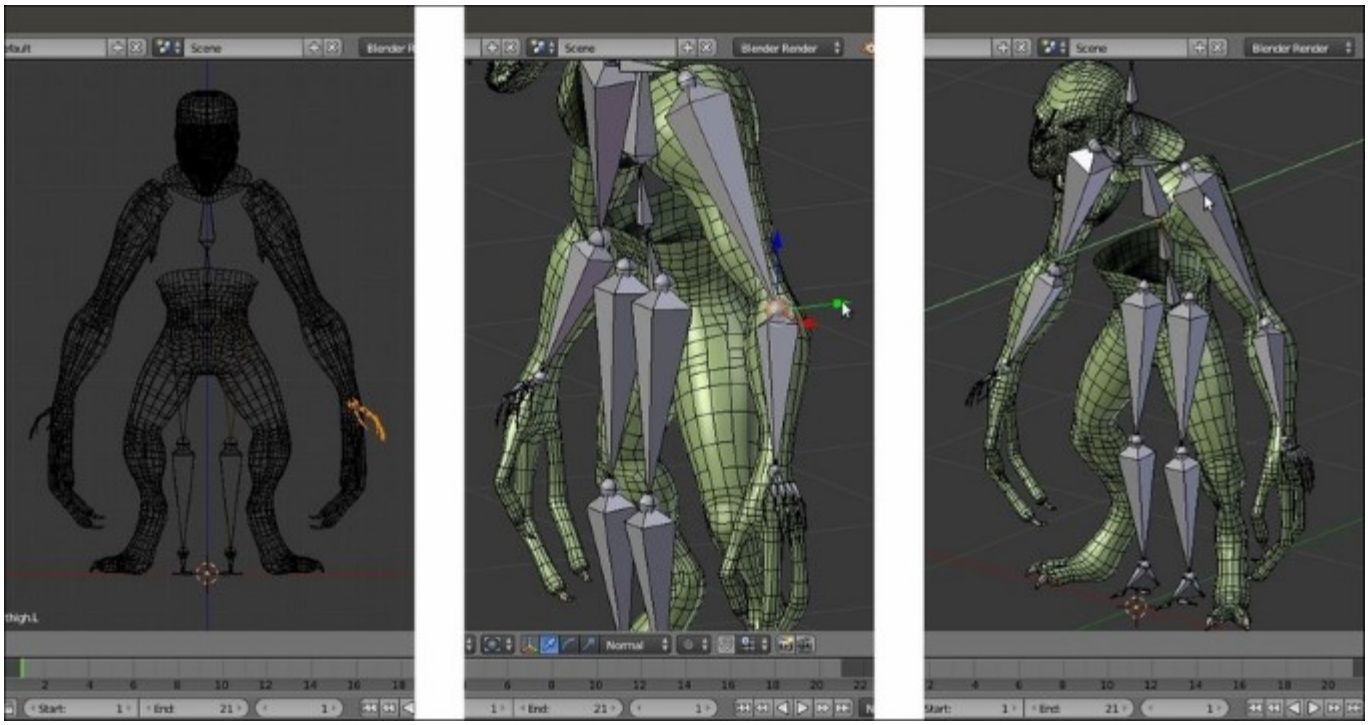
The Human (Meta-Rig) menu and the rig

2. Press the *Tab* key to enter **Edit Mode** and go to the **Options** tab that appeared under the **Tool Shelf** on the left-hand side of the screen; check the box to enable the **X-Axis Mirror** tool under the **Armature Options** item.
3. First, press the period (.) key to set the pivot point around the **3D Cursor** and scale the whole armature bigger while still in **Edit Mode**, and then start to edit locations and proportions of the bones of the **metarig** to fit inside the **Gidiosaurus** shape:



Tweaking the proportions of the bones of the metarig

4. Select the single joints to move them on the right location according to the mesh topology; to do this in a more exact way, just use the snap technique explained in steps 25 and 26 of the *How to do it…* section of the *Building the character's Armature from scratch* recipe. Because of the **X-Axis Mirror** tool we enabled, it's enough to operate only on one side of the **metarig**:



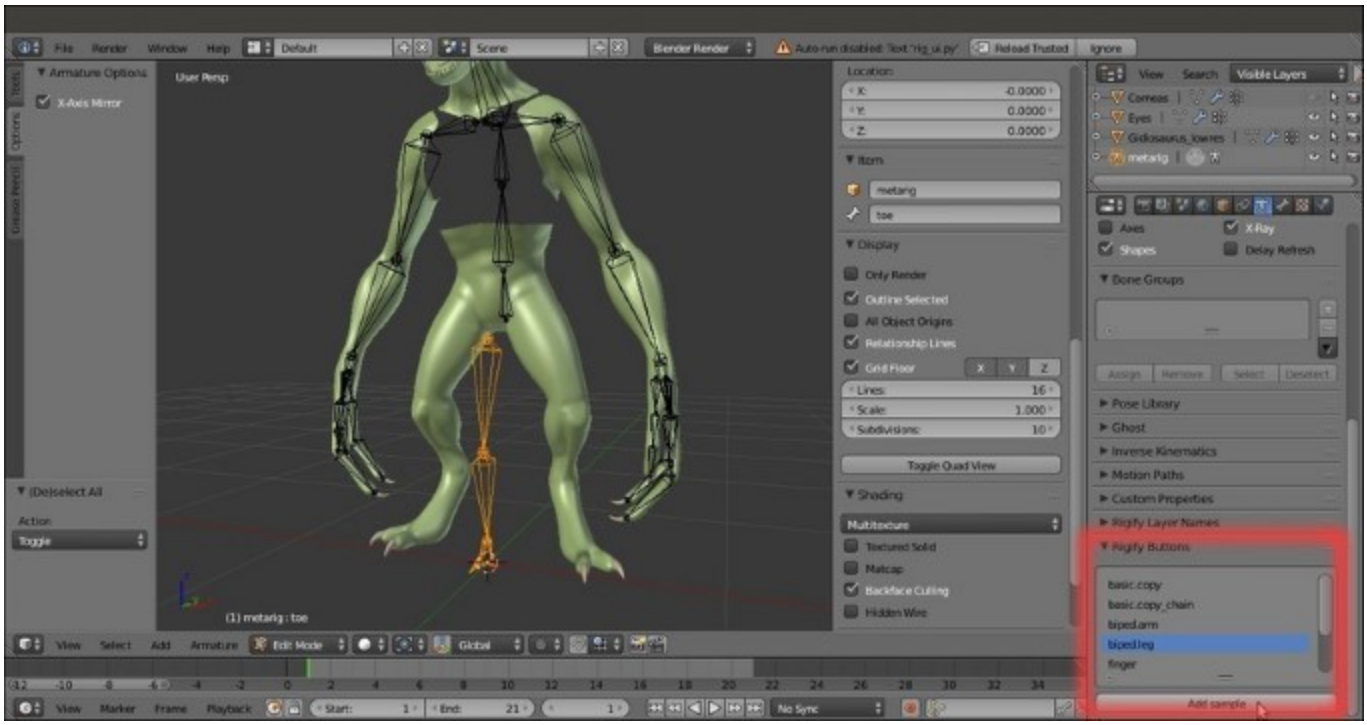
Further tweaking of the bones in Edit Mode

5. Delete the bones that you don't need, for example the extra **fingers** (consider that the **Gidiosaurus** has only three **fingers** in each **hand**), use *Shift + D* to duplicate the bones to be added, for example for the **toes**, and add new bones where missing, for example for the **jaw**, and then parent them. In short, just edit the rig as usual. Again, it should be enough to do all these operations just on one side of the rig:

The completed skeleton rig

6. Save the file.

We can also add premade rigging sets, for example a whole new **leg**, **spine**, or **arm**, by going, with the **metarig** still in **Edit Mode**, to the **Rigify Buttons** subpanel under the **Armature** window in the main **Properties** panel. Select the desired item to be added to the rig and click on the **Add sample** button; the new part gets added to the rig's **pivot point** location and must be moved to the right place and tweaked, rotated, and scaled as needed. Also, the new bones must be named with the correct **.R** or **.L** suffix and the top chain bone must be parented to the bottom **metarig** bone; for example, in the case of a **biped.leg** part addition, the **thigh** bone must be parented (*Ctrl + P* | **Keep Offset**) to the **hips** bone:



Adding premade rig to the skeleton

How it works…

The **Human metarig** is actually only the first part of a more complex and complete auto-rigging system named **Rigify**, and this we'll see in the next recipe. However, even used by itself, it gives us a readymade humanoid skeleton to be simply tweaked to fit the character's shape: a good shortcut to quickly build the **Armature** rig considering that, at least in its basic form, all the bones are already properly connected and named with the **.L** and **.R** suffices.

Building the animation controls and the Inverse Kinematic

Whether we built the **Gidiosaurus** deforming rig part by hands from scratch or by the **Human Meta-rig**, we must now add the necessary constraints and controls to allow the animators to easily manipulate the character.

Note

Note that once the mesh is skinned, the rig, as it is at this point, can actually already work by directly selecting the interested bones and rotating them in **Forward Kinematics**; however, to simplify the animator's work (and complicate our life a little bit more), it's good practice to add the **Inverse Kinematic** constraints and the control bones.

Getting ready

Let's start by opening the `Gidiosaurus_rig_from_scratch_02.blend` file; as usual, enter **Edit Mode** to ensure that the **X-Axis Mirror** item in the **Armature Options** subpanel under the **Tool Shelf** is enabled.

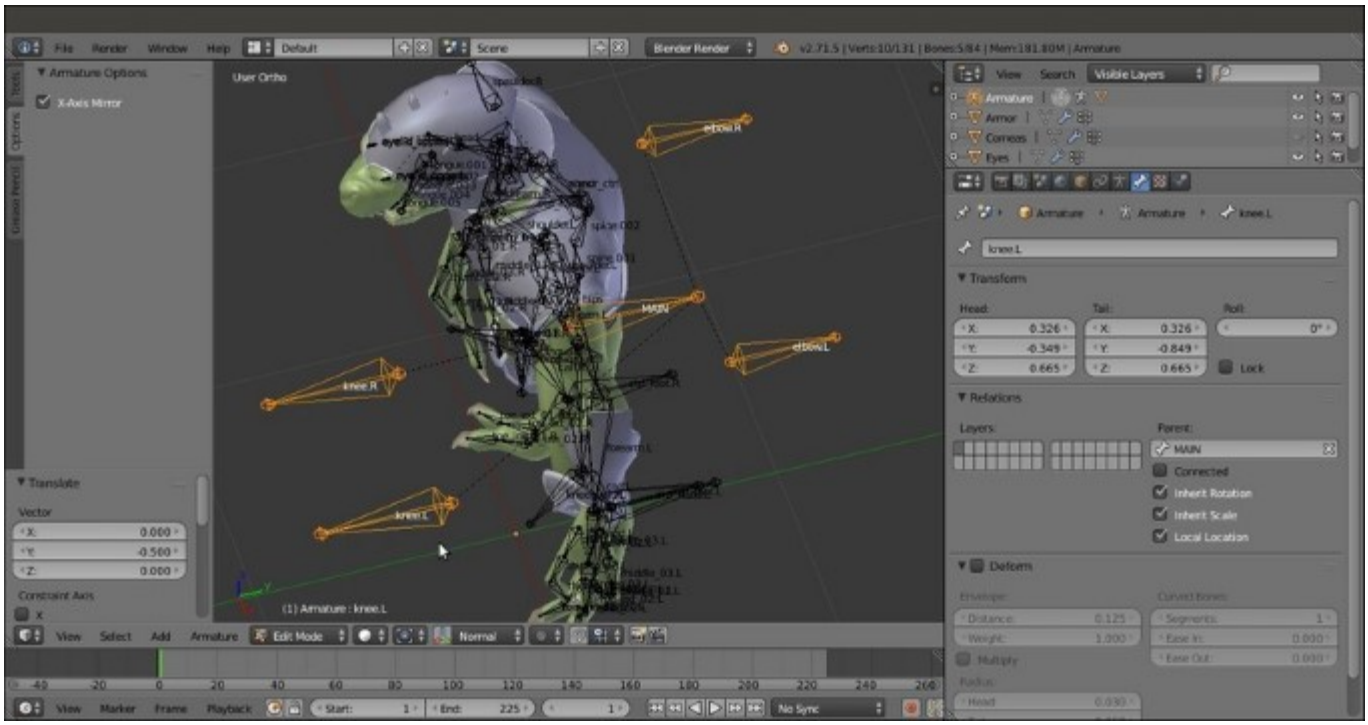
How to do it…

We now need to create the control bones; we can do it by extruding from the bones they will drive:

1. Press the *3* key on the numpad to go in the **Side** view and, if not already, select the **Armature**; if necessary, in the **Display** subpanel, change the visualization of the bones from **B-Bone** to **Octahedral**.
2. While still in **Edit Mode**, use *Shift* to select the joints of the **hand** with the **forearm** and the **calf** with the **foot** (it's enough only on one side) and extrude them going backwards (**0.400** along global *y* axis).
3. Rename the new extruded bones as **ctrl_hand.L**, **ctrl_hand.R**, **ctrl_foot.L**, and **ctrl_foot.R** respectively. Deselect the **Deform** item and unparent them all.
4. Select the **Head** of the **hips** bone and repeat: rename the extruded bone as **MAIN**.
5. Select the **hips** bone and in the **Relations** subpanel, parent it as a child of the **MAIN** bone:

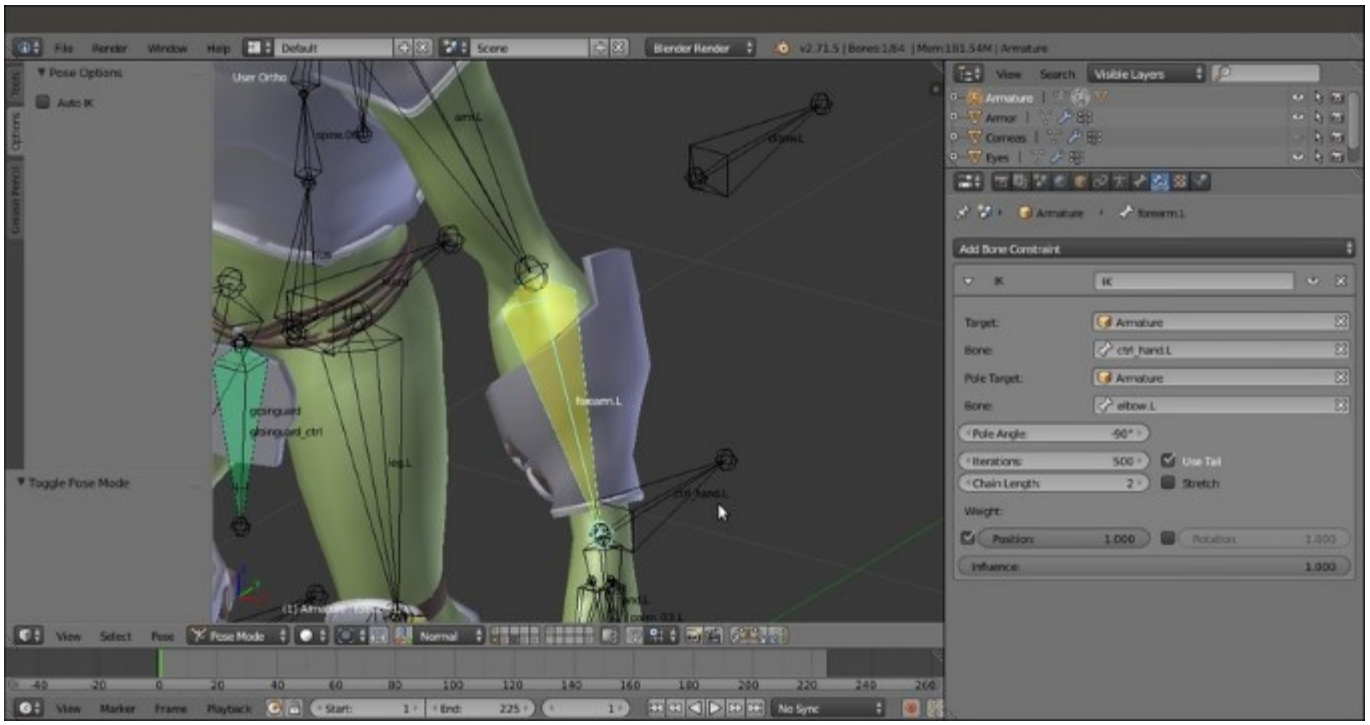
Extruding the control bones part 1

6. Select the **elbow** joint (between the **forearm** and **arm**) and extrude a new bone backwards; rename the extruded bone and the mirrored one as **elbow.L** and **elbow.R**. Disable the **Deform** item and parent them (**Keep Offset**) to the **MAIN** bone. Move them backwards by **0.500** along the global *y* axis.
7. Select the **knee** joint (between the **thigh** and **calf**) and extrude forward; rename the new bones as **knee.L** and **knee.R**. Disable the **Deform** item and parent them (**Keep Offset**) to the **MAIN** bone as well. Move them forward by **-0.500** along the global *y* axis;



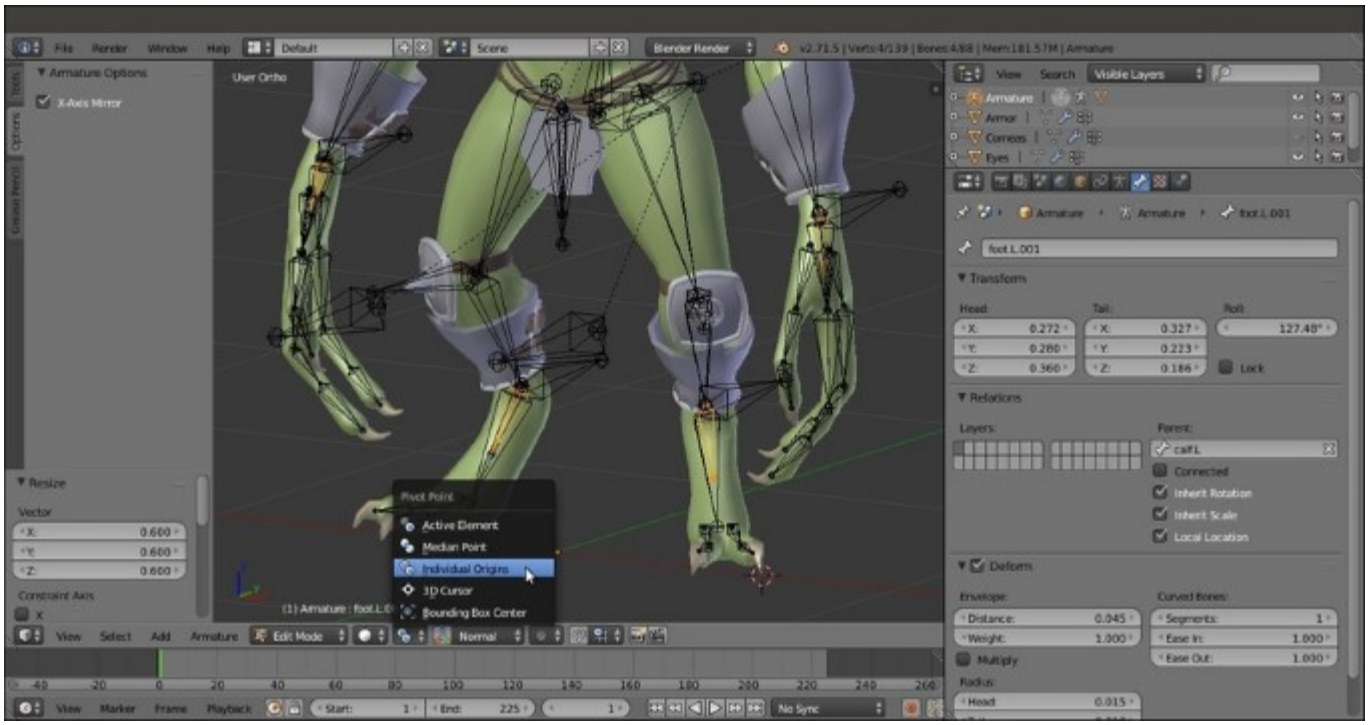
Extruding the control bones part 2

8. Go into **Pose Mode** and select the **forearm.L** bone; go to the **Bone Constraints** window and assign an **Inverse Kinematics** constraint. Set **Target** = **Armature**, **Bone** = **ctrl_hand.L**, **Pole Target** = **Armature**, **Bone** = **elbow.L**, **Pole Angle** = **-90°**, and **Chain Length** = **2**, and deselect **Stretch**. Repeat the process for the **forearm.R** bone:



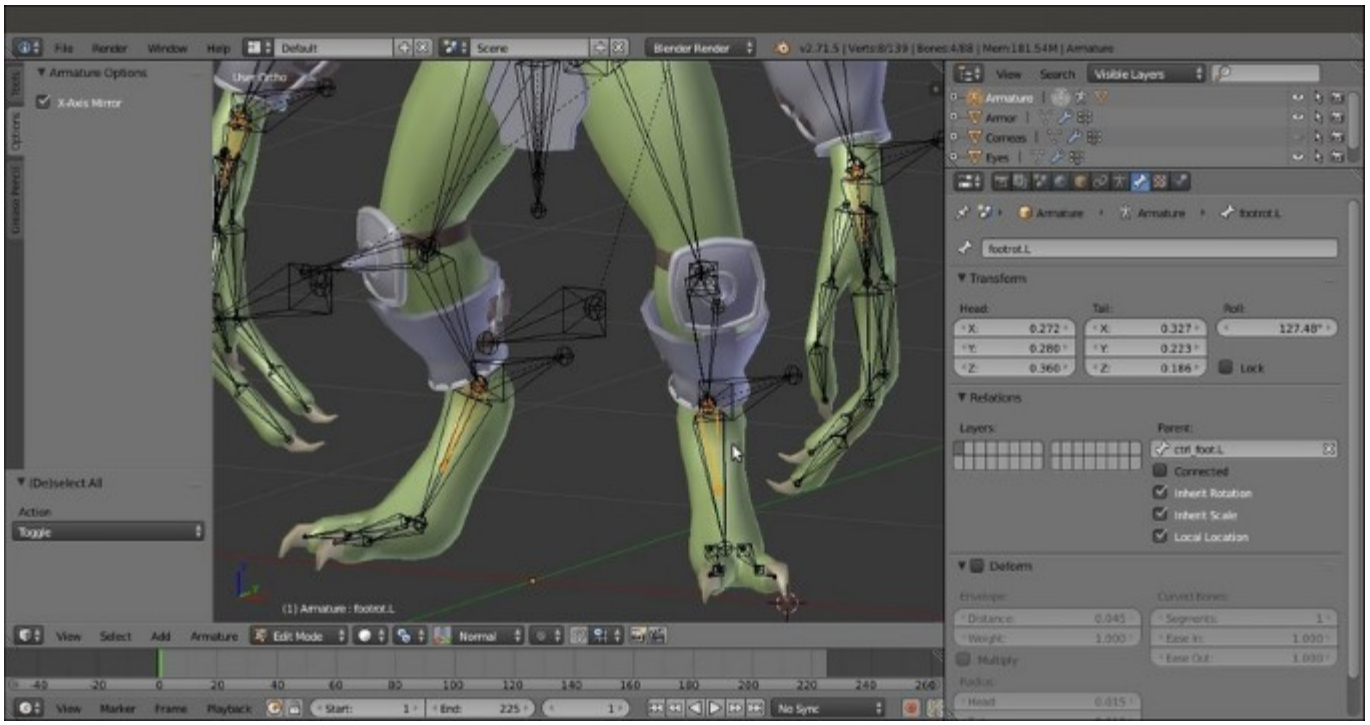
Assigning the IK constraint to the forearm.L bone

9. Do the same for the **calf.L** and **calf.R** bones, using the **ctrl_foot.L** and **ctrl_foot.R** bones as targets and the **knee.L** and **knee.R** bones as poles, but set the **Pole Angle** to **90°** for both.
10. Now, go back into **Edit Mode**, select the **hand** and **foot** bones, and use *Shift + D* to duplicate them. Click on the **Pivot Point** button on the 3D view toolbar, select the **Individual Origins** item, and then scale the duplicated bones smaller to **0.600**:



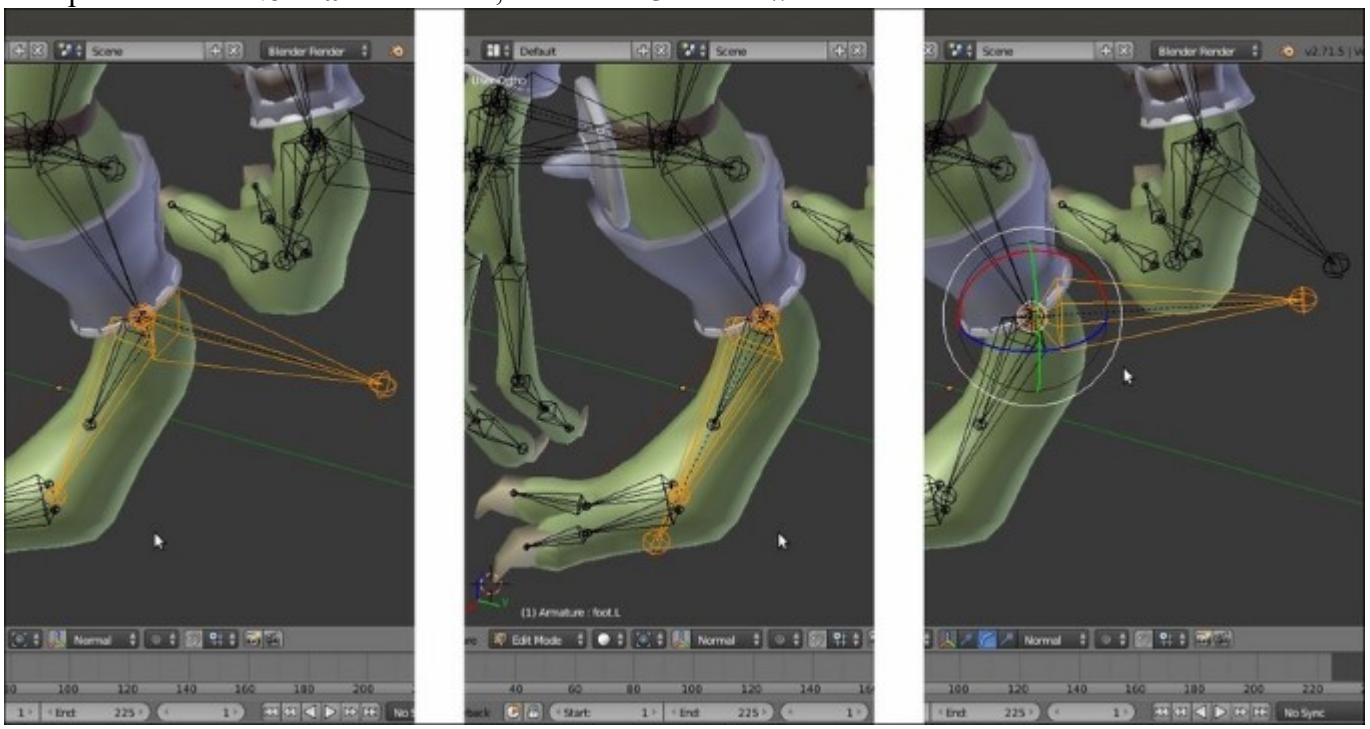
Scaling the bones smaller on their individual origin

11. Deselect the **Deform** item for all of them, and then rename them as: **handrot.L**, **handrot.R**, **footrot.L**, and **footrot.R**.
12. In the **Relations** subpanel (or by the *Ctrl* + *P* | **Keep Offset** shortcut), parent **handrot.L** to **ctrl_hand.L**, **handrot.R** to **ctrl_hand.R**, **footrot.L** to **ctrl_foot.L**, and **footrot.R** to **ctrl_foot.R**:



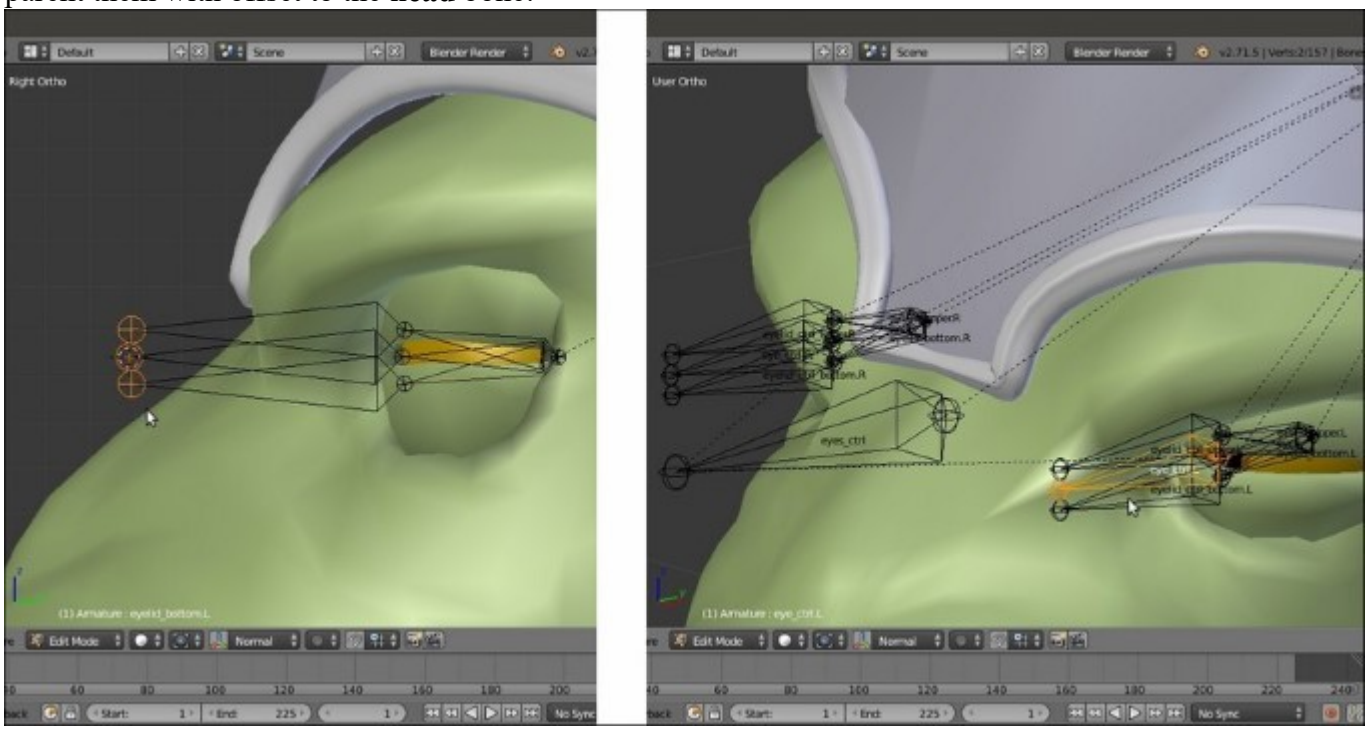
Using the Parent slot under the Relations subpanel

13. Use *Shift* to select the **ctrl_foot.L** bone and the **foot.L** bone and press *Ctrl* + *Alt* + *A* to align the first one with the active one; then, select only the **ctrl_foot.L** bone, and by the toolbar widget manipulator set to **Normal** orientation, rotate it **245°** on the *x* axis:



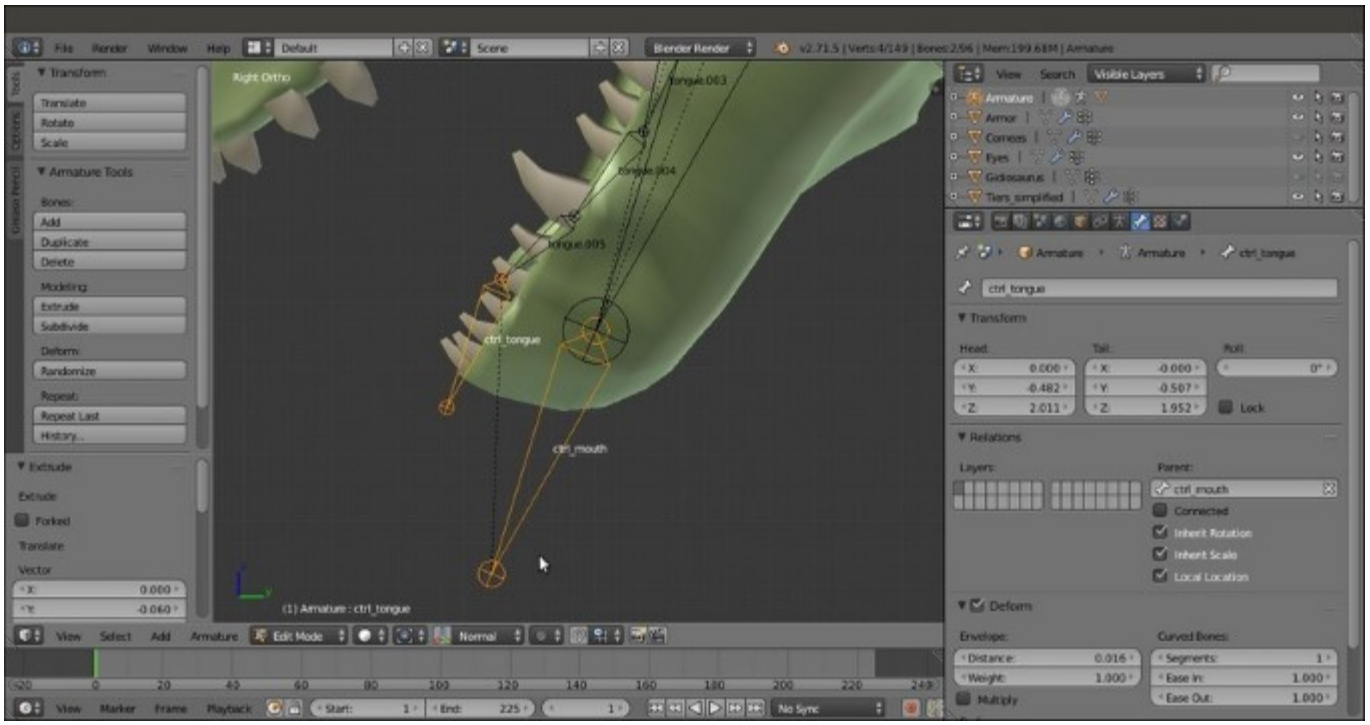
Rotating the bone on the Normal orientation by the widget

14. Go into **Pose Mode** and select the **hand.L** bone; assign a **Copy Rotation** bone constraint with **Target** = **Armature** and **Bone** = **handrot.L**, and set **Space** = **Pose Space** to **Pose Space**.
15. Repeat for the other **hand** bone and **feet**.
16. Select the **Tails** of the **eyelid_upper.L**, **eyelid_bottom.L**, and **eye.L** bones and extrude forward by **0.0600** along the *y* axis; rename them as **eyelid_ctrl_upper.L**, **eyelid_ctrl_bottom.L**, and **eye_ctrl.L** and the same names with the **.R** suffix for the mirrored ones.
17. Add a new bone in the middle front of the **eyes**, rename it **eyes_ctrl**, and parent it with offset to the **head** bone; then, select the **eye_ctrl.L** and **eye_ctrl.R** bones and parent them with offset to the **eyes_ctrl** bone.
18. Select the **eyelid_upper.L**, **eyelid_upper.R**, **eyelid_bottom.L**, and **eyelid_bottom.R** bones and parent them with offset to the **head** bone:



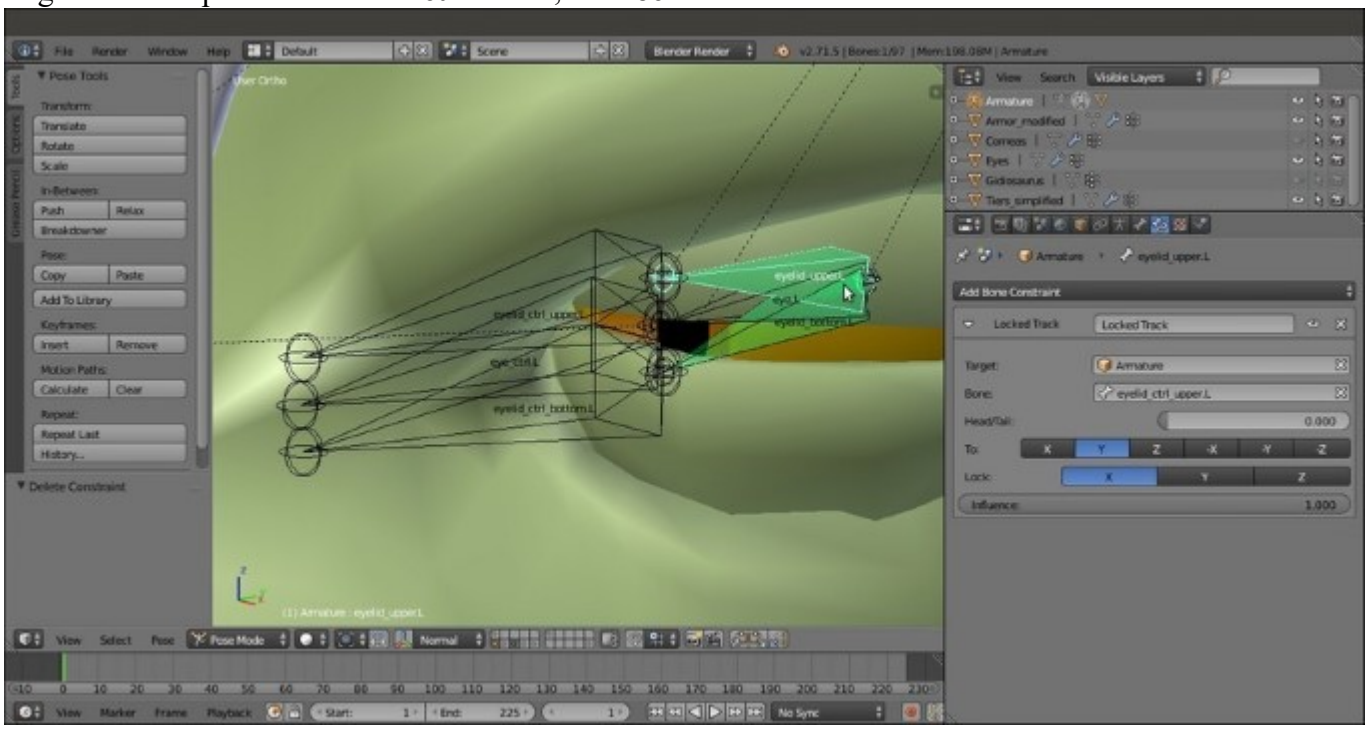
The eyes control rig

19. Select the **Tails** of the **mand** and **tongue.005** bones and extrude; rename the extruded bones as **ctrl_mouth** and **ctrl_tongue**. Parent with offset the **ctrl_tongue** bone to the **ctrl_mouth** bone and this latter bone to the **head** bone:



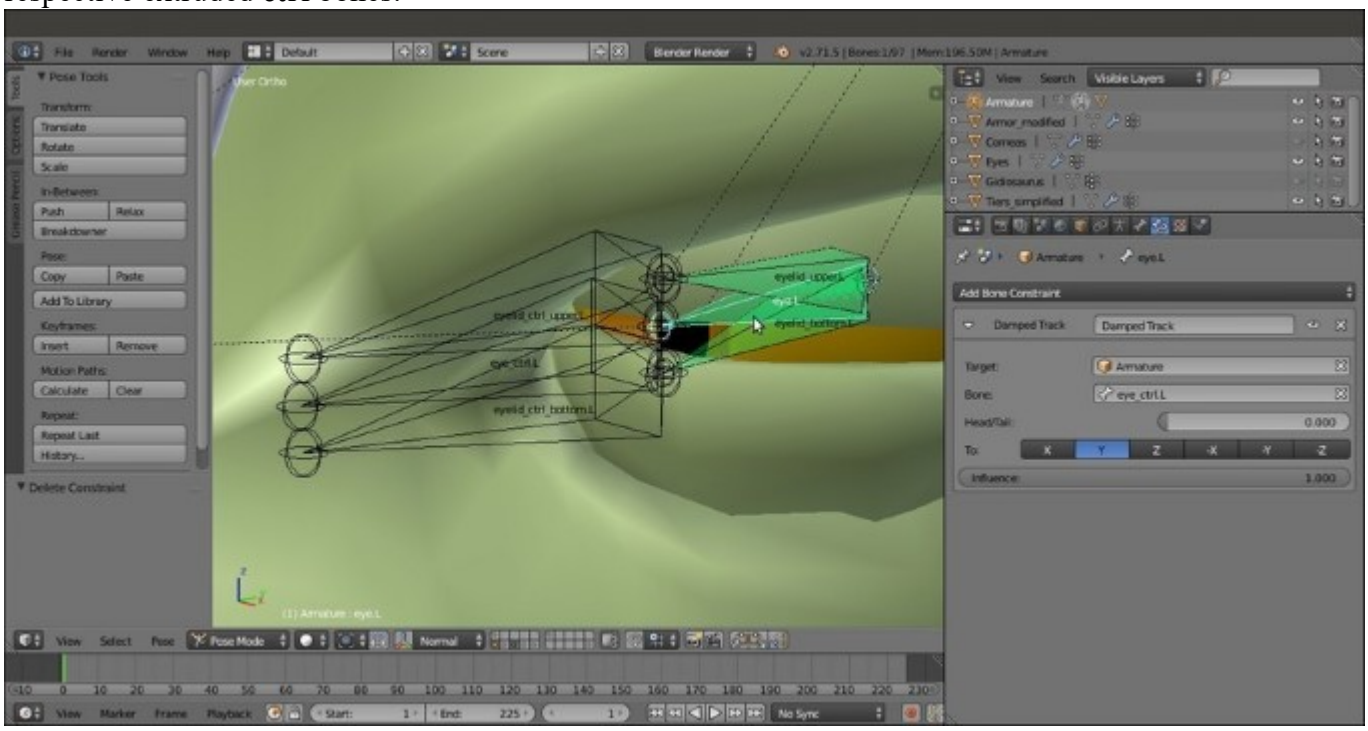
Extruding the control bones for the tongue and jaw

20. Go into **Pose Mode** and assign **Locked Track** constraints to the **eyelid_upper** and **bottom** with target to the respective extruded **ctrl** bones; set **Lock** to **X**:



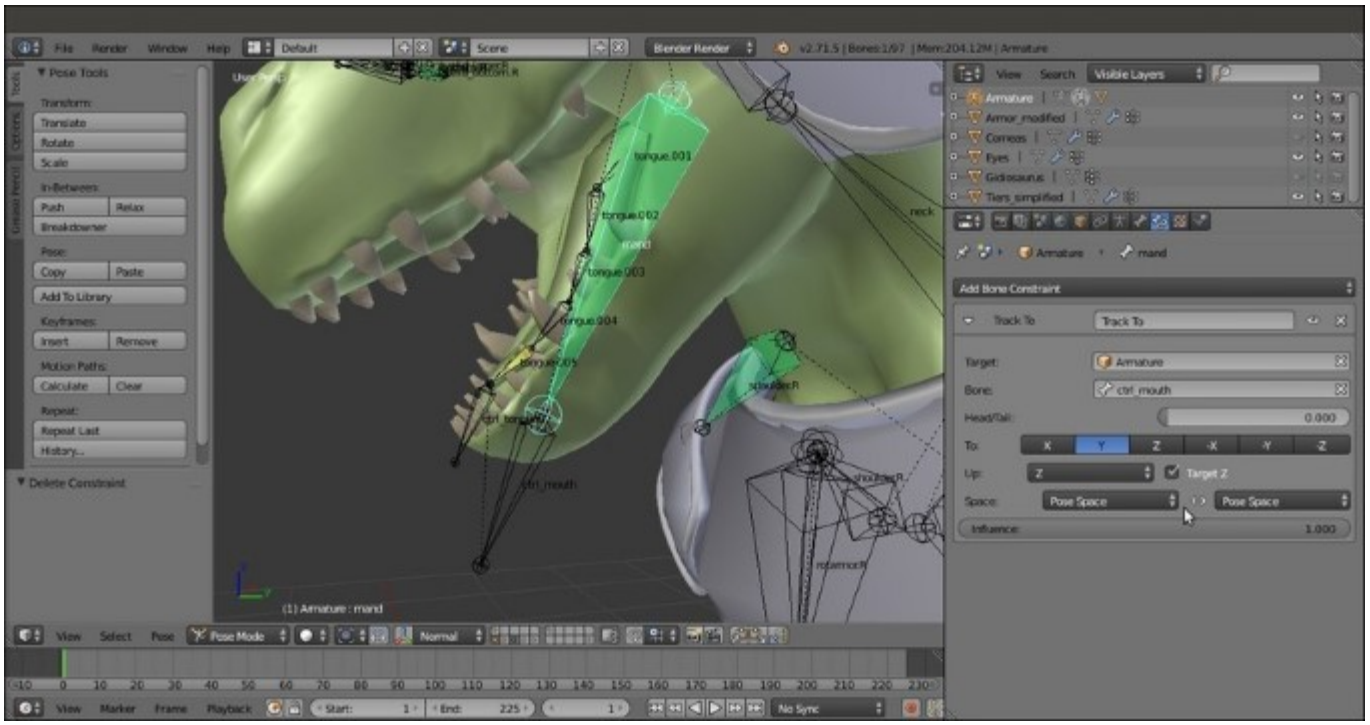
Assigning the Locked Track constraints for the eyelid's controls

21. Assign **Damped Track** constraints to the **eye.L** and **eye.R** bones, again with target to the respective extruded **ctrl** bones:



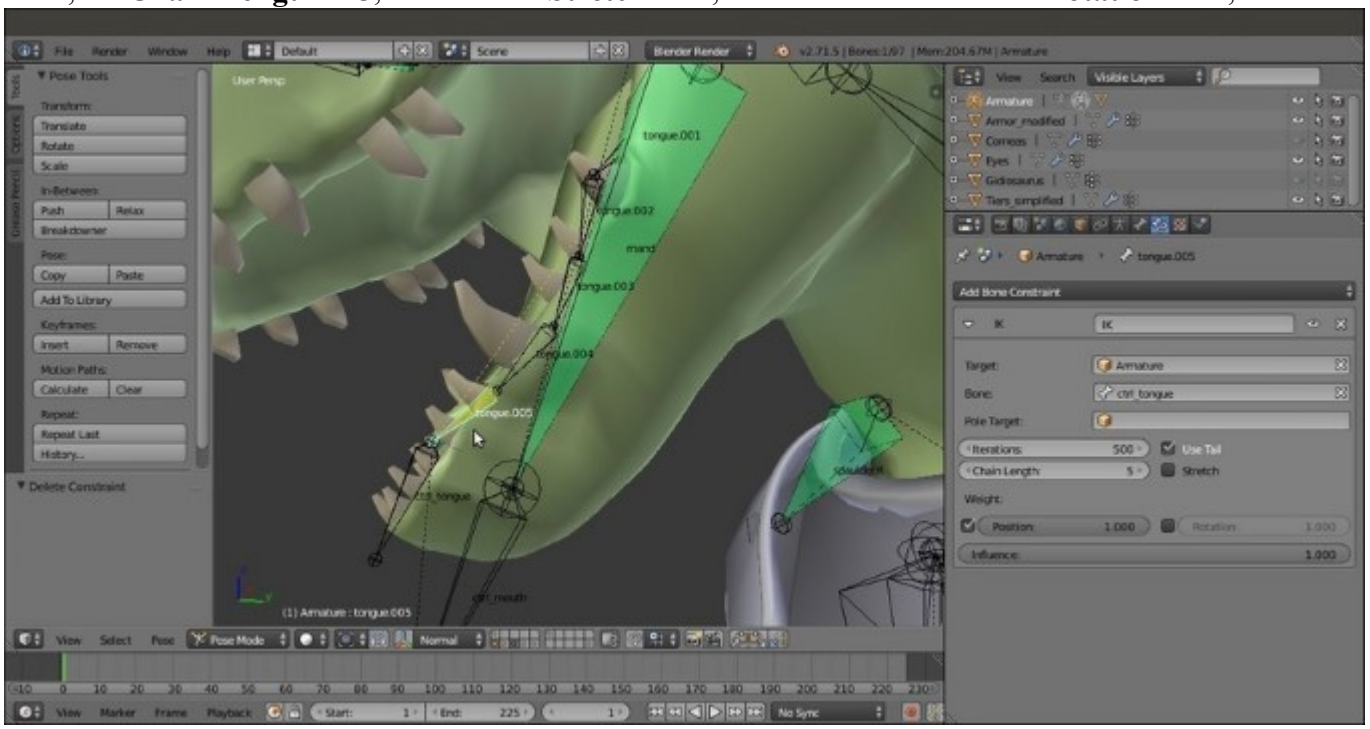
Assigning Damped Track constraints for the eye's controls

22. Assign a **Track To** constraint to the **mand** bone with target to the **ctrl_mouth** bone; check the **Target Z** item box and set **Space** = **Pose Space** to **Pose Space**:



Assigning a Track To constraint to the mand bone

23. Assign an **Inverse Kinematics** constraint to the **tongue.005** bone with target to the **ctrl_tongue** bone; set **Chain Length** to **5**, deselect the **Stretch** item, and then enable also the **Rotation** item;

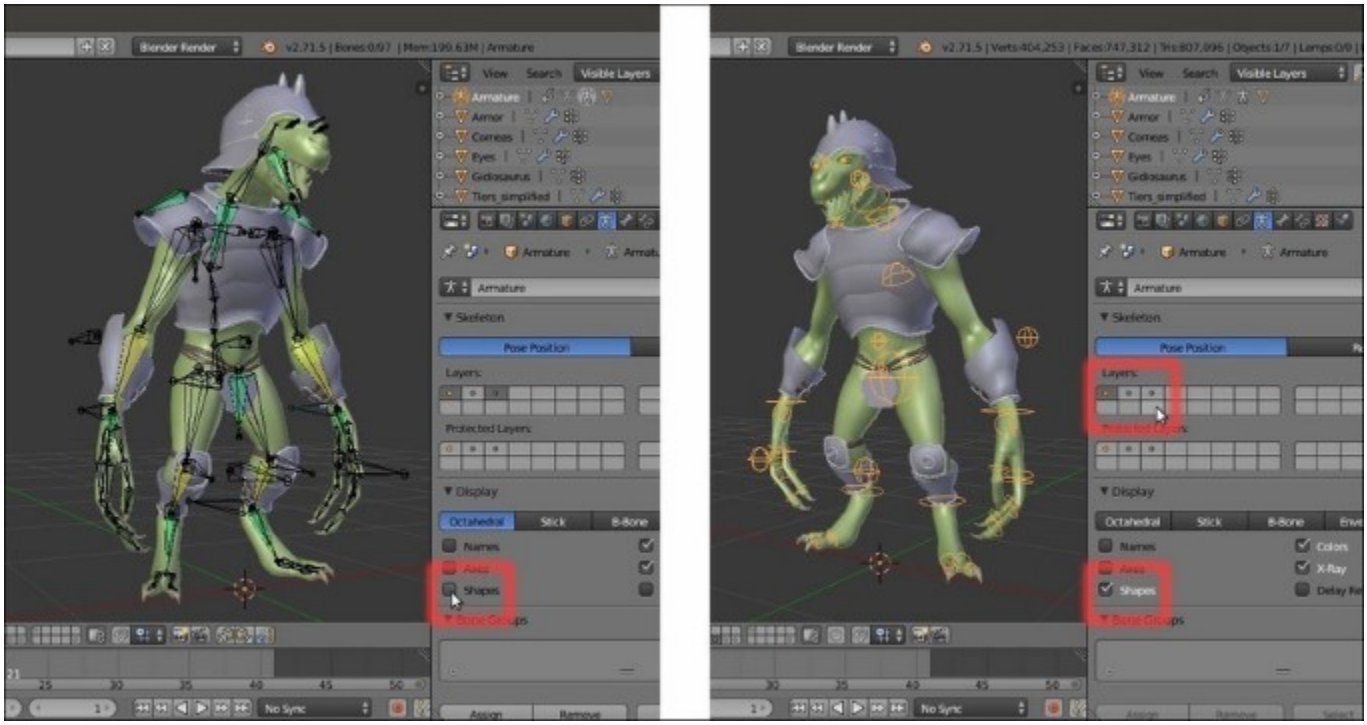


The IK constraint for the bone's chain of the tongue

At this point, the main controls for the **Gidiosaurus** rig are made; still something is missing, for example, the controls to drive **fingers** or/and **toes** bones as a whole, and also a *muscle system* layer of bones with the **Stretch To** constraints that can be added to improve the realism of the model. However, this latter option is quite a complex matter and, for the moment, we will stop here (maybe in another book).

The very last thing to do is to assign **Custom Shapes** (usually, simple meshes located on the last scene layer) to the control and animatable bones widget, and move the rest of the bones to the third **Armature** layer to be out of view.

To see the completed rig with the **Custom Shapes** assigned to the control bones, load the `Gidiosaurus_rig_from_scratch_03.blend` file;



The rig with and without Custom Shapes and with the deformation bones hidden on the third (disabled) Armature layer

See also

- http://www.blender.org/manual/rigging/index.html

Generating the character's Armature by using the Rigify add-on

We have already seen that the **Human Meta-Rig** armature is part of the **Rigify** add-on. It is a tremendously useful Python script, coded by Nathan Vegdhal, that we enabled two recipes ago, and in this recipe, we are going to use that to build the final rig for the **Gidiosaurus**.

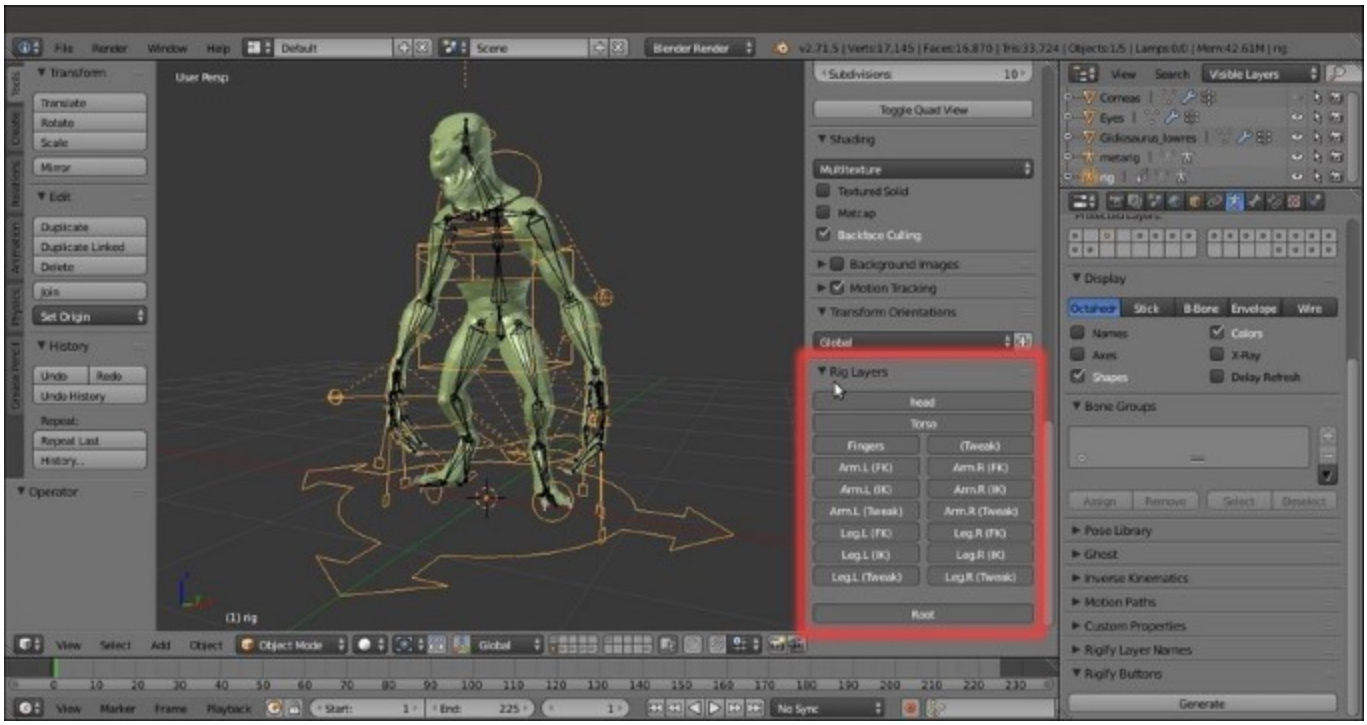
Getting ready

The preparation steps to use the **Rigify** add-on are the same as we did in the *Building the character's Armature through the Human Meta-Rig* recipe: after we have enabled the add-on in the **User Preferences** panel, we load the `Gidiosaurus_unwrap_final.blend` file, add the **Human metarig** to the scene, and then tweak the bone's position, rotation' and size in **Edit Mode** to fit the character's shape and topology.

Also, because the rig generated by the **Rigify** add-on uses some Python script, in the **User Preferences** panel, we must enable the **Auto Run Python Scripts** item (in the **File** | **User Preference** | **File** tab, click on the **Auto Run Python Scripts** checkbox).

How to do it…

At this point, in **Object Mode**, we can go to the bottom of the **Armature** window under the main **Properties** panel and click on the **Generate** button in the **Rigify Buttons** subpanel at the bottom of the **Armature** window; the add-on will automatically generate a new **rig** (simply named **rig** in the **Outliner**) using the **metarig** skeleton as an input and adding all the necessary **IK** constraints, the bone's widget controls (generated and located in the last scene layer), and also placing the different bones on different **Armature** layers that are easily accessible through the Python interface created by the script in the 3D window **Properties** sidepanel on the right-hand side (the **Rig Layers** subpanel):



The generated rig with the Rig Layers subpanel

Keep the **metarig** and move it to another layer, just in case we need to do some editing to it in the future; in fact, by testing the generated rig, sometimes you discover that something must be changed to work in a different way. In this case, it is enough to modify the **metarig** and generate the rig again by the add-on that automatically reuses the elements of any already existing rig and the bone's widgets on the last scene layer.

Keep in mind that the generated rig can (and often must) be edited later anyway; after the rig generation, save the file as `Gidiosaurus_rigify_01.blend`.

How it works…

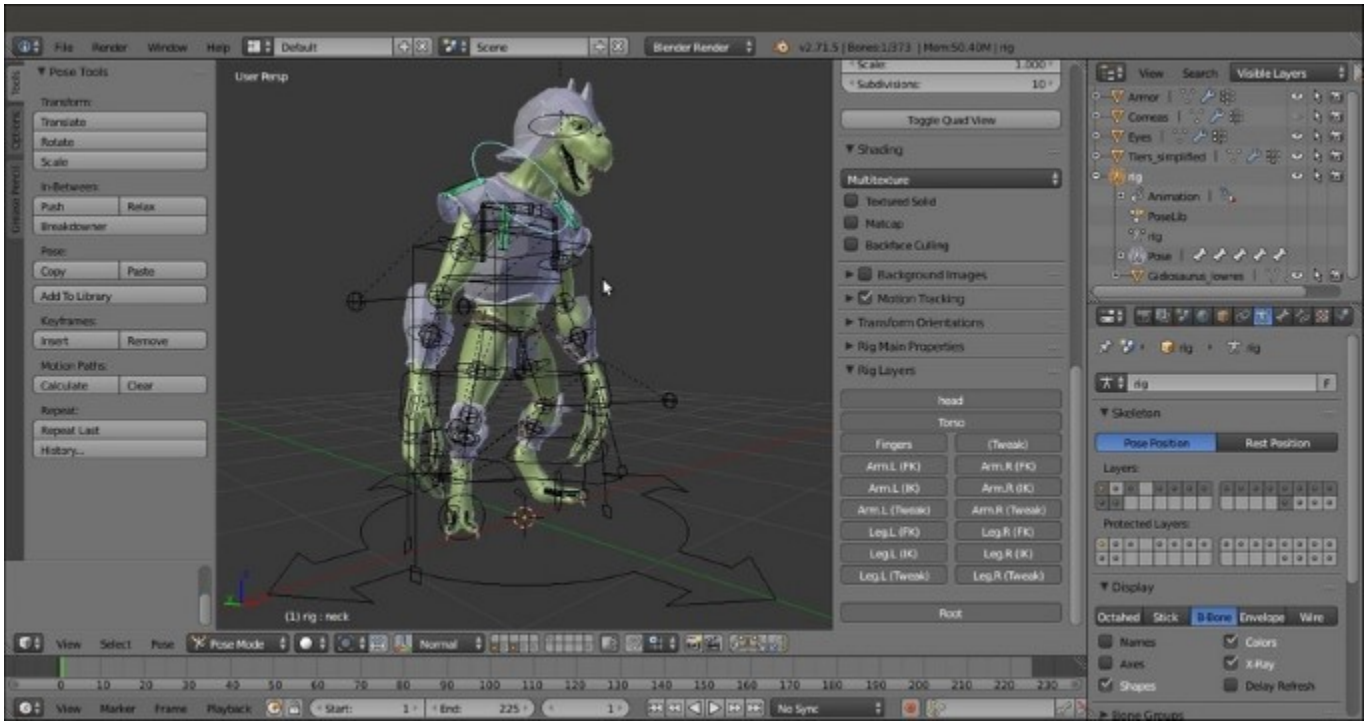
Being conceived to build a rig for a *generic biped humanoid* character, the **Rigify** add-on doesn't generate everything you need automatically: in our case, bones for the **jaw**, **tongue**, **eyes**, and **eyelids** must be added by hands after the rig regeneration and as explained in the *Building the character's Armature from scratch* recipe.

The choice to let face-rig elements, at least initially, out of the **Rigify** add-on has been intentional by Vegdhal, who thinks that a face-rig tool would probably be better as a separate add-on. By the way, in the last Blender releases, it is available, in the **Armature** menu, a **Pitchipoy human rig** option, which is an addition to the **Rigify** script that should help in the face's rig construction (http://pitchipoy.tv/?p=2026).

Also, at least for the moment, the **Rigify** add-on doesn't accept custom rig parts, but only the premade parts that we can add to the **metarig** by the **Add Sample** button under the **Rigify Buttons** subpanel in **Edit Mode**; for example, the premade leg rig (**biped.leg**) has only one bone and not two for the toes, as would be necessary for the **Gidiosaurus** character, but in any case, once the final rig is generated by the script, all the necessary additions and modifications can be (quite) easily made by hand.

Obviously, to modify the generated rig, knowing how a rig works in Blender is mandatory: you can rest upon the *Building the character's Armature from scratch*, *Perfecting the Armature to also function as a rig for the Armor*, and *Building the animation controls and the Inverse Kinematic* recipes in this chapter.

In the following screenshot, you can see the **Rigify**-generated rig modified with all the additional bones for the **Armor**, **eyes**, **mouth**, and **tongue**, with the necessary added constraints and the **two toed feet** bones; the file is saved as `Gidiosaurs_rigify_02.blend`:



The final total rig

See also

- http://blenderartists.org/forum/showthread.php?200371-Rigify-Auto-rigging-system-new-and-improved

Chapter 7. Skinning the Low Resolution Mesh

In this chapter, we will be covering the following recipes:

- Parenting the Armature and Mesh using the Automatic Weights tool
- Assigning Weight Groups by hand
- Editing Weight Groups using the Weight Paint tool
- Using the Mesh Deform modifier to skin the character
- Using the Laplacian Deform modifier and Hooks

Introduction

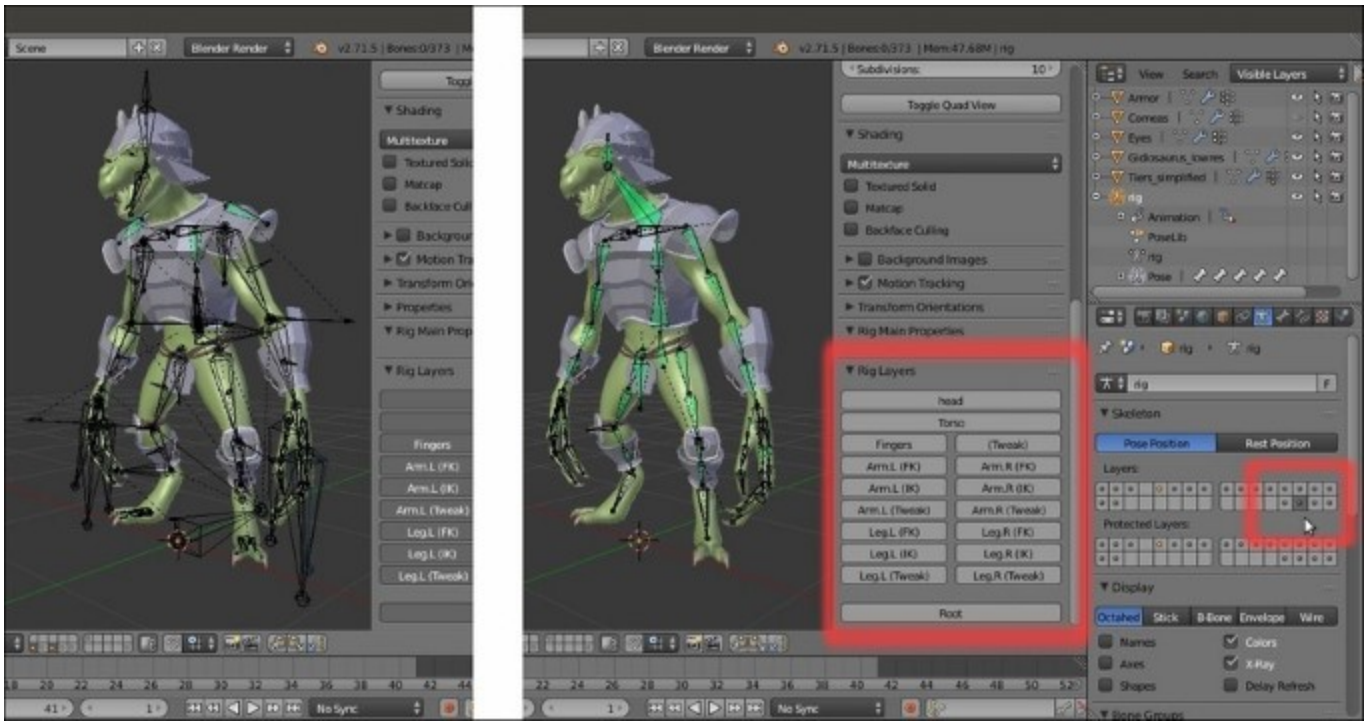
In the previous chapter, we saw the **rigging** stage, that is, how to build the character's rig (which in Blender is called an **Armature**) that will be used to deform the mesh for animations. In this chapter, instead, we are going to see quicker and more effective ways to do the **skinning** that is a necessary step to bind the bones of the **Armature** to the mesh's vertices so that they can be deformed.

To allow an **Armature** to deform a **Mesh**, they must be parented with some kind of relation; in Blender, usually you must select the **Mesh** and then *Shift* select the **Armature** and press *Ctrl + P* to parent them with different options.

This automatically makes the **Mesh** object a child of the **Armature** object and assigns the **Armature** modifier to the **Mesh**. In fact, the parenting would not be strictly necessary; it would be enough to assign an **Armature** modifier to the mesh and manually select the rig as a deforming object, but it's a good habit to use the *Ctrl + P* parenting to have the rig as a parent of the mesh, also in **Object Mode**. This way, whenever you move the **Armature** in **Object Mode**, the mesh will follow it automatically.

For the examples in these recipes, to skin the **Armature** to the **Gidiosaurus** mesh, we are going to use the final version of the rig we have built with our hands: the one saved as `Gidiosaurus_rig_from_scratch_02.blend`.

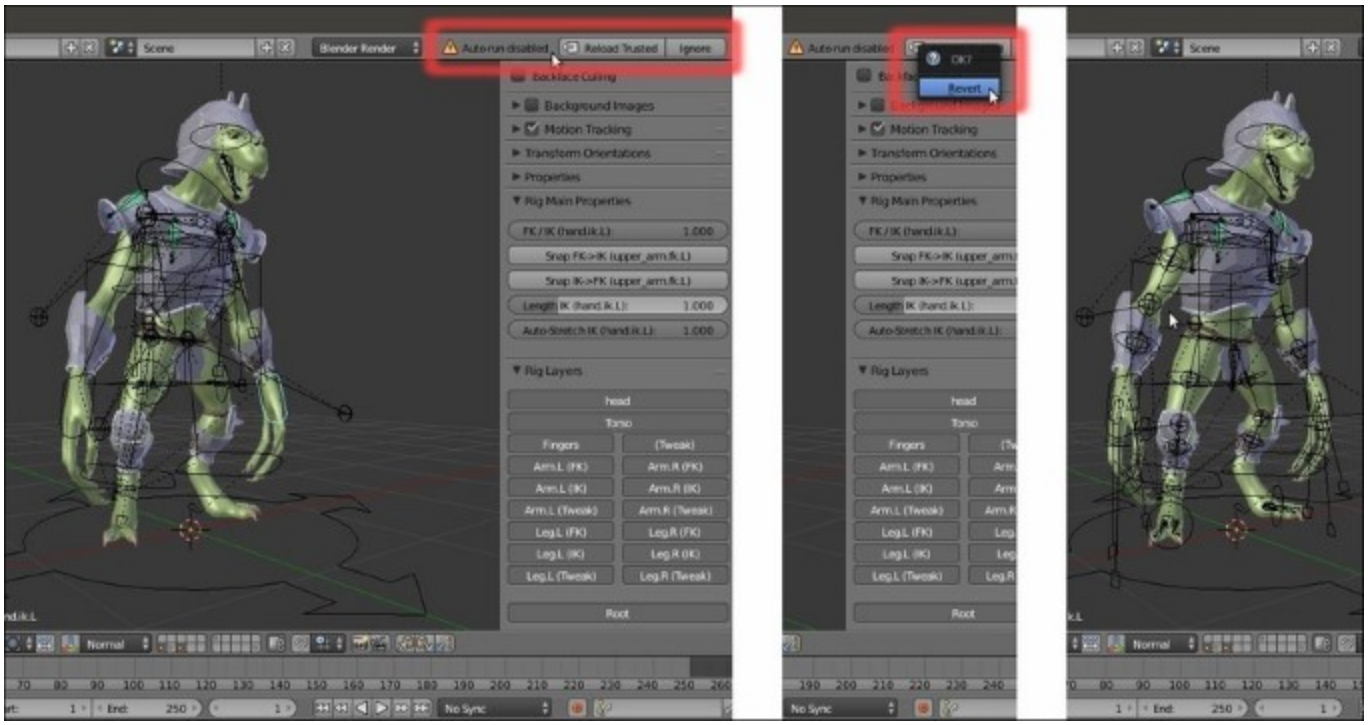
Anyway, if you want to put this to practice, in this chapter, with a more complex and complete **Rigify** armature (`Gidiosaurus_rigify_02.blend`), the procedure is exactly the same. In this case, even if not strictly necessary, remember that you can enable the **30th Armature layer** (in total there are **32**) to show the deforming bones; instead, disable the visibility of all the other bone layers also by the Python button interface in the **Rig Layers** subpanel under the 3D window **Properties** side panel:



The Rig Layers panel in the N Properties sidepanel and the Armature bone layers button in the Skeleton subpanel under the main Properties panel

Remember to check in your **User Preferences** panel (press *Ctrl* + *Alt* + *U* to call it) if you have, under the **File** tab, the **Auto Run Python Scripts** item enabled; otherwise, the rig based on Python scripts or expressions (like the rigs obtained through the **Rigify** add-on) won't work properly.

In this case, Blender will warn you through an **Auto-run disabled** message visible in the top main header; it's enough to click the **Reload Trusted** button to the right and then confirm by clicking on the **Revert** item in the pop-up menu that appears, to reload the `.blend` file with the scripts enabled and to have everything working as expected:



To the left, you can see several bones apparently missing in the rig because it is wrongly oriented, and the "Auto-run disabled" warning in the top main header; to the right, you can see the restored rig

Parenting the Armature and Mesh using the Automatic Weights tool

In this recipe, we are going to see one of the more commonly used parenting options: the handy **Automatic Weights** tool.

Getting ready

Start Blender and open the `Gidiosaurus_rig_from_scratch_02.blend` file.

1. Select the **Armature** item in the **Outliner** and press *Ctrl* + *Tab* to go out of **Pose Mode** and enter **Object Mode**.
2. Go to the **Armature** window under the **Properties** sidepanel to switch the **Display** mode from **Wire** to **Octahedral** and deselect the **Shapes** item.
3. Enable the third **Armature** layer by clicking on the **3rd** button under the **Skeleton** subpanel.
4. Disable the **13th** scene layer to hide the **Armor**.
5. Go in to **Edit Mode** and *Shift* multi-select the **MAIN** bone, the **pole** bones and the **ctrl** bones; in short, all the bones that don't have to deform anything, but are used to control the rig. Press *Shift* + *W* and in the **Toggle Bone Options** pop-up panel, select the **Deform** item to disable it for all of them at once:

Toggling the Deform item for all the selected bones at once

6. Now, deselect everything and select all the bones that, in the previous chapter, we had added specially to rig the **Armor** object, using the **Armature Layers** buttons: the **armor_ctrl** bone,

groinguard, **vanbrace.L** and **.R**, **greaves.L** and **.R**, **kneeguard.L** and **.R**, **spaulder.L** and **.R**; again, press *Shift* + *W* | **Deform** to disable the option.

Repeating for the Armor object bones

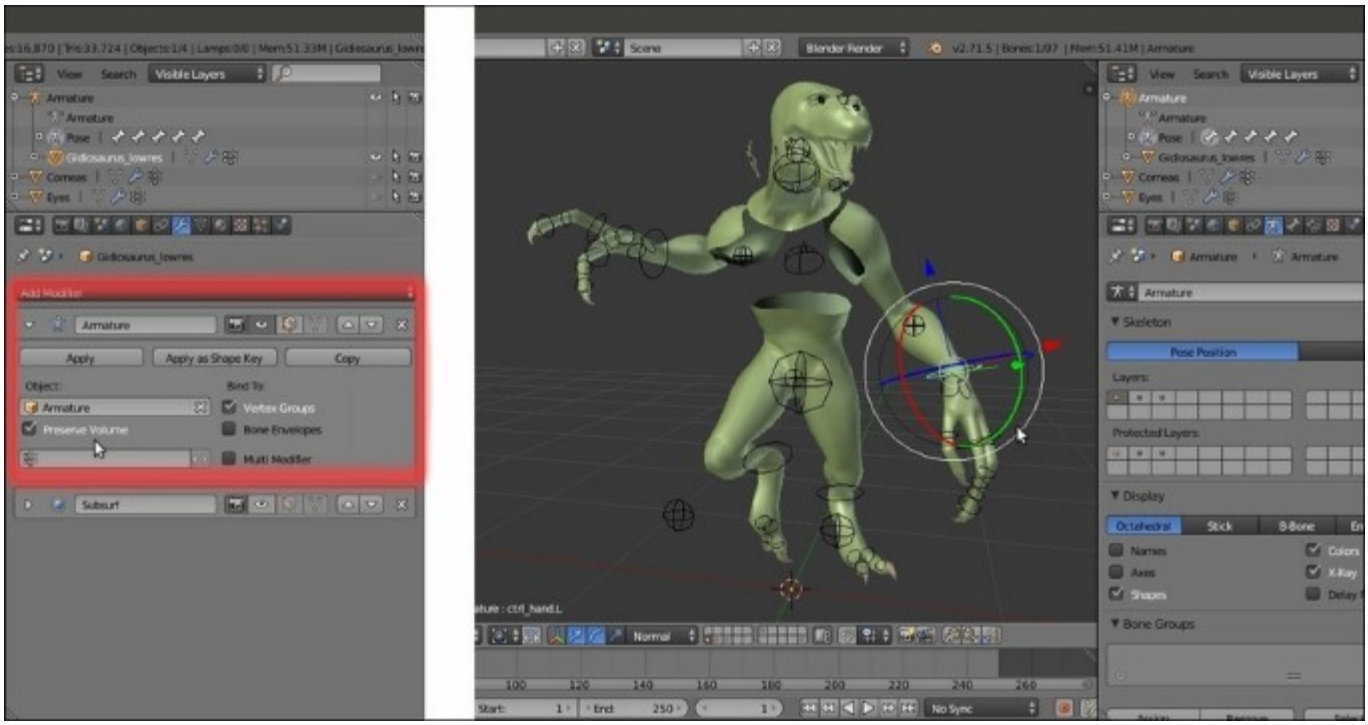
7. Don't *deselect* the **Armor** bones, simply switch from **Edit Mode** to **Object Mode**.

How to do it…

1. Select the **Gidiosaurus_lowres** object and then *Shift*-select the **Armature**, and press *Ctrl* + *P*; in the **Set Parent To** pop-up menu; select the **With Automatic Weights** item:

The Set Parent To pop-up menu

2. Reselect the **Armature**, go in to **Edit Mode**, and press *Shift + W* | **Deform** to re-enable the item for the still-selected **Armor** bones; then, go out of **Edit Mode**.
3. Now, reselect the **Gidiosaurus** object; go to the **Object Modifiers** window, move the newly created **Armature** modifier upwards in the stack, and enable the **Preserve Volume** item.
4. Disable the *Display modifier in viewport* button (the one with the eye icon) of the **Subdivision Surface** modifier to speed up the 3D viewport (sadly, Blender still has very bad real-time viewport performances, so even if you have a lot of RAM and a powerful workstation, it's wise to stay as light as you can).
5. Select the **Armature** and under the **Object Data** window, re-enable the **Shapes** item and hide the second and the third **Armature Layer**; press *Ctrl + Tab* to go in **Pose Mode** and try to select some of the control bones to move or rotate them and so control how they are deforming the mesh; temporarily, hide the **Eyes** object in the **Outliner**.



The Armature modifier subpanel and the posed mesh

To rotate the bones on their local axis, enable the **3D manipulator widget** in the 3D view toolbar (*Ctrl* + Spacebar), click on the **Rotate** icon, and set **Transformation Orientation** to **Normal**.

6. Save the file as `Gidiosaurus_autoweights.blend`.

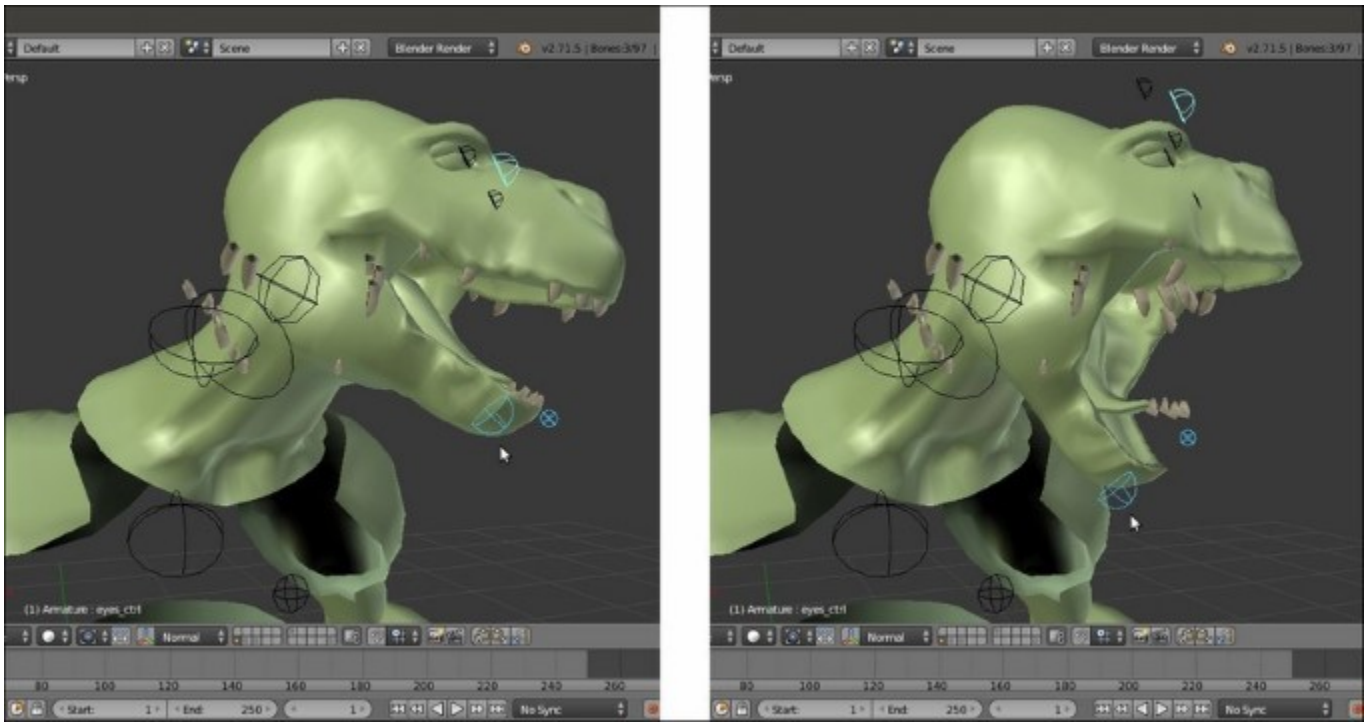
How it works…

The **Automatic Weights** tool creates the necessary **Vertex Groups** based only on the bones that have been set as **Deformers** in the subpanel under the **Bone** window. It then assigns weights inside a range from **0.000** to **1.000** to the vertices contained in these vertex groups, calculating their proximity to the bone with the same name. In short, the **arm.L** bone will deform only the vertices inside the **arm.L** vertex group, and with an intensity based on their weights.

Because we used the **Automatic Weights** tool to skin only the sole **Gidiosaurus** mesh (leaving the skinning of other objects such as the **Eyes** or the **Armor** for the next recipe and method), before the parenting we had to check for any bone erroneously left as a deformer (that is, one of the several control bones in the previous chapter), but, especially we had to temporarily disable the **Deform** item for the **Armor** bones, which otherwise would have also been evaluated by the tool for the **body**.

In most cases, the **Automatic Weights** tool can give quite good results without the need of further tweaking; however in some areas, for example the **head**, where the **head** bone length doesn't fully fit the upper part of the shape of the mesh and where there are also other deforming bones, it can easily fail.

Look at the following screenshot; at first, by rotating the **head** control, the only issue seems to be some of the **teeth** left out from the calculations but then, simply by moving the controls for the **eyes**, **tongue**, and **jaw**, it becomes evident that the tool assigned several vertices to the wrong bones merely based on their proximity to that part of the mesh:



The failure of the Automatic Weights tool parenting

Although at first sight this can appear to be a total mess, it's usually less complex to fix than one might think.

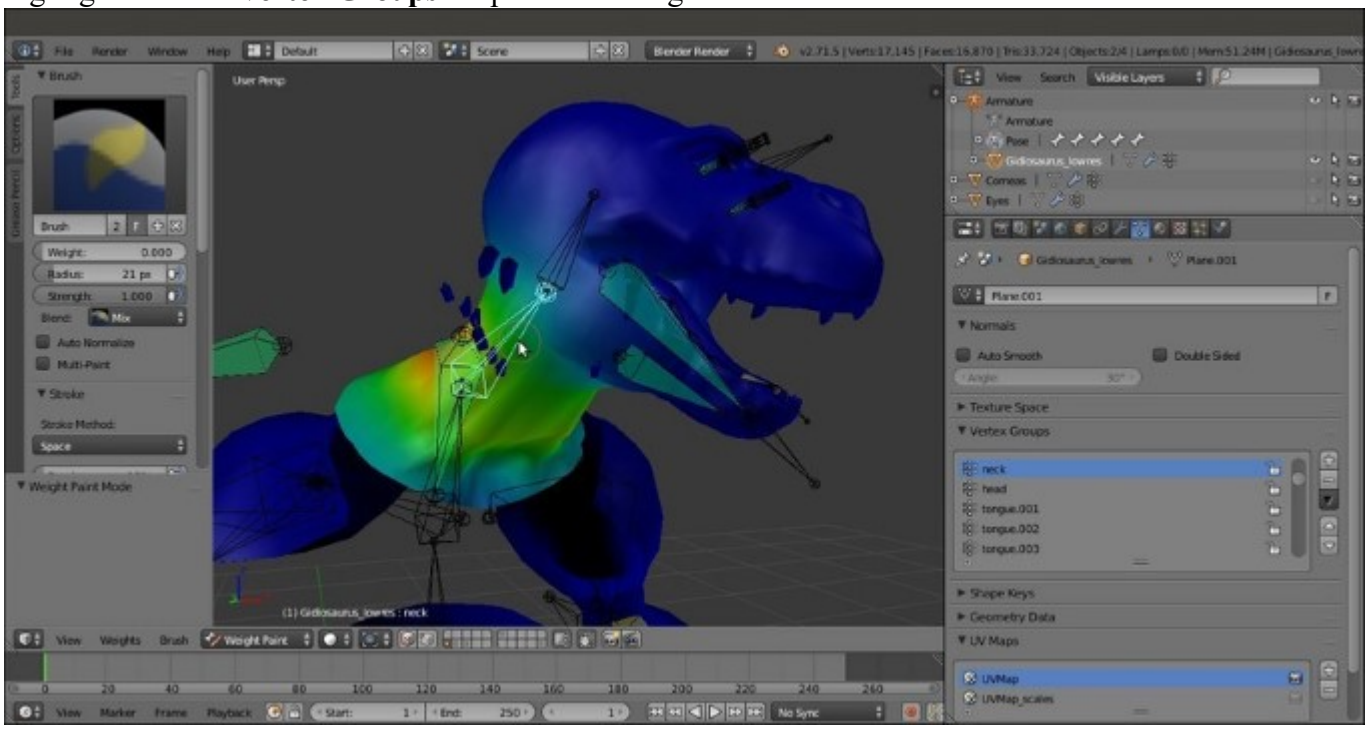
For the moment, by selecting the **Gidiosaurus** mesh and pressing *Ctrl* + *Tab*, we go in to **Weight Paint** mode, and by right-clicking on a bone (the **Armature** is still in **Pose Mode**), the weights of the corresponding vertex group became visible as colored areas on the mesh; the color **red** corresponds to a weight value of **1.000** and **blue** to a value of **0.000**, with all the intermediate hues corresponding to the intermediate values. For example, **green** = **0.500** and so on.

There's more…

Let's see all this step by step:

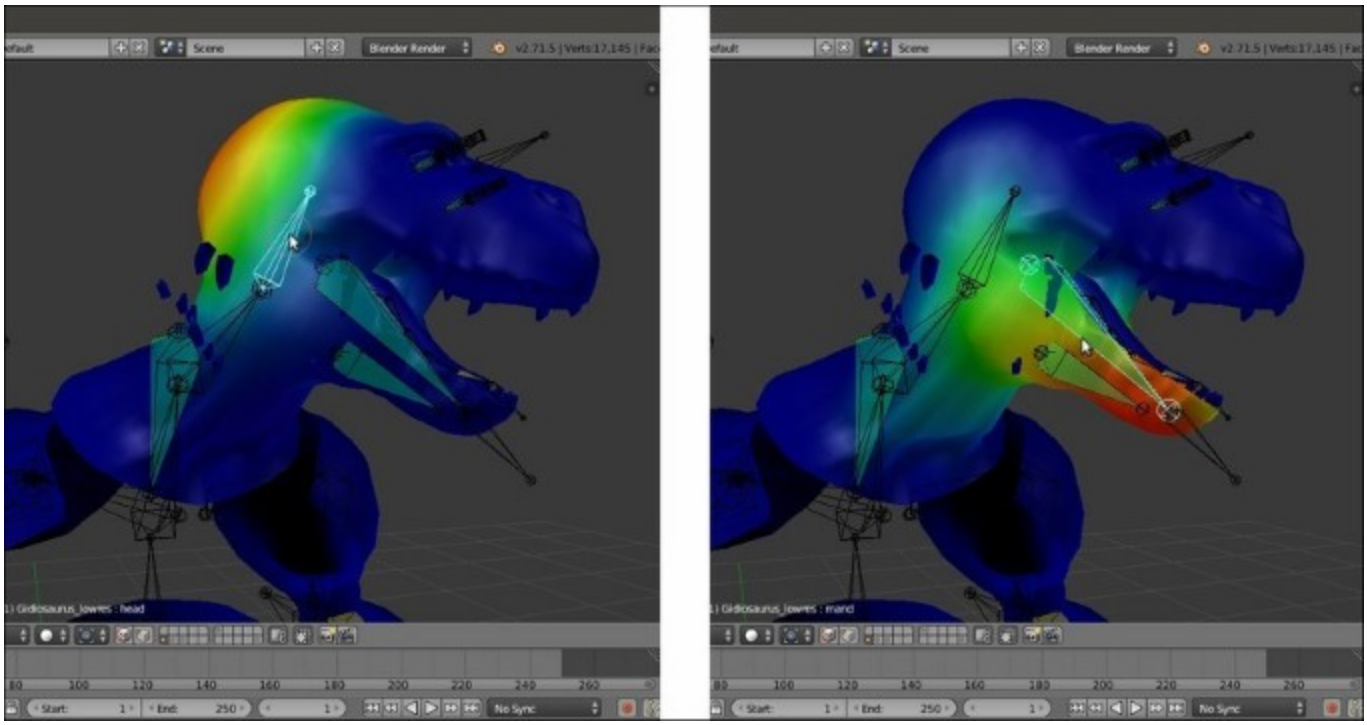
1. Select the **Armature** and, while still in **Pose Mode**, enable the visibility of the **third Armature layer** (and therefore of all the deforming bones) and then disable the **Shapes** item again.
2. Select the **Gidiosaurus** mesh and by pressing *Ctrl* + *Tab*, enter **Weight Paint** mode (or switch to it by the *object interaction mode* button on the toolbar of the 3D view).

3. Click on any one of the deforming bones, for example the **neck** bone, and notice that while the weights appear on the mesh surface, at the same time the corresponding vertex group is highlighted in the **Vertex Groups** subpanel to the right:



Visualizing the vertex groups on the mesh

By clicking on the **head** bone and/or the **mand** bone, the reasons for the bad deformations are immediately clear: the **Automatic Weights** tool didn't assign the whole upper part of the **head** of the character to the sole **head** vertex group (and therefore to the bone with the same name) with a full value of **1.000**; instead, it assigned part of the **head** mesh to the **eyes** bones, other parts to the **tongue.005** bone, some to the **mand** bone, and so on.



Different weights of the vertex groups associated with different bones

Obviously, this isn't the tool's fault, but it is an *unavoidable issue* due to the particular arrangement of the bones in the **head** area and can be quite easily fixed anyway; we'll see how in the next and the *Editing the Weight Groups by the Weight Paint tool* recipe.

See also

- http://www.blender.org/manual/rigging/skinning/obdata.html

Assigning Weight Groups by hand

This technique is the oldest way to assign weights to vertices groups in Blender. Although now there are quicker ways to do the same thing, in some cases it's still one of the best approaches, which can reveal itself to be quite useful mainly because you can precisely select individual or edge-loops of vertices to be weighted inside a group.

Getting ready

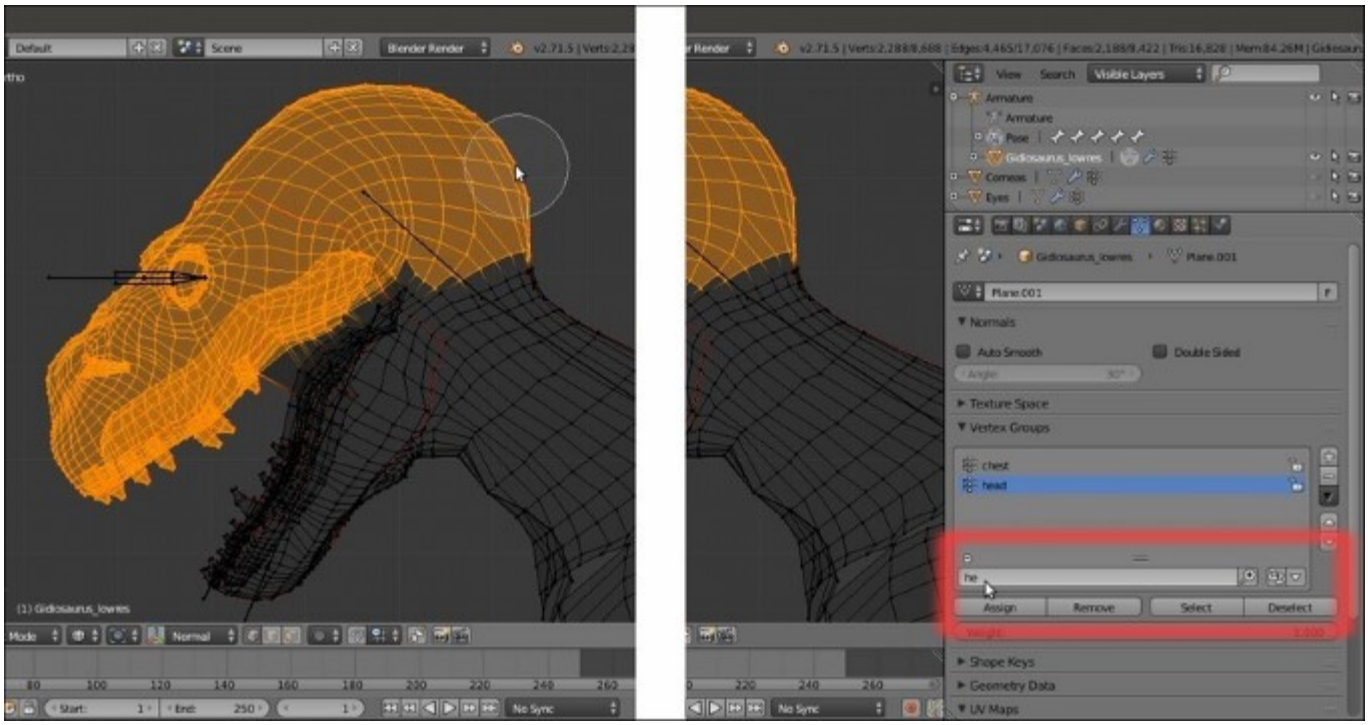
Open the `Gidiosaurus_autoweights.blend` file we saved in the previous recipe.

1. If necessary, press *Ctrl* + *Tab* to go out of **Weight Paint** mode.
2. Select the **Armature** (which should still be in **Pose Mode**), press the *A* key twice to deselect-select all the bones, and press *Alt* + *R* and *Alt* + *G* to clear any rotation or position and restore the default pose.
3. Press the *3* key on the numpad to go in to **Side** view; if necessary, the *5* key on the numpad to go in to **Ortho** view and the *Z* key to go in to **Wireframe** viewport shading mode.
4. Select the **Armature** and disable the **Shapes** item; switch the draw mode of the bones to **Stick** and enable the third **Armature** layer to show the deforming bones.
5. Save the file as `Gidiosaurus_skinning_01.blend`.

How to do it…

First, we are going to use this technique to fix the **head** deformation as follows:

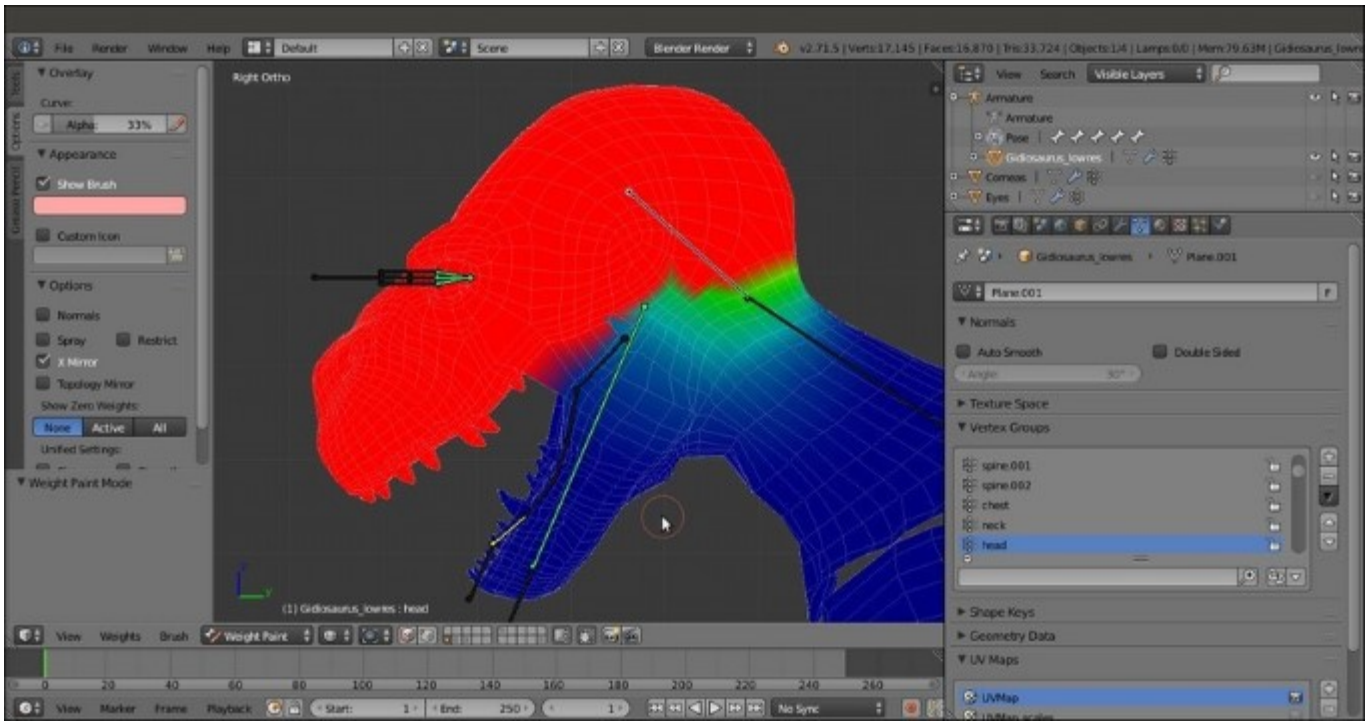
1. Press *Shift* + *B* to draw a box around the **head** of the **Gidiosaurus** mesh and automatically zoom to it. Select the mesh and enter **Edit Mode**.
2. Press the *C* key, through which the mouse cursor turns into a circle whose diameter can be set by scrolling the mouse wheel.
3. Start to *paint-select* the vertices you want to add to the vertex group; in this case, we must add the whole **upper head** to the **head** vertex group and also include the **upper teeth** that were missing in the group.
4. Be sure that the **head** vertex group is the selected one in the **Vertex Groups** subpanel under the **Object Data** window to the right, and that the **Weight** slider is set to **1.000**; then, click on the **Assign** button.
5. To quickly find a required vertex group, instead of slowly scrolling the list, just click on the grayed out little **+** icon at the bottom of the **Vertex Groups** window (just above the **Assign** button) to expand a blank search field and then write a few letters of the group's name followed by the *Enter* key:



The vertex group names search function

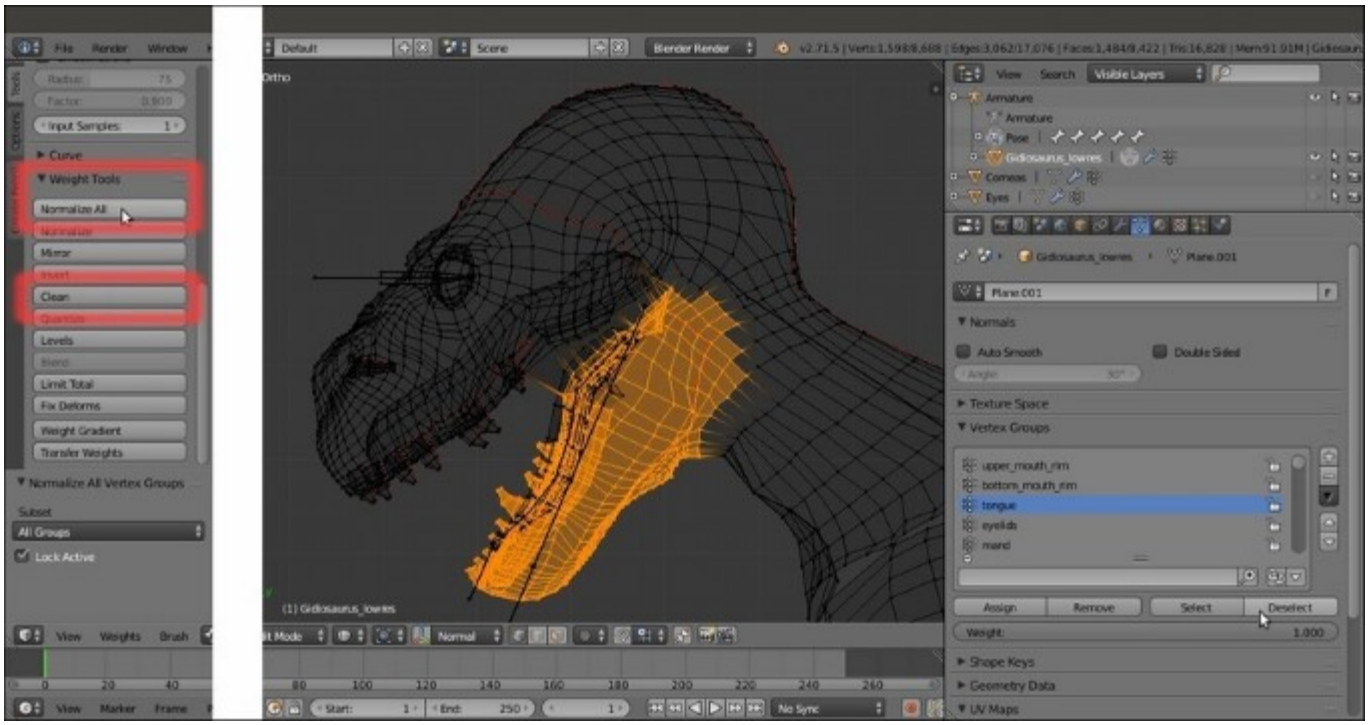
To get the complete list of vertex groups' names back, just erase the letters you wrote in the field and press *Enter*.

6. Now, switch from **Edit Mode** to **Weight Paint** mode again, where the **head** vertex group colors show a lot different than before:



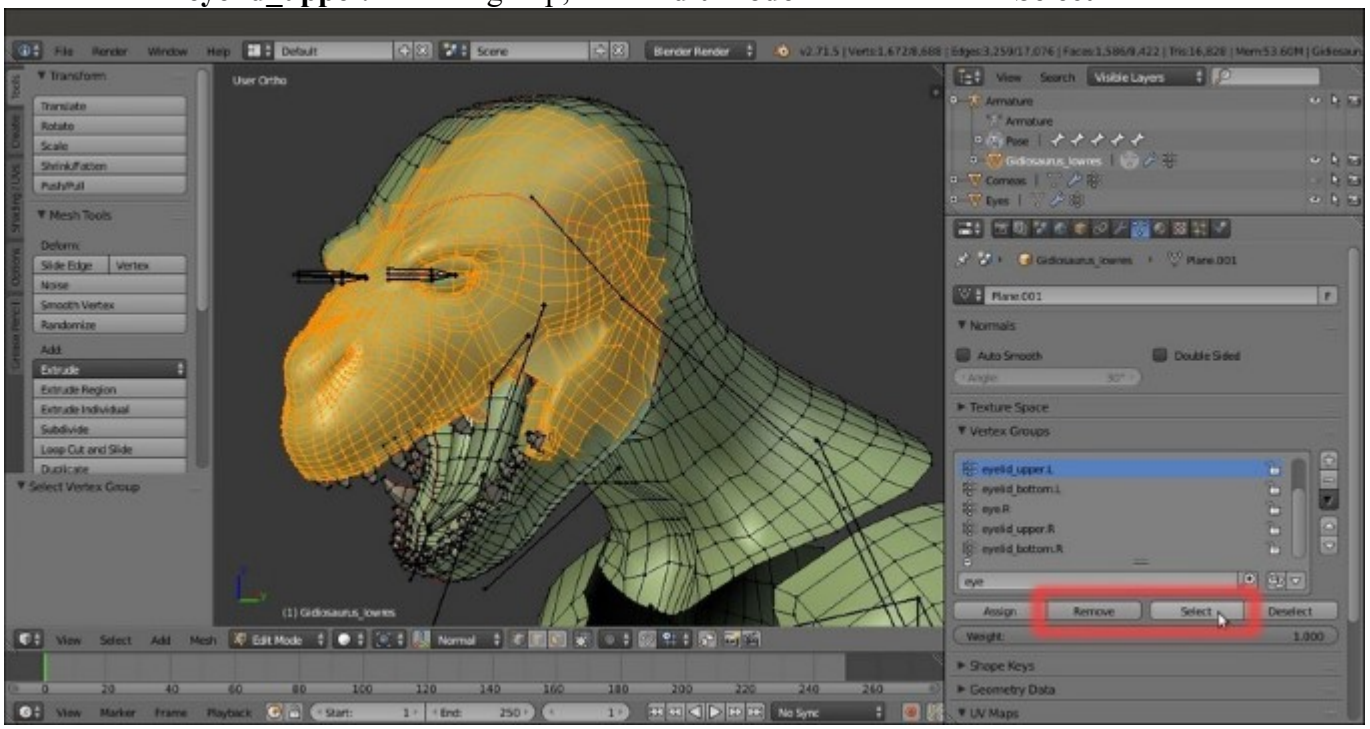
The modified "head" vertex group

7. Now, while still in **Weight Paint** mode, go to the **Tools** tab under the **Tool Shelf** to the left of the screen and, in the **Weight Tools** subpanel, first click on the **Normalize All** button and then on the **Clean** button.
8. Select the **mand** bone, go in to **Edit Mode**, and press *A* to deselect the vertices of the **head** vertex group. Select all the vertices of the **jaw**, including the **bottom teeth**; then go to the **Vertex Groups** subpanel to the right and deselect the **tongue** group by clicking on the, yes, **Deselect** button (we created the **tongue** vertex group chapters ago, during the modeling stage; otherwise, just deselect the tongue's edge-loops manually).
9. Find and select the **mand** group and click on the **Assign** button.
10. Go again in to **Weight Paint** mode and click on the **Normalize All** and **Clean** buttons under the **Weight Tools** subpanel:



The Weight Tools subpanel and the "mand" vertex group

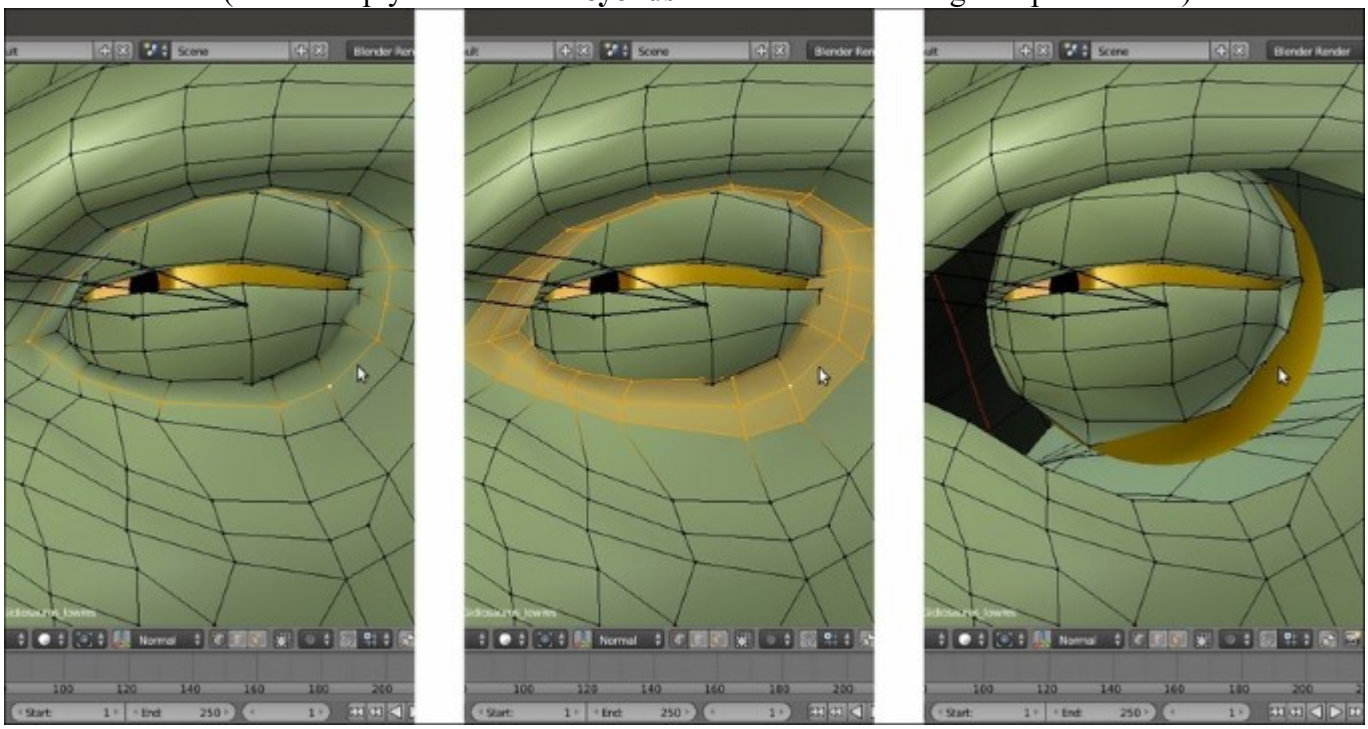
11. Now, go out of **Weight Paint** mode, select the mesh and, in the **Vertex Groups** subpanel, search for the **eyelid_upper.L** vertex group; enter **Edit Mode** and click on the **Select** button:



The selected eyelid_upper.L vertex group

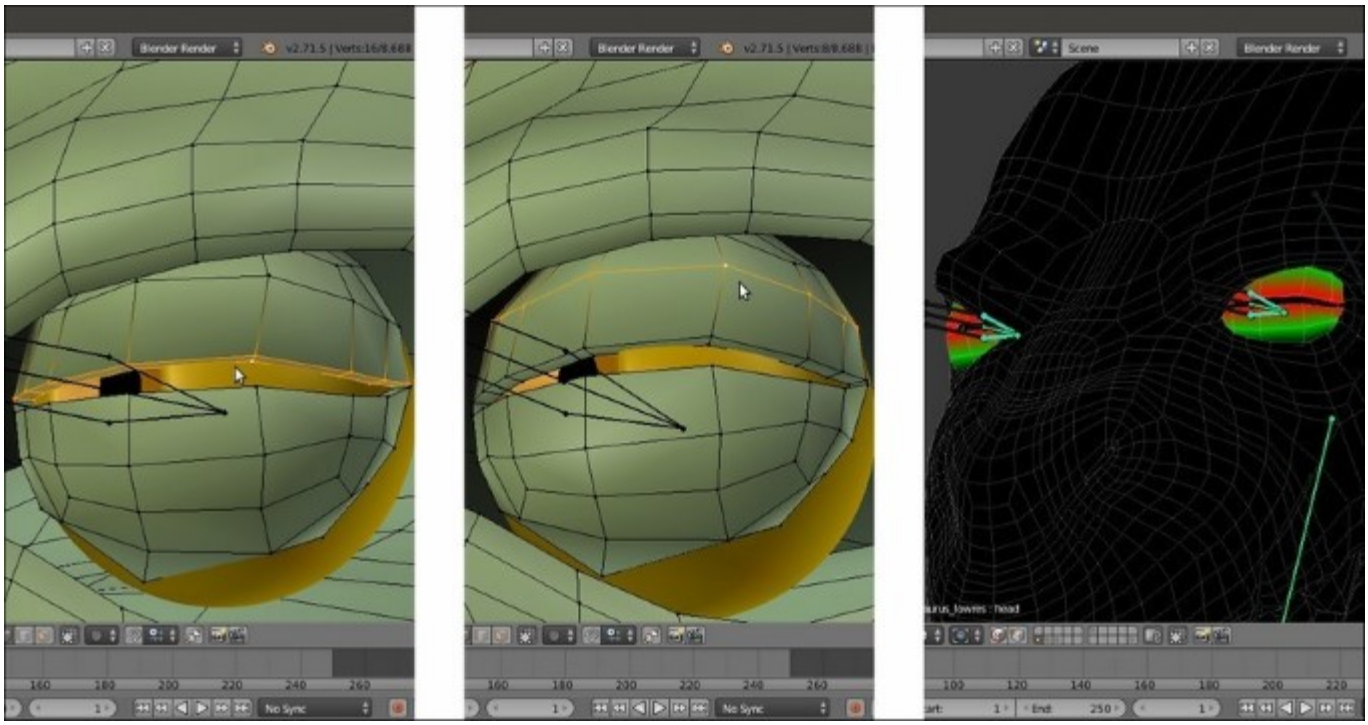
We must get rid of all these vertices erroneously assigned to the vertex group by the **Automatic Weights** tool.

12. Click on the **Remove** button and then press the *A* key to deselect everything.
13. Repeat this for the **eyelid_bottom.L**, **eyelid_upper.R**, **eyelid_bottom.R**, and also for the **eye.L** and **eye.R** vertex groups.
14. Zoom to the **eyes** area. Select an edge-loop (*Alt* + right-click) around the left **eyelids** and then press *Ctrl* and the + key on the numpad to extend the selection; press the *H* key to hide the selected vertices (this is simply to isolate the **eyelids** vertices for easier edge-loops selection):



Isolating the eyelids vertices

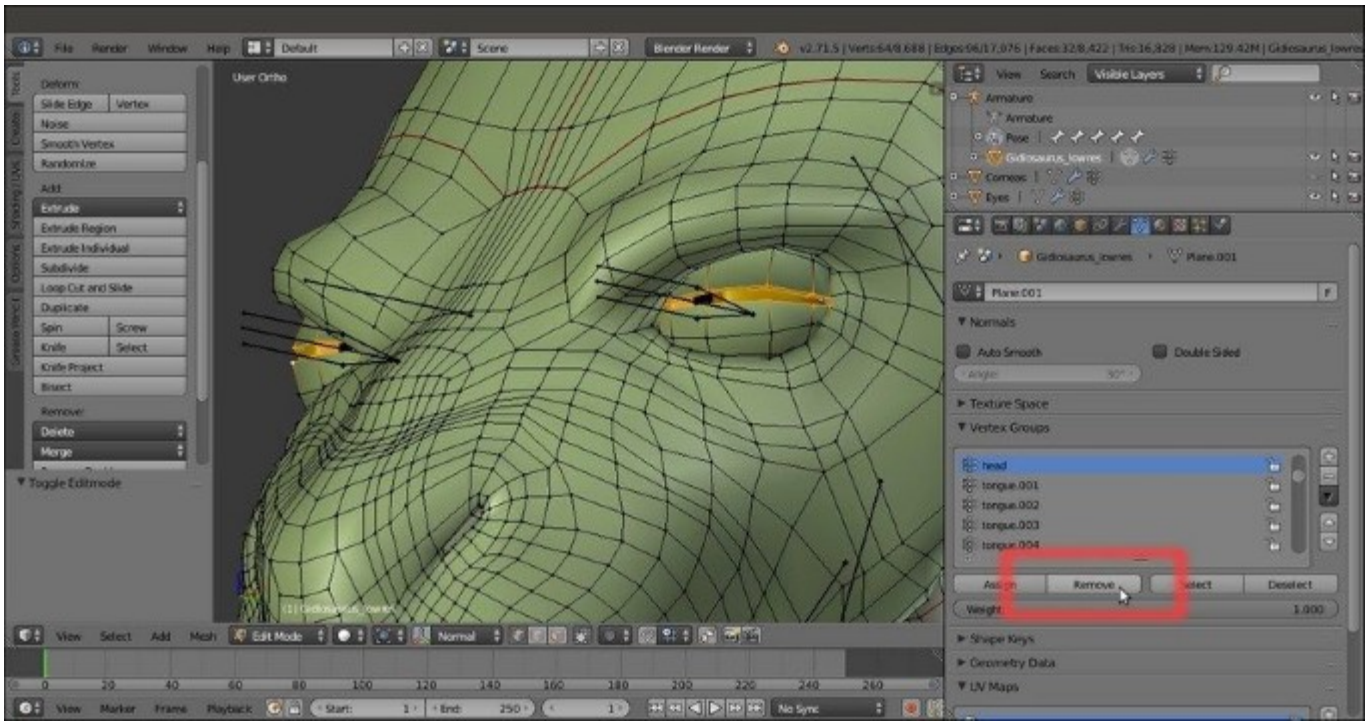
15. Select the border upper edge-loops and assign them to the **eyelid_upper.L** vertex group with a weight of **1.000**; select the second upper edge-loop and again assign it to the **eyelid_upper.L** vertex group, but with a weight of **0.500** (see the following screenshot).
16. Do the same for **eyelid_bottom.L**:



The visualization of the eyelids vertex group with different weights

In the preceding screenshot, to the right, you can see the weights of **eyelid_upper** and **eyelid_bottom** vertex groups on both sides, made visible at the same time by the **Multi-Paint** item enabled in the **Brush** subpanel under the **Tool Shelf**; here it is used only for visualization purposes.

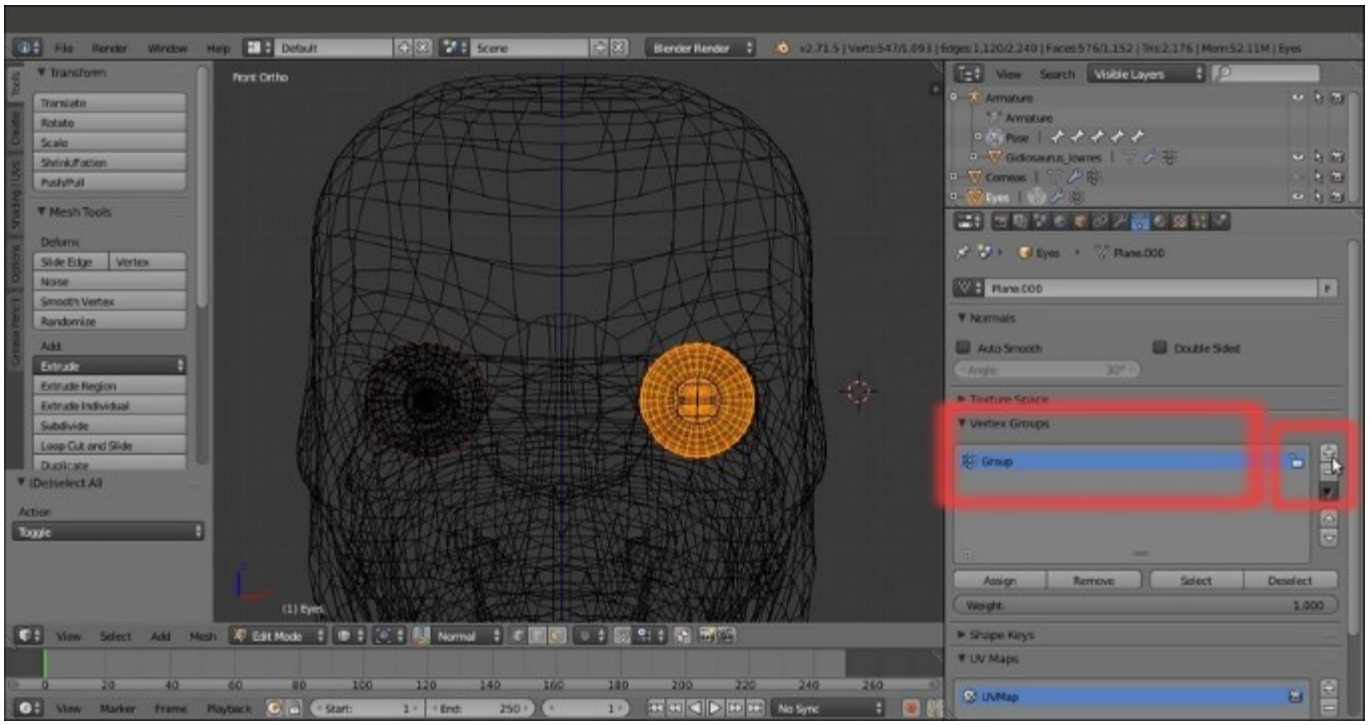
17. Repeat the procedure for the **eyelids** on the right side (**eyelid_upper.R** and **eyelid_bottom.R**).
18. In the **Vertex Groups** subpanel, select the **head** vertex group and then select the border edge-loops of both the **left** and **right eyelids**. Click on the **Remove** button to remove those vertices from the group's evaluation:



Removing the eyelids vertices from the "head" vertex group

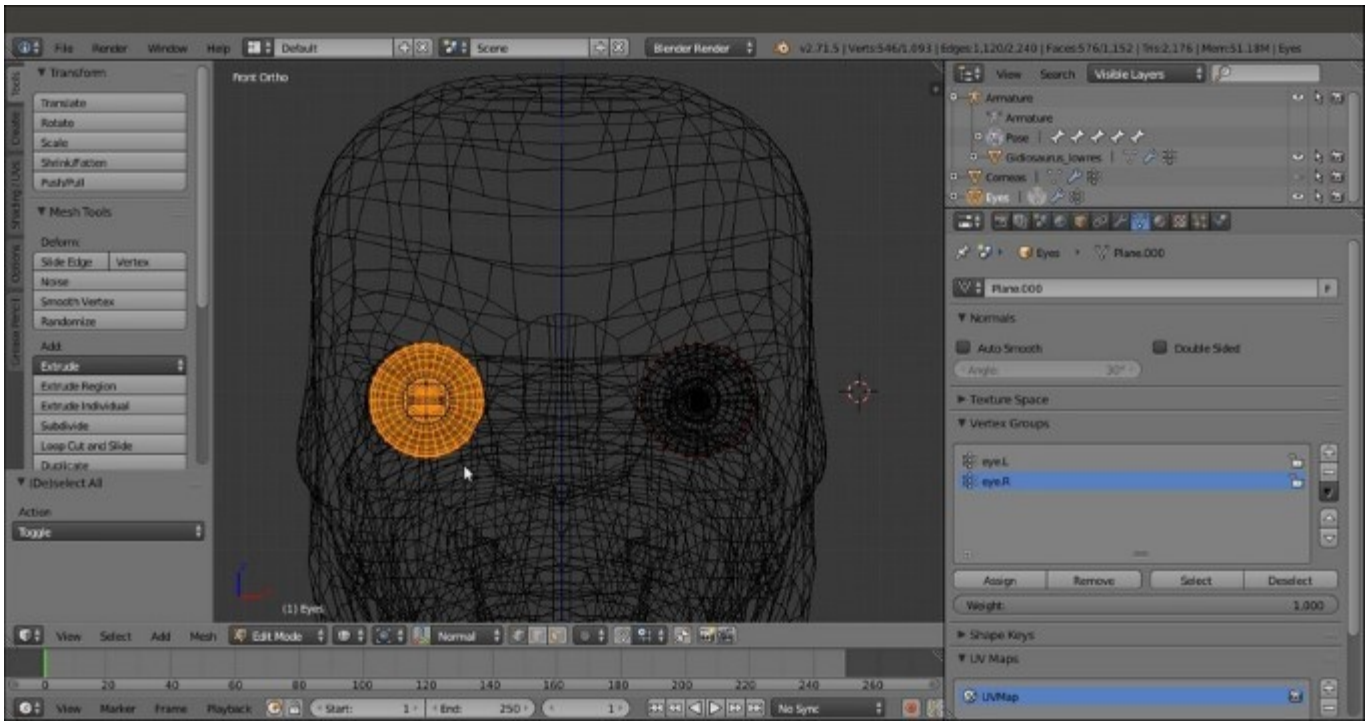
Assigning weights by hand can be a handy method also for other parts, for example, the **eyeballs**, which are separate objects from the **Gidiosaurus** mesh.

19. Go out of **Edit Mode,** and in the **Outliner** select the **Eyes** object; press *Tab* again to go in to **Edit Mode**.
20. Go in to **Front** view and box-select all the vertices of the **left eye**; then, go to the **Vertex Groups** subpanel under the **Object Data** window and click on the + icon to the right to create a new group:



Creating the vertex group for the eyeball

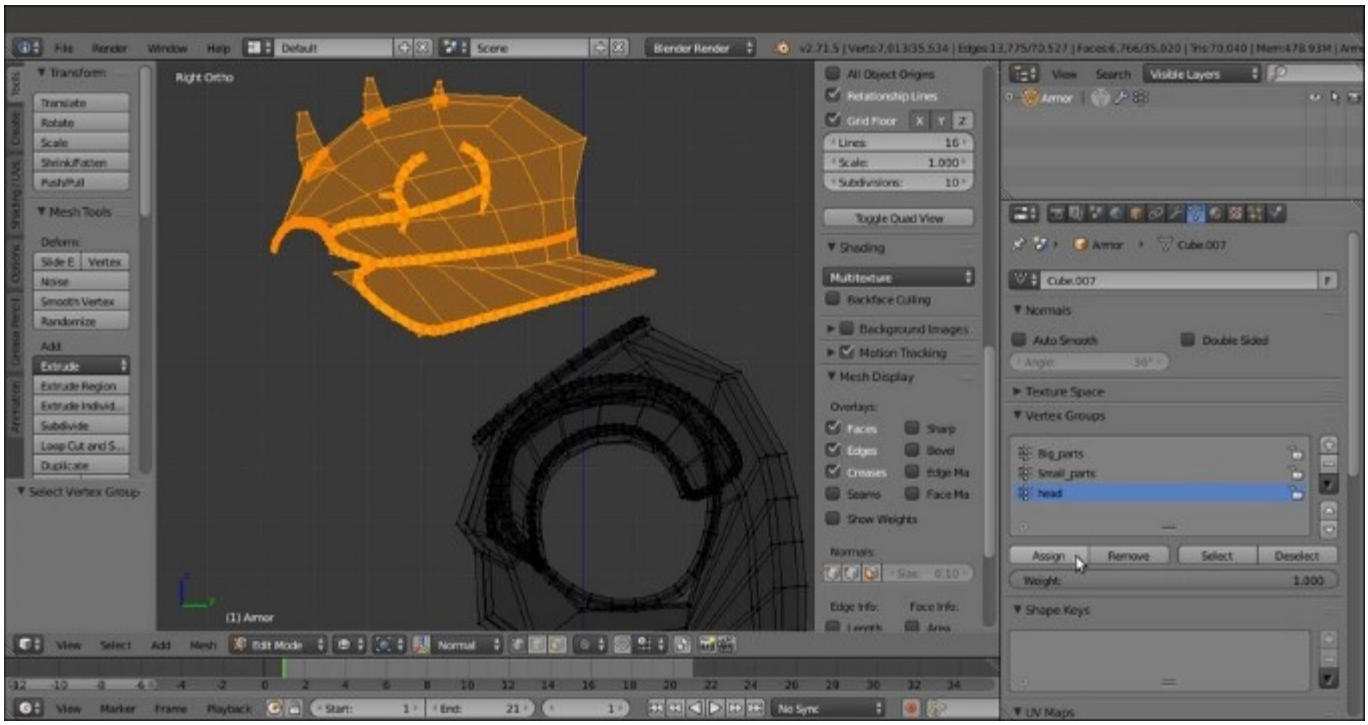
21. *Ctrl* + left-click on the name of the vertex group to rename it as **eye.L** and then click on the **Assign** button to assign all the selected vertices to the group with a value of **1.000**.
22. Deselect everything, select the vertices of the other **eye**, and create a new vertex group; rename it as **eye.R** and assign the vertices.



Creating the eye.R vertex group

23. Exit **Edit Mode** (*Tab*) and go to the **Object Modifier** window; assign an **Armature** modifier, move it upwards in the stack, and click on the **Object** field to select the **Armature** item as a deforming object.
24. Temporarily, unhide the **Corneas** object in the **Outliner** and repeat from step 19 to step 23, where we created the **eye.L** and **eye.R** vertex groups and assigned the appropriate mesh vertices and the **Armature** modifier.

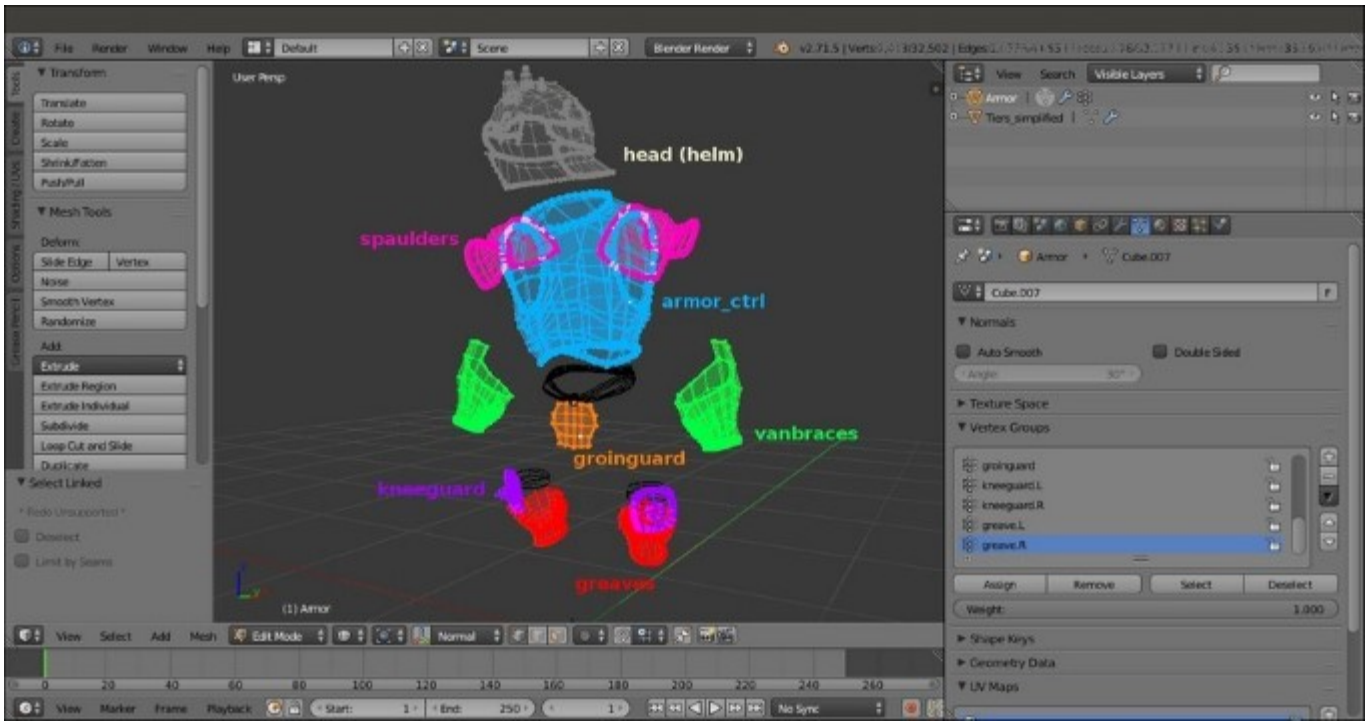
    The same process must be applied to the skinning of the **Armor** that, being a single object made of stiff elements, can be easily and ideally divided into different vertex groups; each one is skinned with the full value of **1.000**.
25. Click on the button to activate the **13th** scene layer and show the **Armor**; select it and enter **Edit Mode**.
26. Select all the vertices of the **helm**, including the **decorations**, create the **head** vertex group in the **Armor** mesh (remember that the name must be the one of the deforming bones), and click on the **Assign** button with a weight value of **1.000**:



The "head" vertex group for the Armor object

27. Repeat the operation with each part of the **Armor**, so creating, always with a weight of **1.000**, the vertex groups: **vanbrace.L** and **vanbrace.R** (covering the **forearms**), **greave.L** and **.R** (covering the **calves**), **groinguard** (the **front hips**), **kneeguard.L** and **.R**, **spaulder.L** and **.R**, and **armor_ctrl**.

    You can use the following screenshot as a guide:



The happy colorful Armor guideline

28. Go out of **Edit Mode**; then, go to the **Object Modifier** window, assign an **Armature** modifier to the **Armor**, move it upwards in the stack, and click on the **Object** field to select the **Armature** item as a deforming object.
29. Disable the *Display modifier in viewport* button (the one with the eye icon) for the **Subdivision Surface** modifier, to speed up the 3D viewport.

For the moment, ignore the **Tiers** (which have been separated by the **Armor** object and simplified by deleting several alternate edge-loops, this we'll skin in the next recipe) and *save the file*.

How it works…

The **Normalize All** button normalizes the weights of all the vertex groups so that their sum is not superior to **1.000**; because we had assigned a weight of **1.000** to the upper head vertices, the vertices in the other groups that were interfering with the **head** deformation have been automatically set to **0.000**.

The **head** group, instead, remained the same because it was locked in the **Options** bottom panel; the **Clean** button, then, took care of removing all the unwanted vertices in the active group, restricting the inclusion of its vertices only to those with a weight greater than **0.000**.

When assigning vertices to a group, the **Weight** slider under the **Vertex Groups** subpanel can obviously be set to any value between **0.000** and **1.000**, so it's also possible to select a single edge-loop or rows of vertices and assign them at different times to the same vertex groups, but with different weight values. For example, a central edge-loop of vertices with the weight of **1.000** can be surrounded by external

edge-loops with weight values of **0.750**, **0.500**, **0.250**, and so on. This is what we have done for the **eyelids** after the cleaning of the **eye sockets** area, thanks to the **Select**, **Deselect** and **Remove** buttons. Be aware that the same result can be obtained by painting and/or blurring the weights on the mesh, but we'll see this in the next recipe.

See also

- http://www.blender.org/manual/modeling/meshes/vertex_groups/index.html

Editing Weight Groups using the Weight Paint tool

Both the **Automatic Weights** parenting as well as the **Weight Groups** created and assigned by hand must, at a certain point, inevitably be edited for several reasons. As we have already seen, the parenting tool didn't do a perfect job, or maybe the transition between different weights is too sharp and must be blurred to smoothly deform the mesh. In any case, the ideal tool for this editing work is the **Weight Paint** tool.

Getting ready

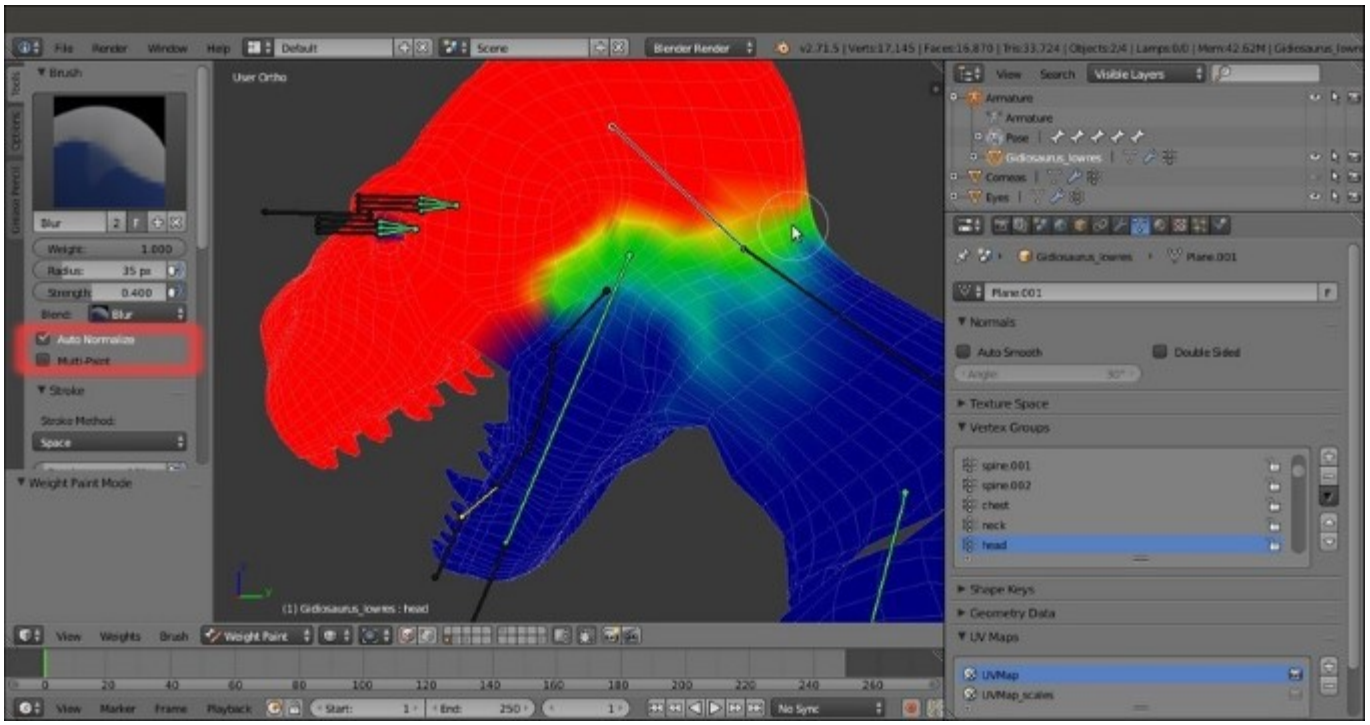
As usual, let's first prepare the scene to work on:

1. Open the `Gidiosaurus_skinning_01.blend` file and hide the **13th** scene layer.
2. Enable the 3rd **Armature** layer and then deselect the **Shapes** item.
3. Press the *3* key on the numpad to go in to **Side** view; if necessary, press the *5* key on the numpad to go in to **Ortho** view and the *Z* key to go in to **Wireframe** viewport shading mode.
4. Save the file as `Gidiosaurus_skinning_02.blend`.

How to do it…

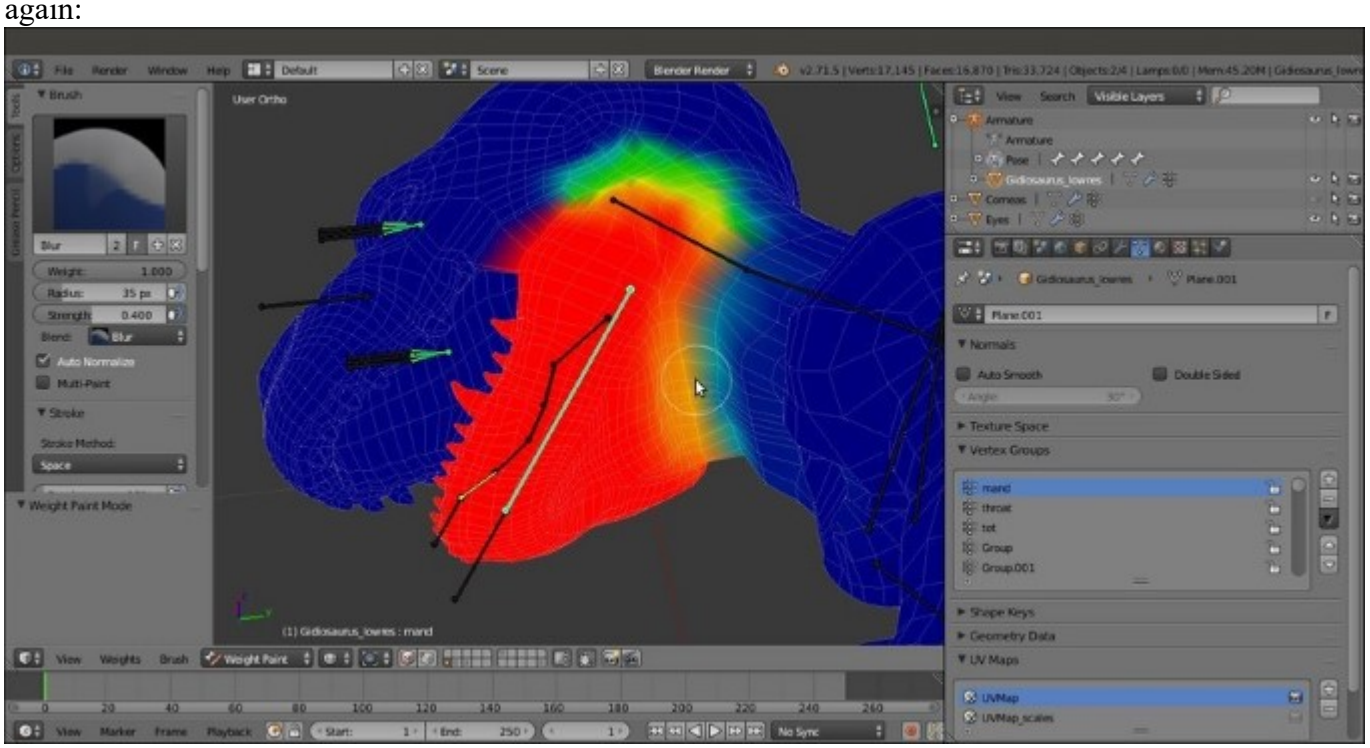
Also, now let's start with the **Weight Paint** tool itself:

1. Select the **Gidiosaurus** mesh and go in to **Weight Paint** mode; the tabs under the **Tool Shelf** on the left-hand side of the 3D window (press the *T* key in case they are not already present), change to show the **Weight Paint** tools.
2. In the viewport, right-click on the **head** bone to show the **head** vertex group on the mesh's surface.
3. Go to the **Tool Shelf** and click on the **Options** tab to verify that the **X Mirror** item in the **Options** subpanel is activated. Then, go back to the **Tools** tab and click on the big **Brush** window at the top to select a **Blur** brush; set **Weight** to **1.000** and **Strength** to **0.400**.
4. Select the **Auto Normalize** item at the bottom of the **Brush** subpanel.
5. Start to paint on the borderline of the vertex group, blurring the separation between the red and blue colors and trying to obtain, in general, a transition as smooth as possible:



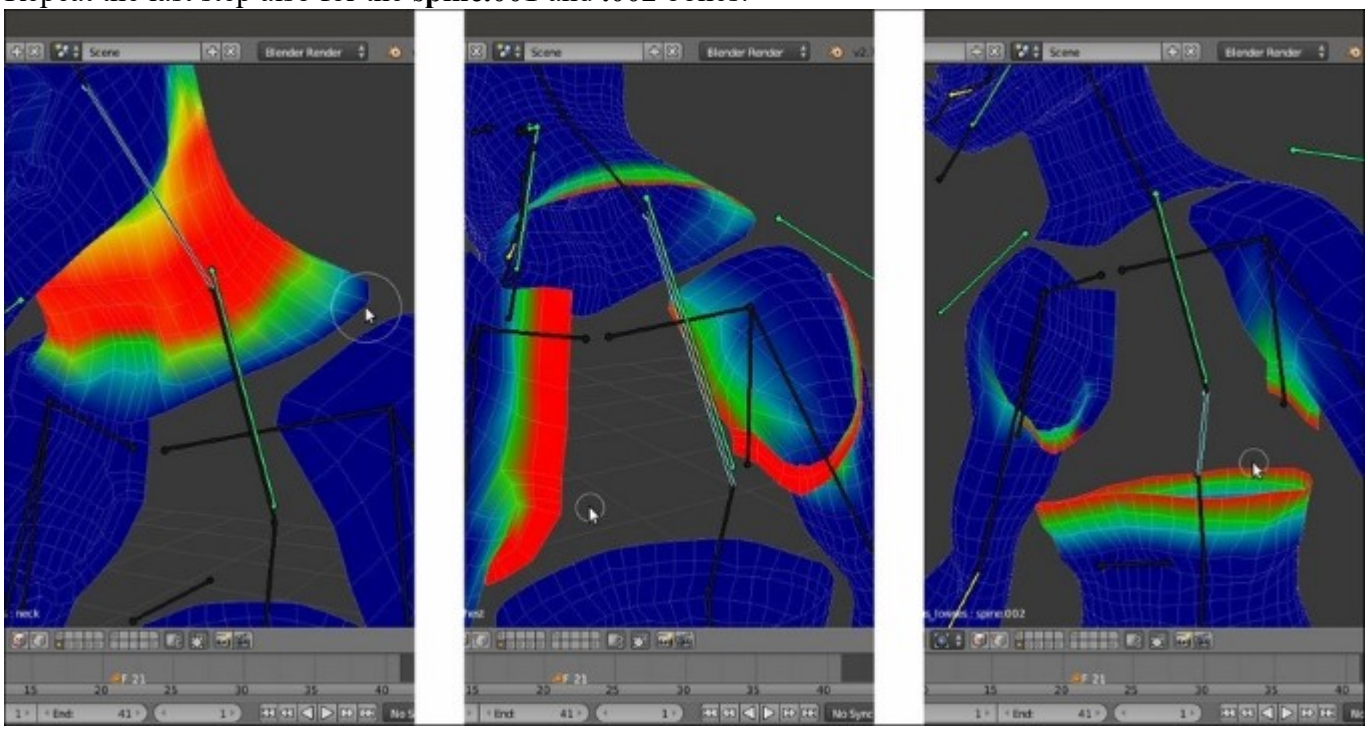
Blurring the edges of the vertex group

6. Switch to the **mand** vertex group by selecting the corresponding bone and smooth the transition again:



Smoothing the transition of the "mand" vertex group

7. If you need to reduce the weight of a vertex, switch the **Blur** brush with a **Subtract** one, and with a low **Strength** (**0.100** or even less) paint on it. Then, if necessary, blur the area again.
8. Alternatively, instead of using a **Subtract** brush, you can paint on the mesh with a **Mix** brush set with **Strength** = **1.000** and **Weight** = **0.000**.
9. Select the **neck** bone and reduce the weight of the vertices at the **neck** edges to **0.000**.
10. Select the **chest** bone and paint the vertices at the **chest** edges to **1.000**.
11. Repeat the last step also for the **spine.001** and **.002** bones:

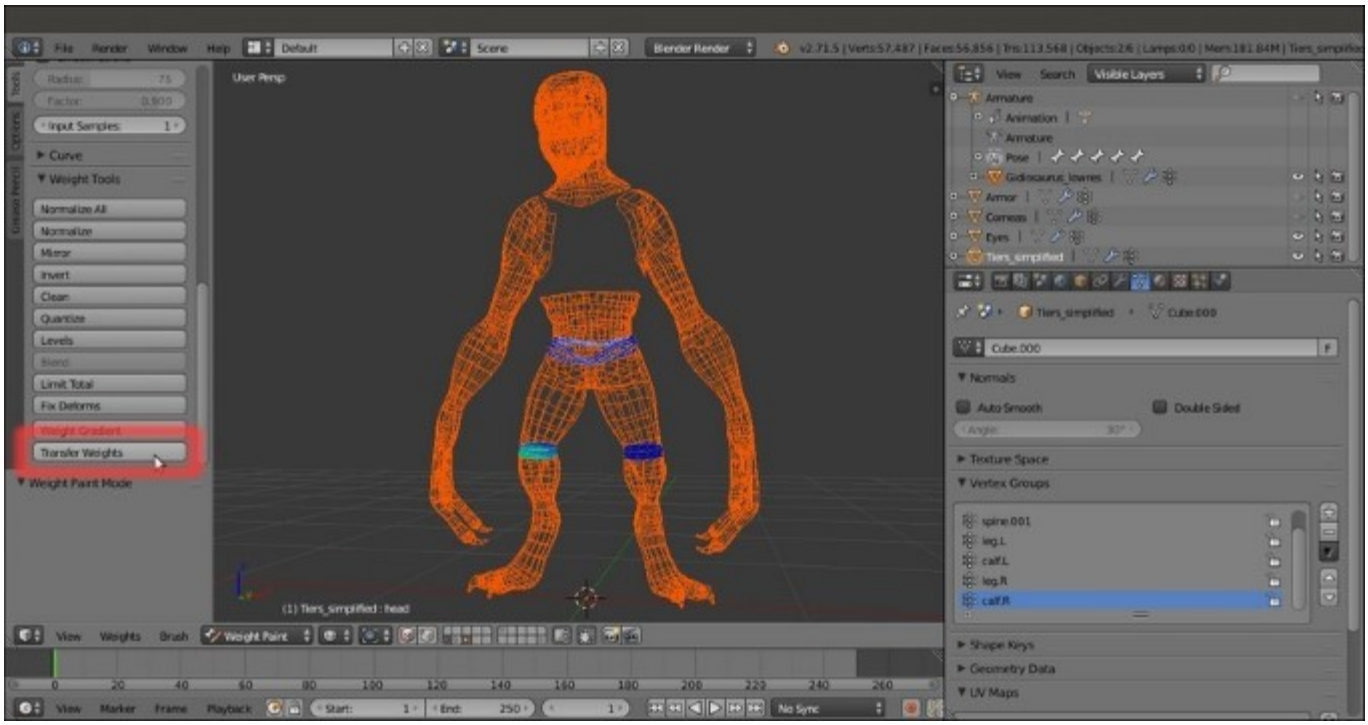


Other vertex groups

To look at exactly how the weights have been edited by the **Weight Paint** tool, open the `Gidiosaurus_skinning_03.blend` file, hide the **Armor**, select the **Gidiosaurus** mesh and press *Ctrl* + *Tab* to go in to **Weight Paint** mode, and then right-click to select the different bones.

One last thing still remains to be done: we must also skin the **Tiers_simplified** object.

12. Enable the **13th** scene layer to show the **Armor** and the **Tiers**; temporarily hide both the **Armature** and the **Armor** object by clicking on the respective eye icon in the **Outliner**.
13. Select the **Gidiosaurus** mesh, then, *Shift*-select the **Tiers** object and press *Ctrl* + *Tab* to go in to **Weight Paint** mode.
14. Go to the **Weight Tool** subpanel under the **Tool Shelf** and click on the **Transfer Weight** button, which is the last button at the bottom. After a bit of calculation, the weights of the vertices for the underlying **Gidiosaurus** mesh have been transferred to the corresponding overlaid vertices of the **Tiers** object and the vertex groups as well:



Transferring the vertex group weights from the Gidiosaurus mesh to the tiers object

15. Go out of **Weight Paint** mode and select the sole **Tiers** object. In the **Object Modifiers** window, assign an **Armature** modifier or, if you prefer, just join it to the **Armor** object (**Armor** as an active object and then press *Ctrl* + *J*). In both cases, just remember to enable the **Preserve Volume** item.
16. Save the file.

See also

- http://www.blender.org/manual/modeling/meshes/vertex_groups/weight_paint.html

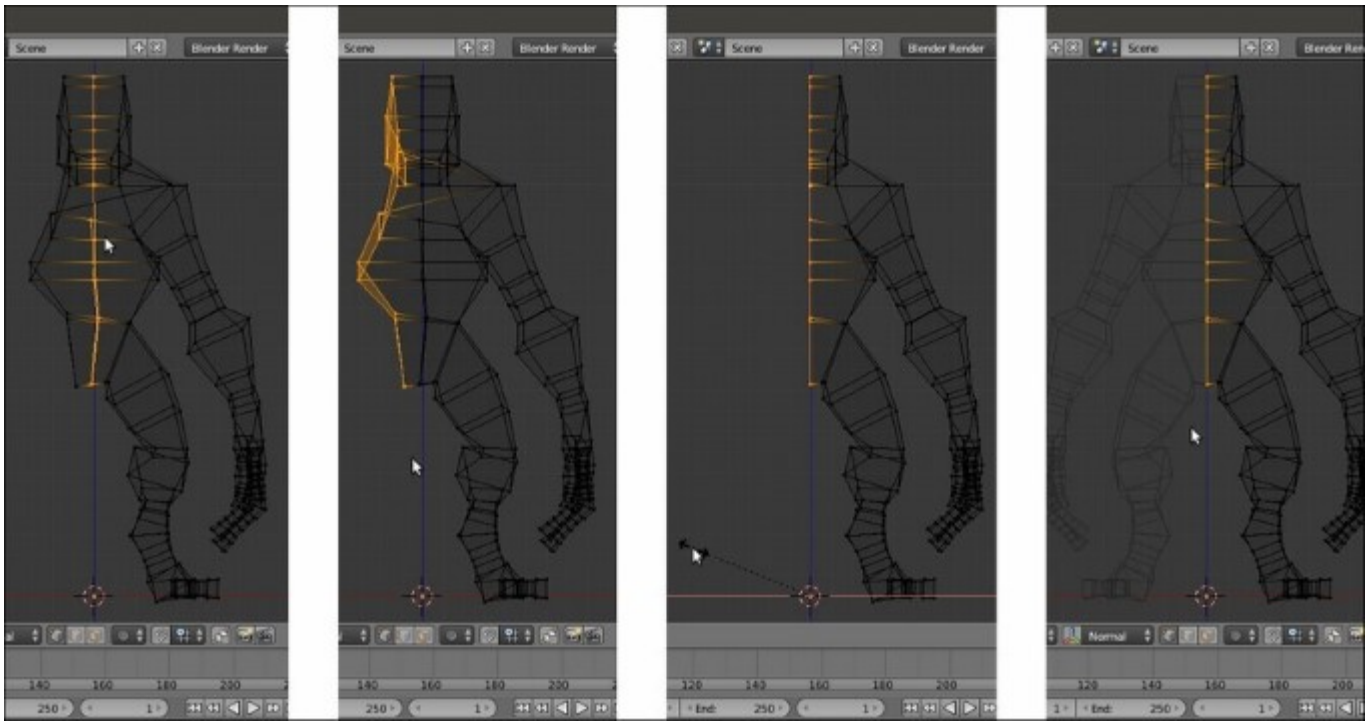
Using the Mesh Deform modifier to skin the character

The **Mesh Deform** modifier has been introduced in Blender for the production of the short open movie *Big Buck Bunny* and it's a very easy and quick way to skin medium and high resolution characters' meshes. Although the utility of this modifier really shows the skinning of fat, chubby characters, it will be useful to see the way it works even if applied to a quite skinny character such as the **Gidiosaurus**.

Getting ready

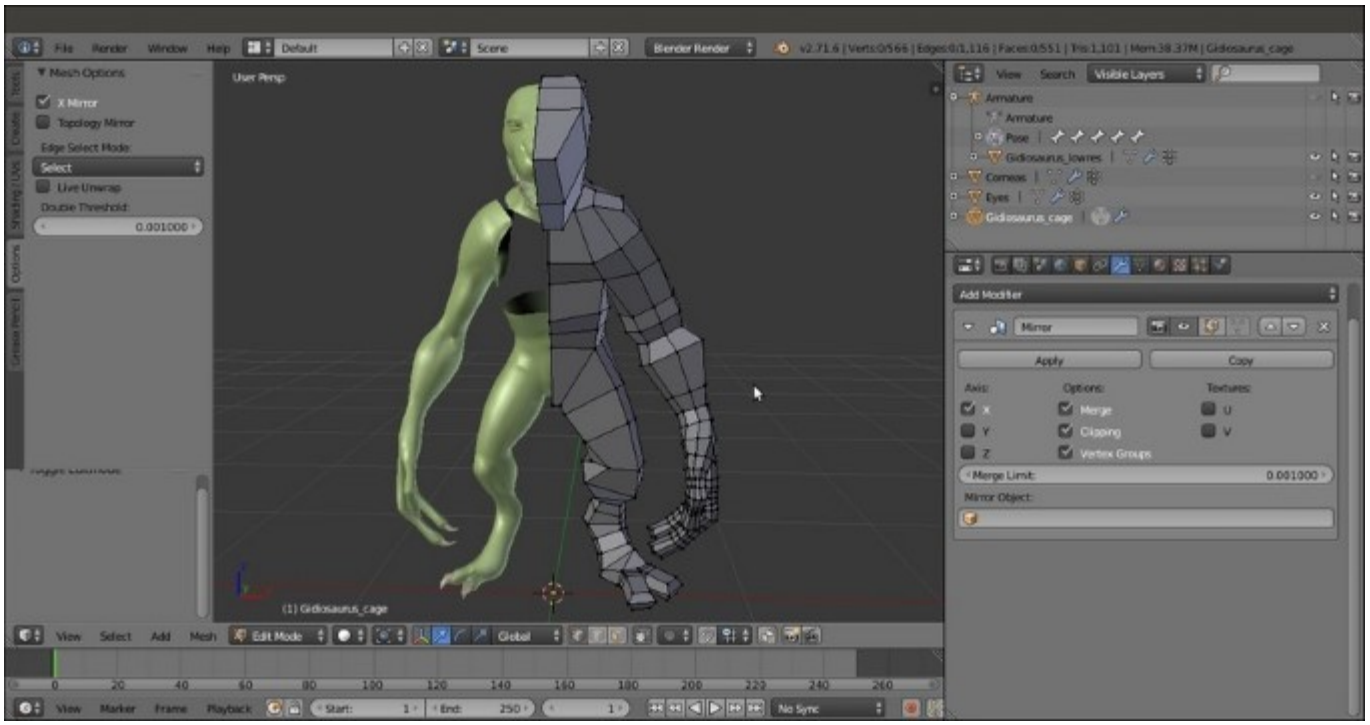
First, we must prepare the **deforming cage**, which is a simplified low poly mesh totally enveloping the character's mesh; to do this, in our case, we can start from an already made object:

1. Open the `Gidiosaurus_skinning_03.blend` file.
2. Click on the **File** | **Append** menu (or press *Shift* + *F1*), browse to the folder with all the project files, and click on the `Gidiosaurus_base_mesh_02.blend` file. Then, click on the `Object` folder and select the **Gidiosaurus** item.
3. Move the just-appended object to the first scene layer; then, go to the **Outliner** and rename it as **Gidiosaurus_cage**.
4. This is the base mesh we built in the first chapter of this module, so go to the **Object Modifier** window and apply the **Skin** modifier; delete the **Mirror** modifier and disable the **Subdivision Surface** modifier visibility in the viewport by clicking on the eye icon button.
5. Go in to **Edit Mode** and by pressing *Ctrl* +*R*, cut a median vertical edge-loop at the center of the mesh.
6. Select the vertices of the *missing* half and delete them; then, select the median edge-loop and, with **Pivot Point** set to **3D Cursor** (and the **3D Cursor** located at the center of the scene), scale them to **0.000** along the global *x* axis.
7. Assign a new **Mirror** modifier and enable the **Clipping** item.



Preparing the deforming cage

8. Now, enable the scene layer with the **Gidiosaurus** mesh, select it, and temporarily disable the **Armature** modifier by clicking on the *Display modifier in viewport* button (the one with the eye icon).
9. In the **Outliner**, click on the eye icon to hide the **Armature**.
10. Reselect **Gidiosaurus_cage**, enter **Edit Mode**, and start to edit. Basically, the cage must be large enough to totally include the character's mesh.
11. Select whole parts such as the **head** or a **hand** and scale the vertices on their normals (*Alt* + *S*) and move the vertices by hand.
12. Where necessary, add edge-loops (*Ctrl* + *R*) to refine the cage's shape, but try to keep it as simple and low resolution as possible.



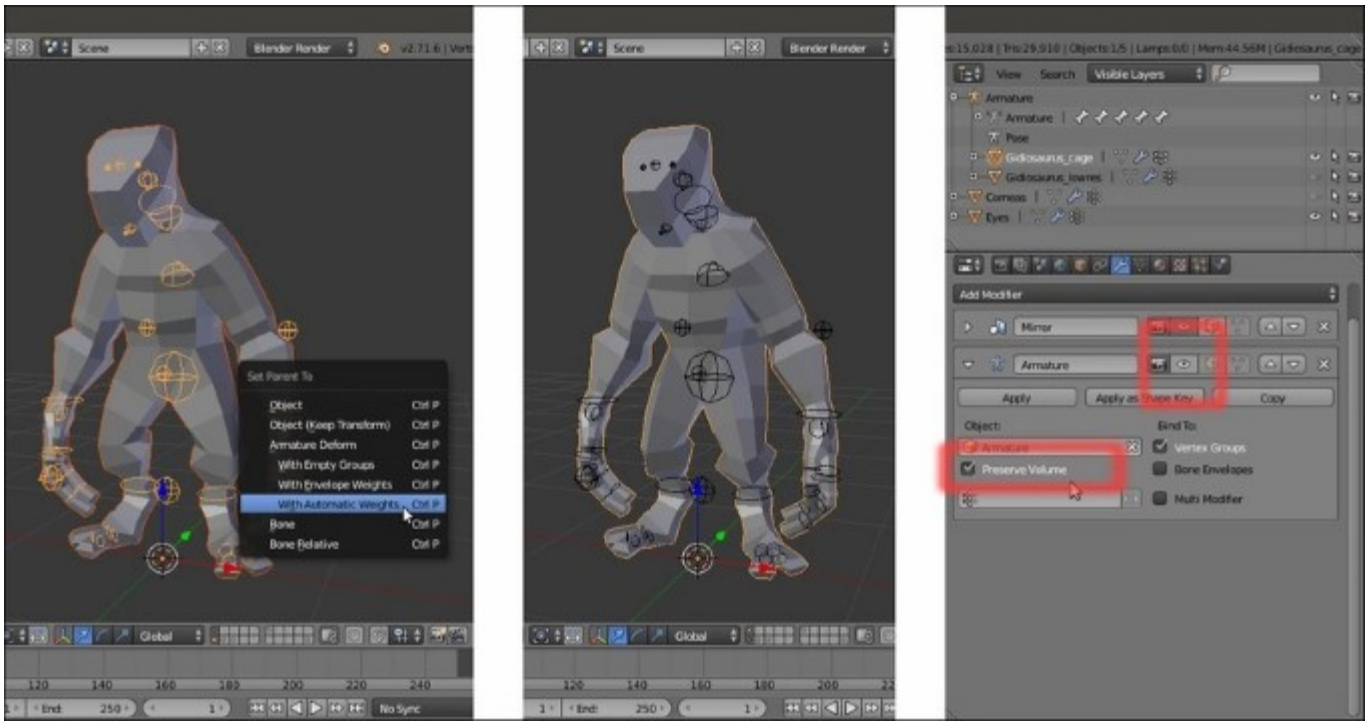
The cage mesh in Edit Mode

13. Once we have confirmed that the **Gidiosaurus** mesh is totally contained in the cage, we can go out of **Edit Mode**.

How to do it…

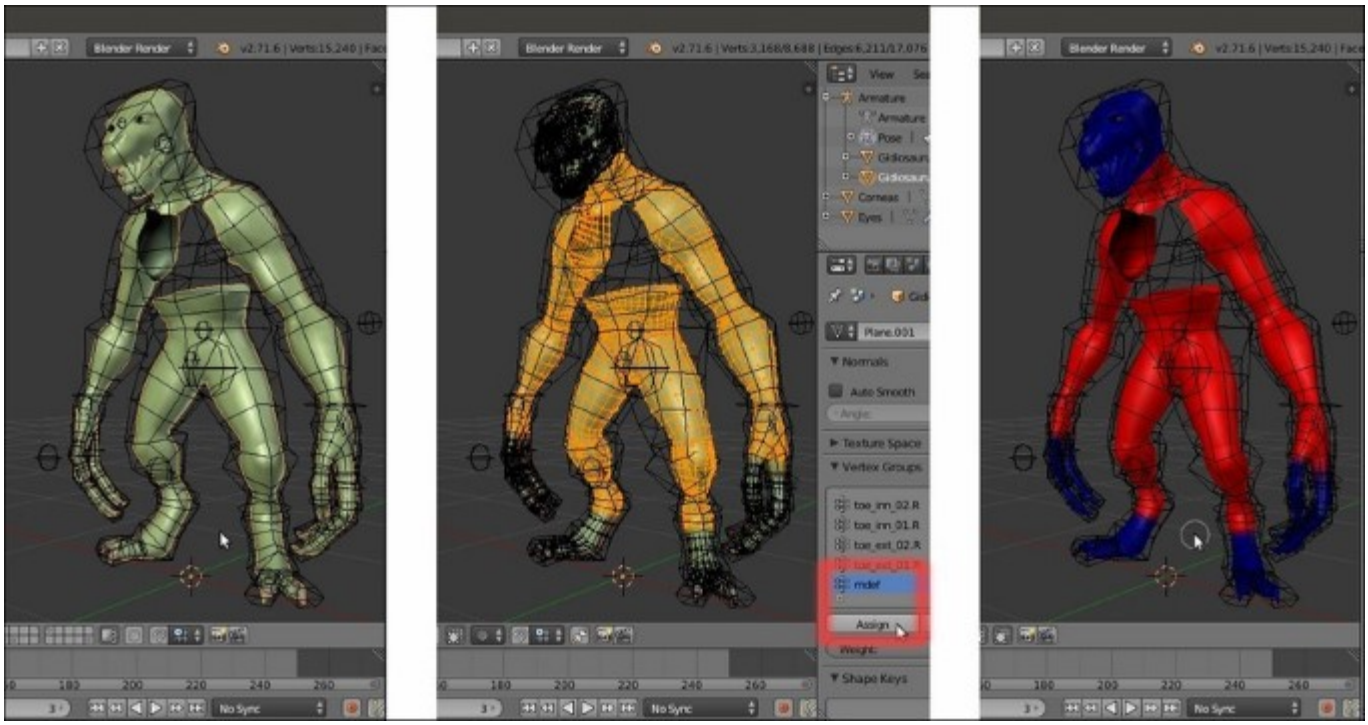
Now that the **deforming cage** is made, we can go on with the **skinning**:

1. Unhide the **Armature** and select it. Go in to **Edit Mode**, select the bones deforming the **Armor** (see the *Parenting the Armature and Mesh using the Automatic Weights tool* recipe in this chapter for this), and press *Shift* + *W* | **Deform**. Don't deselect anything because it will be useful later on to have the bones still selected, and go straight back to **Object Mode**.
2. Now, hide the **Gidiosaurus_lowres** object; then, select the **Gidiosaurus_cage** object, *Shift*-select the **Armature**, and press *Ctrl* + *P* | **With Automatic Weights**.
3. Select the sole **Armature**, go in to **Edit Mode** and press *Shift* + *W* | **Deform**, and then switch to **Pose Mode**.
4. Reselect the **cage** and go to the **Object Modifiers** window; in the **Armature** modifier, enable the **Preserve Volume** item, but temporarily disable the visibility of the modifier in the viewport (eye icon button):



Parenting the deforming cage to the Armature

5. Go to the **Object** window and click on the **Maximum Draw Type** button under the **Display** subpanel to select the **Wire** item. Unhide the **Gidiosaurus_lowres** object.
6. Select the **Gidiosaurus** mesh and go in to **Edit Mode**. In the **Vertex Groups** subpanel under the **Object Data** window, create a new group and rename it as **mdef**; select all the vertices of the **Gidiosaurus** mesh *except feet, fingers, and the head* and assign them to the **mdef** vertex group:



The "mdef" vertex group

7. Go to the **Object Modifiers** window; in the **Armature** modifier panel, click on the vertex group empty field at the bottom (*name of Vertex Group which determines influence of modifier per point*) to select the **mdef** vertex group and then click on the *invert vertex group influence* button to the left (the one with the two arrows pointing in opposite directions). Temporarily, disable the visibility of the modifier in the viewport (eye icon button).
8. Assign a **Mesh Deform** modifier and move it upwards in the stack, before the **Subdivision Surface** modifier but after the **Armature** one.
9. In the **Object** field of the **Mesh Deform** modifier, select the **Gidiosaurus_cage** item; in **Vertex Group** again, select the **mdef** vertex group, check the **Dynamic** item box, and then click on the **Bind** button.
10. Save the file as `Gidiosaurus_mesh_deform.blend`.

How it works…

The **Gidiosaurus_cage** is a very simple mesh. Therefore, it is very easily skinned to the **Armature** (we didn't do it in our case, but obviously, when necessary, the automatic weights assigned by the parenting can be easily edited as in the *Editing the Weight Groups using the Weight Paint tool* recipe) and is therefore deforming, through the binding of the **Mesh Deform** modifier, the more subdivided **Gidiosaurus** mesh.

In fact, if everything went right, now we should have the **Gidiosaurus** body correctly deformed by the **cage** only for the vertices that belong to the **mdef** vertex group, while the **Armature**, which also deforms the **cage**, is still taking care of the vertices outside the group; to check this, just try to pose the

rig and alternatively disable, in the **Object Modifiers** window, the viewport visibility of the **Armature** and **Mesh Deform** modifiers for the mesh.

Note that even we didn't apply the **Mirror** modifier to the **cage** object; the **Mesh Deform** modifier works correctly anyway, exactly like the **Armature** one.

The Gidiosaurus model posed through the Mesh Deform modifier

See also

- http://www.blender.org/manual/modifiers/deform/mesh_deform.html

Using the Laplacian Deform modifier and Hooks

One of the last modifiers introduced in Blender, the **Laplacian Deform** modifier shouldn't actually be considered as an effective tool to rig a character, but more as a tool to modify, change, or refine a default pose. Anyway, if set and used smartly it can often give interesting results, so it has been included in this chapter as well.

Getting ready

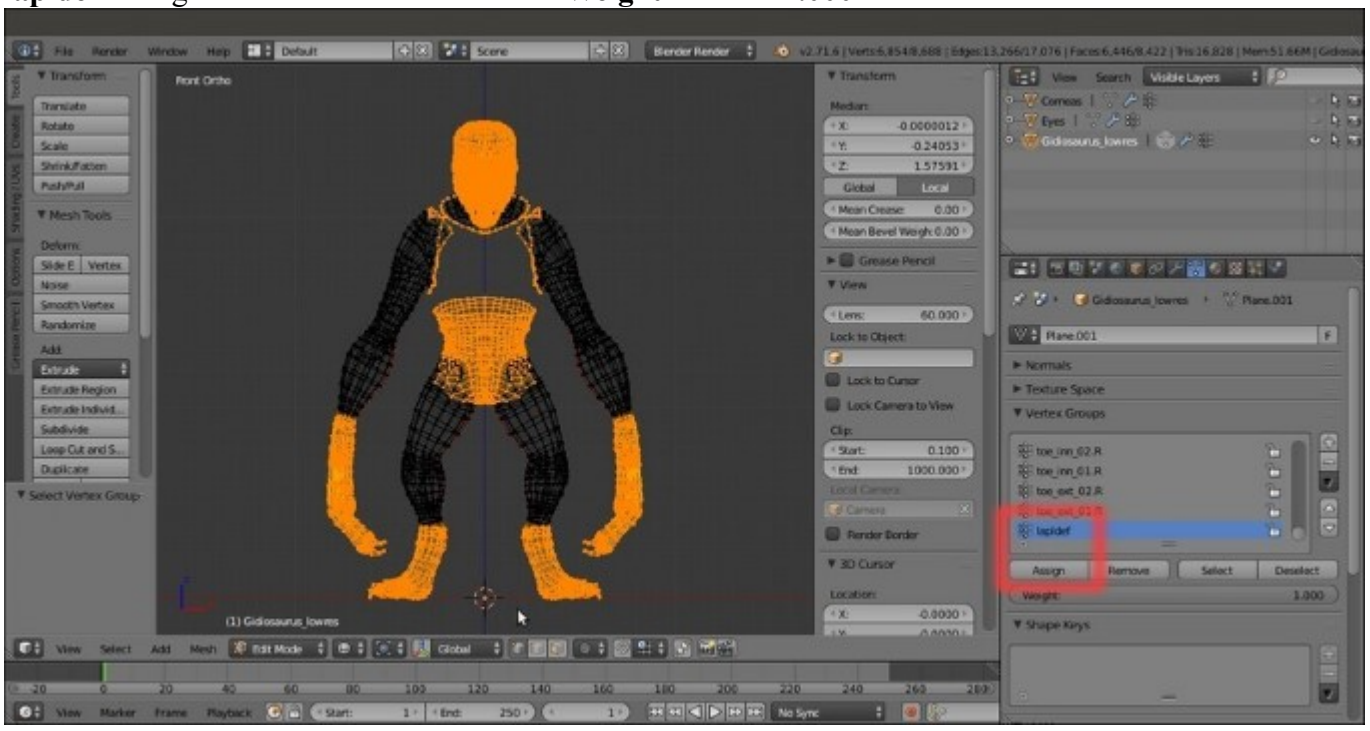
First, let's prepare the scene:

1. Open the `Gidiosaurus_rig_from_scratch_01.blend` file.
2. Select and then delete the **Armature** in **Object Mode**; then. select the **Gidiosaurus** mesh and delete the **Armature** modifier too in the **Object Modifiers** window.
3. In the **Outliner**, hide the **Eyes** object.
4. Press the *Z* key to go in the **Wireframe** viewport shading mode.

How to do it…

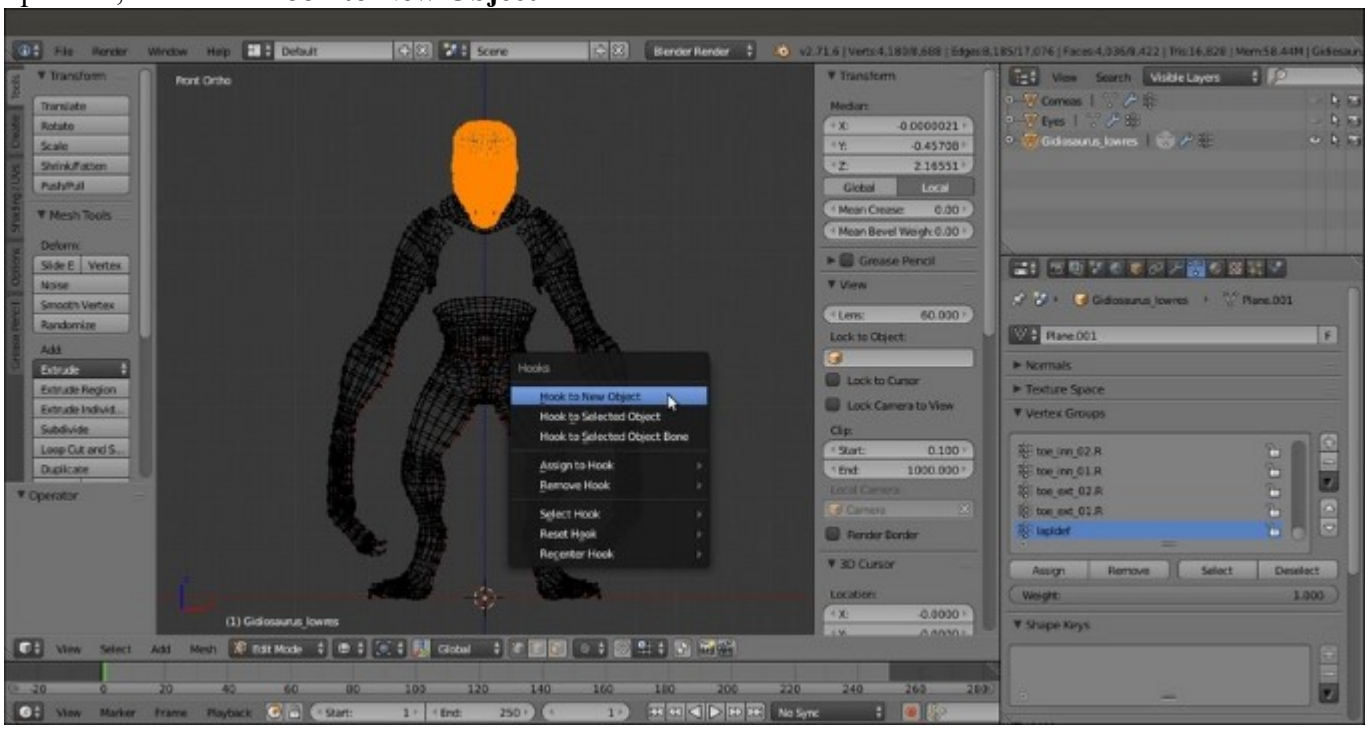
Now let's go with the **Laplacian** modifier setup:

1. With the **Gidiosaurus** mesh still in **Edit Mode**, select all the vertices of the **hands**, **feet**, **hip**, **head**, plus the **boundary edge-loops** where the mesh is missing (look at the following screenshot).
2. Go to the **Vertex Groups** subpanel and create a new group named as you wish; I named it **lapldef**. Assign the selected vertices with a **Weight** value of **1.000**:



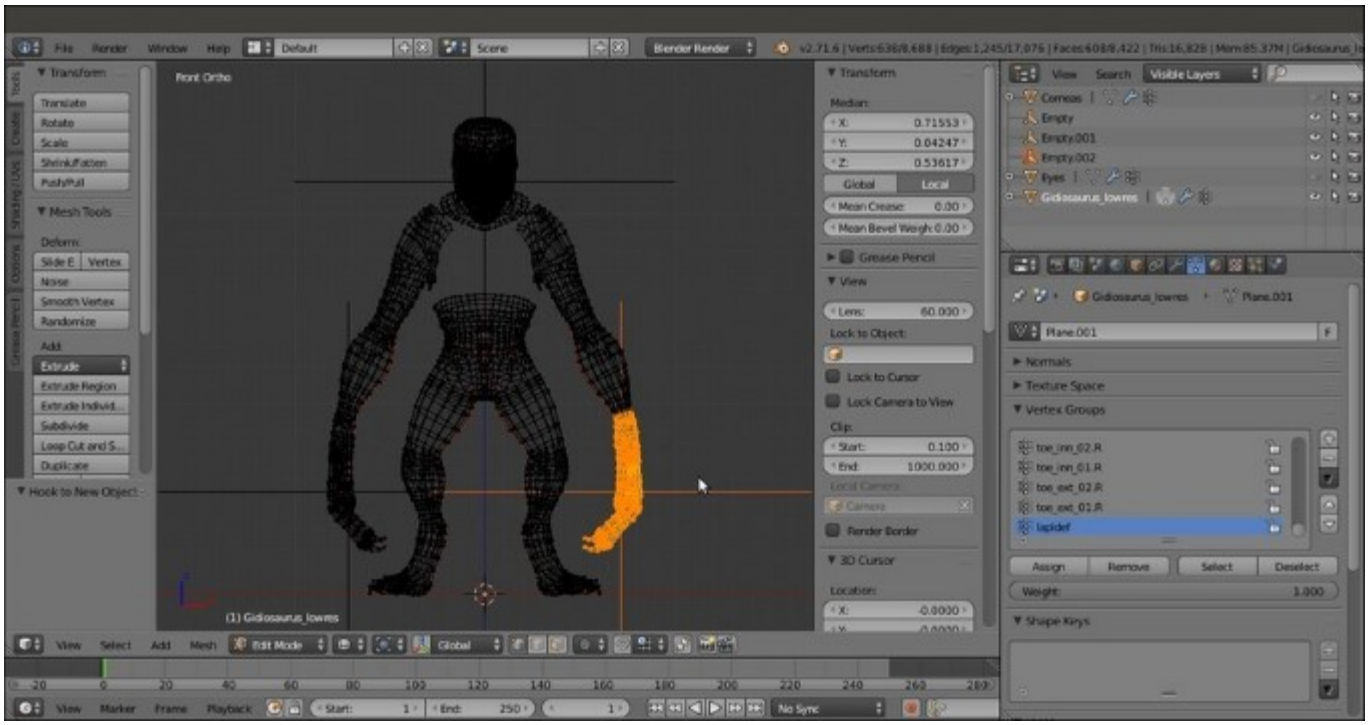
The "lapldef" vertex group

3. Now, box-deselect all the vertices, except the **head** ones; press *Ctrl* + *H* and in the **Hooks** pop-up menu, select the **Hook to New Object** item:



The Hooks menu

4. Click on the **Select** button under the **Vertex Groups** subpanel to the right of the screen and then deselect all the vertices, except the **right hand** ones. Again, press *Ctrl* + *H* and in the **Hooks** pop-up menu, select the **Hook to New Object** item.
5. Click on the **Select** button again, deselect all the vertices, except the **left hand** ones, and repeat the procedure:



The Hook assigned to the left hand vertices

6. Repeat the procedure separately for the left and the right **feet** and then go out of **Edit Mode**.
7. In the **Outliner**, rename the **Empties** (the **Hooks**) respectively as **Empty_head**, **Empty_hand.L** and **.R**, and **Empty_foot.L** and **.R**.

The Hooks assigned to the mesh's vertices

8. Select the **Gidiosaurus** mesh and go to the **Object Modifiers** window. Collapse all the five **Hook** modifiers for better visibility and assign a **Laplacian Deform** modifier; move it upwards in the stack, just before the **Subdivision Surface** modifier (*but* always after the **Hook** modifiers). Click on the **Anchors Vertex Group** to select the **lapldef** vertex group and then click on the **Bind** button.
9. For better visibility, select each **Hook** and in the **Object Data** window, set the size to **0.40**.
10. Enable the **3D manipulator widget** in the 3D view toolbar (or press *Ctrl* + Spacebar), *Shift*-click on the **Translate** and **Rotate** buttons, and set **Transform Orientation** to **Normal**.
11. Select the **Hooks** and start to move and rotate them using the **3D manipulator widget**, to pose the **Gidiosaurus** mesh:

The Gidiosaurus mesh posed through the Hooks and the Laplacian Deform modifier

12. Save the file as `Gidiosaurus_laplacian.blend`.

How it works…

Remember that because they don't work through joints, the **Laplacian Deform** modifier and the **Hooks** don't give a realistic deformation and should be used more to tweak a character pose only inside a limited range. Building a more complex rig, also with **Hooks** at the **elbows** and **knees**, is possible but probably more useful for other types of *unreal* characters' shapes.

It should also be remembered that the **Hooks**, once moved out of their location, can't be simply moved back to their original position by the *Alt* + *G* shortcut because this command would set them at their original **0, 0, 0** location. Instead, any rotation can be easily removed by the *Alt* + *R* shortcut.

In any case, the *Ctrl* + *Z* (**Undo**) shortcut can be used, but first check the number of **Steps** set in the **User Preferences** panel under the **Global Undo** item (there are only **32** by default).

See also

- http://www.blender.org/manual/modifiers/deform/hooks.html
- http://www.blender.org/manual/modifiers/deform/laplacian_deform.html

Chapter 8. Finalizing the Model

In this chapter, we will cover the following recipes:

- Creating shape keys
- Assigning drivers to the shape keys
- Setting movement limit constraints
- Transferring the eyeball rotation to the eyelids
- Detailing the Armor by using the Curve from Mesh tool

Introduction

In this chapter, we'll see how to create and add **shape keys** (the Blender term for **morphing**) to the model, to create facial expressions for the **Gidiosaurus** and to add shape modifications in a non-destructive way to the model.

Then, we'll see how to set a limit to the **Armature** bones' rotation using constraints and how to slightly transfer a portion of the rotation movement of the **eyeballs** to the covering **eyelids**.

Last, we'll add some detail to the **Armor** by quickly adding **rivets** through a simple and effective technique.

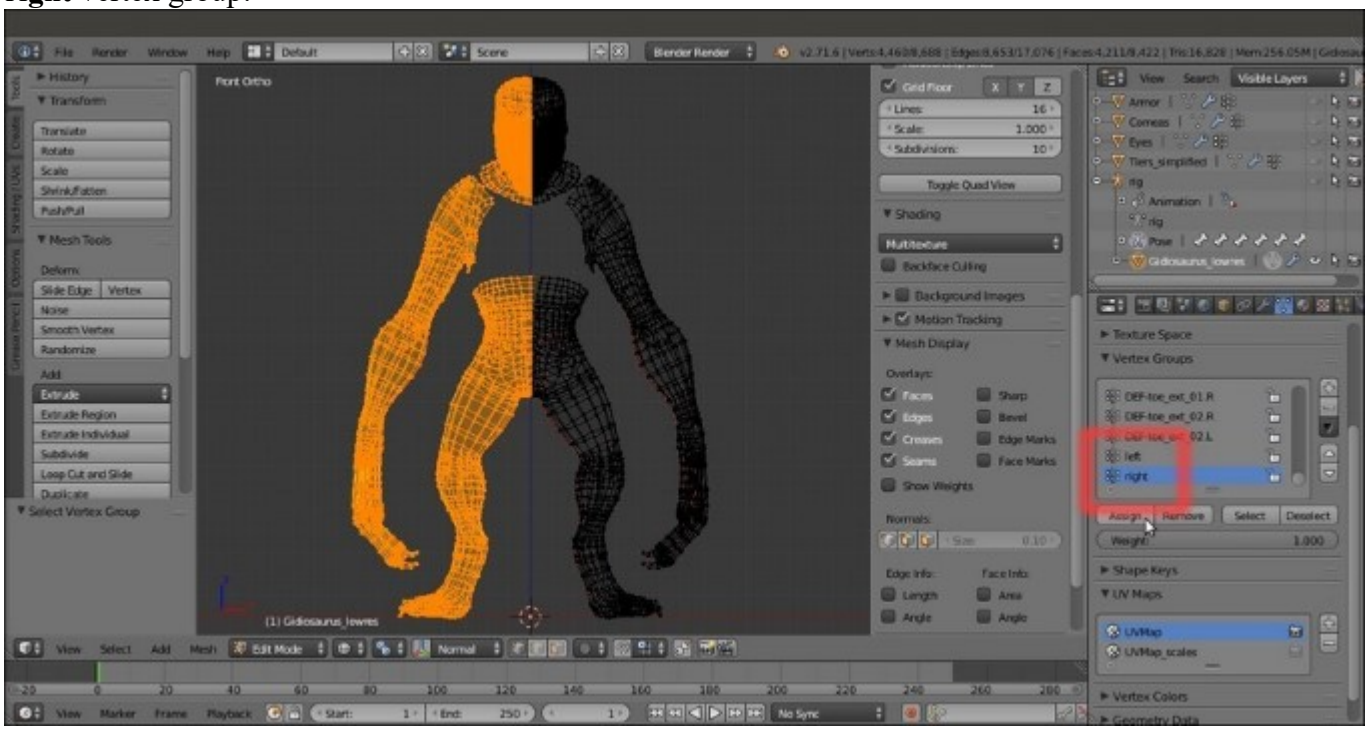
Creating shape keys

In this recipe, we'll set the **shape keys** to create (even if limited) **facial expressions** and to fake the stretching and the contracting effect of the character's **arm muscles**, and we'll add some more shape keys to fix issues in the character's shape.

Getting ready

First, let's prepare a bit the scene and the model:

1. Start Blender and load the `Gidiosaurus_skinning_rigify.blend` file, which is the same as the `Gidiosaurus_skinning_03.blend` file but with the **rig** created by the **Rigify** add-on (and later edited to add the other bones exactly as explained in the last chapter's recipes).
2. In the **Outliner**, click on the respective eye icons to hide the **Armor**, **Eyes**, and **Tiers_simplified** objects and the **Armature**, whose name in this case is **rig**.
3. Select the **Gidiosaurus_lowres** object and press the *1* key on the numpad to go into the **Front** view.
4. Press *Z* to go into the **Wireframe** viewport shading mode and the *5* key on the numpad to switch to the **Ortho** view if it is not already.
5. Enter **Edit Mode** and box-select the **left half** of the mesh vertices (including the **middle vertical edge-loop**) and in the **Vertex Groups** subpanel under the **Object Data** window, create a new vertex group; rename it **left** and assign the selected vertices a weight of **1.000**.
6. Deselect everything and repeat this process for the **right half** of the mesh's vertices to create the **right** vertex group:



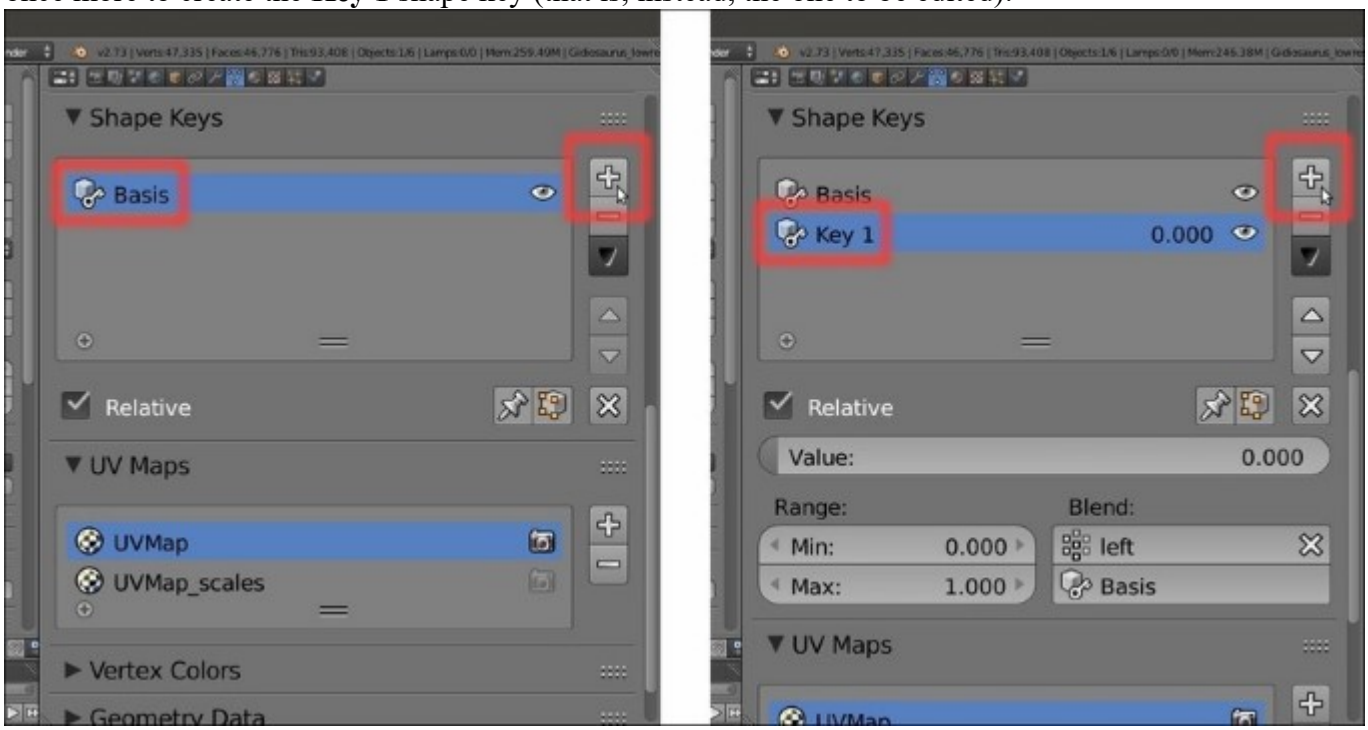
Assigning the mesh's right side vertices to the "right" vertex group

7. Save the file as `Gidiosaurus_shapekeys.blend`.

How to do it…

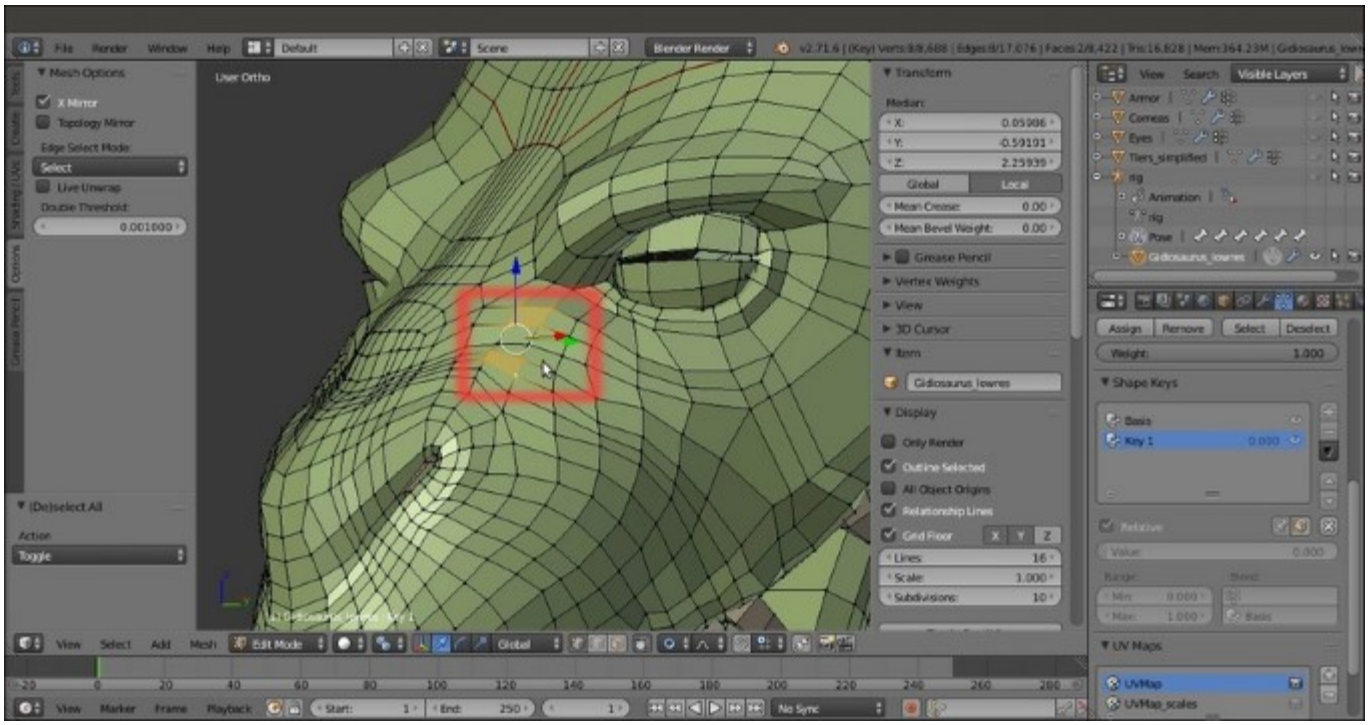
Let's start now by creating the **facial expressions** shape keys:

1. Exit **Edit Mode** and expand the **Shape Keys** subpanel under the **Object Data** window; click on the + icon button to the top left to create the **Basis** shape key (that mustn't be edited), then click once more to create the **Key 1** shape key (that is, instead, the one to be edited):



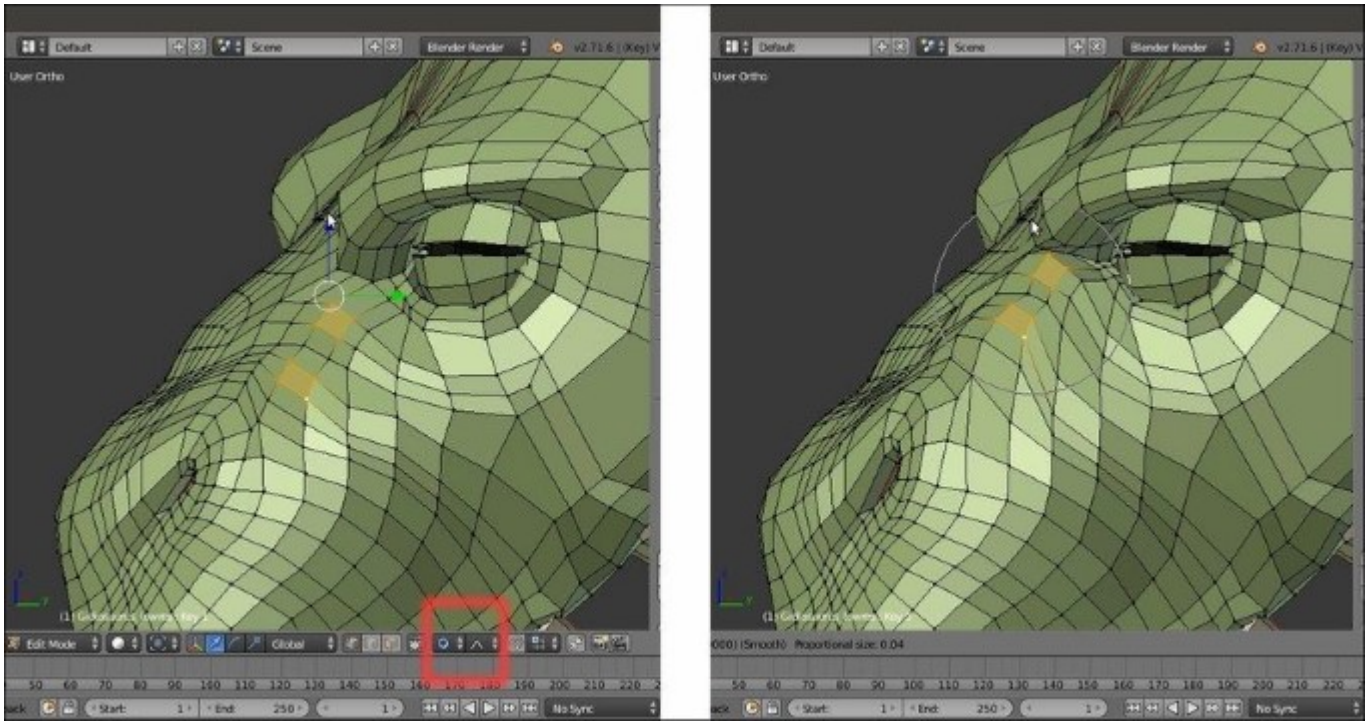
Creating the Basis and the first shape key

2. Be sure that the **X Mirror** item in the **Mesh Options** tab under the **Tool Shelf** is activated and click on the **PET** (**Proportional Editing Tool**) button in the 3D viewport toolbar (or activate it by pressing the *O* key).
3. Zoom to the **Gidiosaurus** head and again in **Edit Mode**, select some of the vertices at the center of the **snout** area, as indicated in the following screenshot:



Selecting the vertices

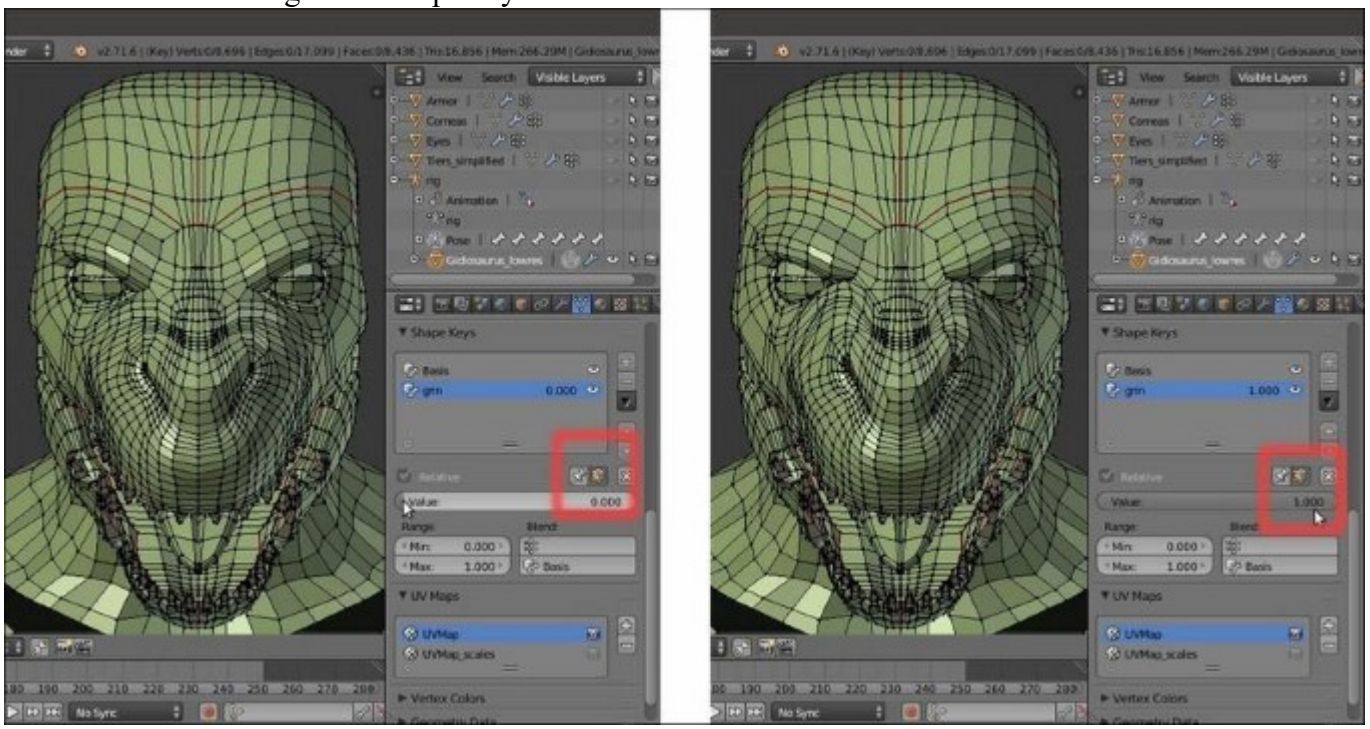
4. Move them upwards and towards the **eye**, set the amount for the **PET** smoothing by scrolling the mouse wheel:



Moving the selected vertices slightly backwards and upwards

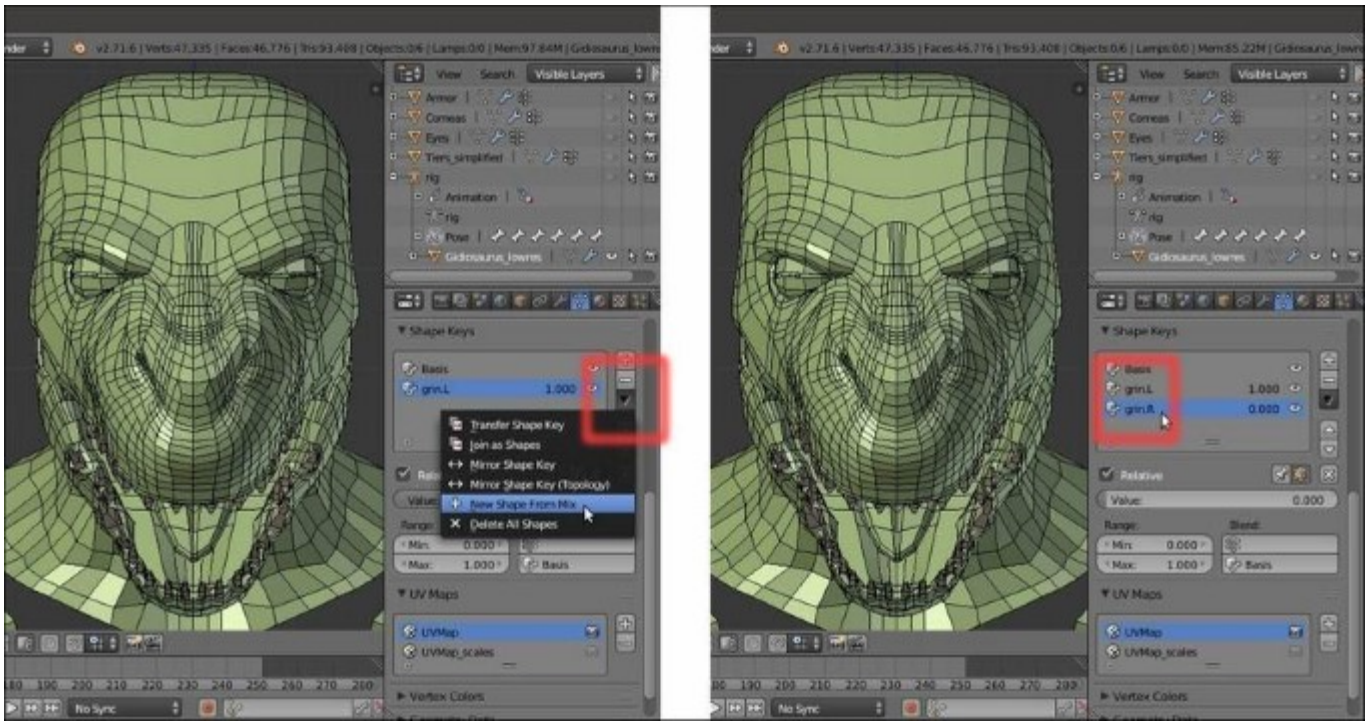
Note that we selected two faces instead of the folder edges, because the muscular *scrunching* involves both movement of the skin and a slight folding of the skin as well; the middle edge does not move as much as the selected faces due to **PET** falloff, hence creating a very slight scrunch and a more *naturalistic* skin sliding.

5. If necessary, adjust the position of the single vertices, maybe also disabling the **PET**.
6. Go to the **Shape Keys** subpanel and rename the **Key 1** shape key as **grin**.
7. Click on the *Apply shape key in edit mode* button located right above the **Value** slider to enable it; go into the **Front** view and by moving the **Value** slider from **0.000** to **1.000** and back, check for the correct working of the shape key on both the sides of the character:



Checking how the "grin" shape key works

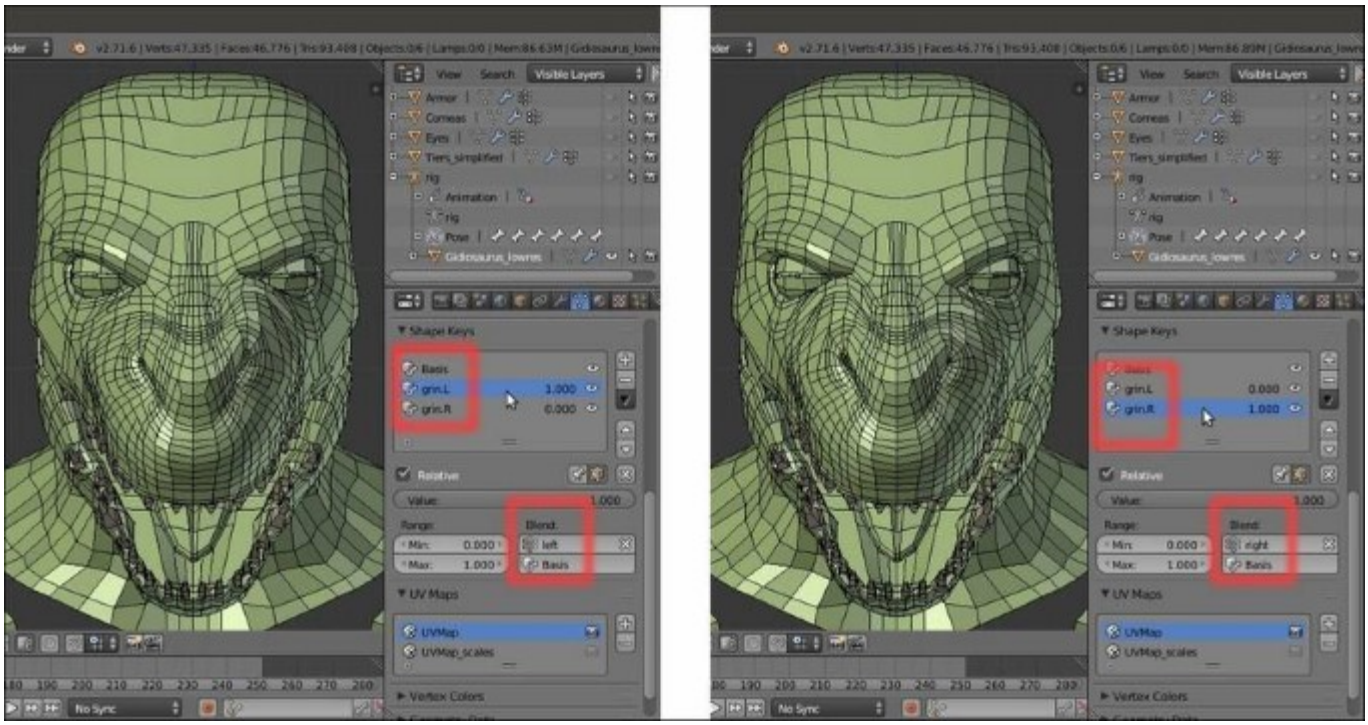
8. Go out of **Edit Mode** and just to have better visibility of the shape key modifications, go to the **Object** window to enable the **Wire** item in the **Display** subpanel.
9. Go back to the **Object Data** window and click on the button that has an icon of a downward pointing arrow (*Shape keys specials*); from the pop-up menu select the **New Shape from Mix** item: this adds a new shape key made by the sum of all the active shape keys. In this case, it is just a perfect copy of the sole **grin** shape key (the **Basis** shape key is, well, just the base starting position of the vertices in the mesh). Rename the two shape keys as **grin.L** and **grin.R**.



Copying the "grin" shape key to a new one

10. Click on the **grin.L** shape key, set the **Value** slider back to **0.000**, and then click on the *Vertex weight group* slot, the one under the **Blend** item and above the **Basis** one, and select the **left** vertex group.
11. Repeat for the **grin.R** shape key by selecting the **right** vertex group, and then once again set the **Value** slider to **1.000** and ensure it works correctly.

    Each shape key now works only on the respective side, according to the selected vertex group:

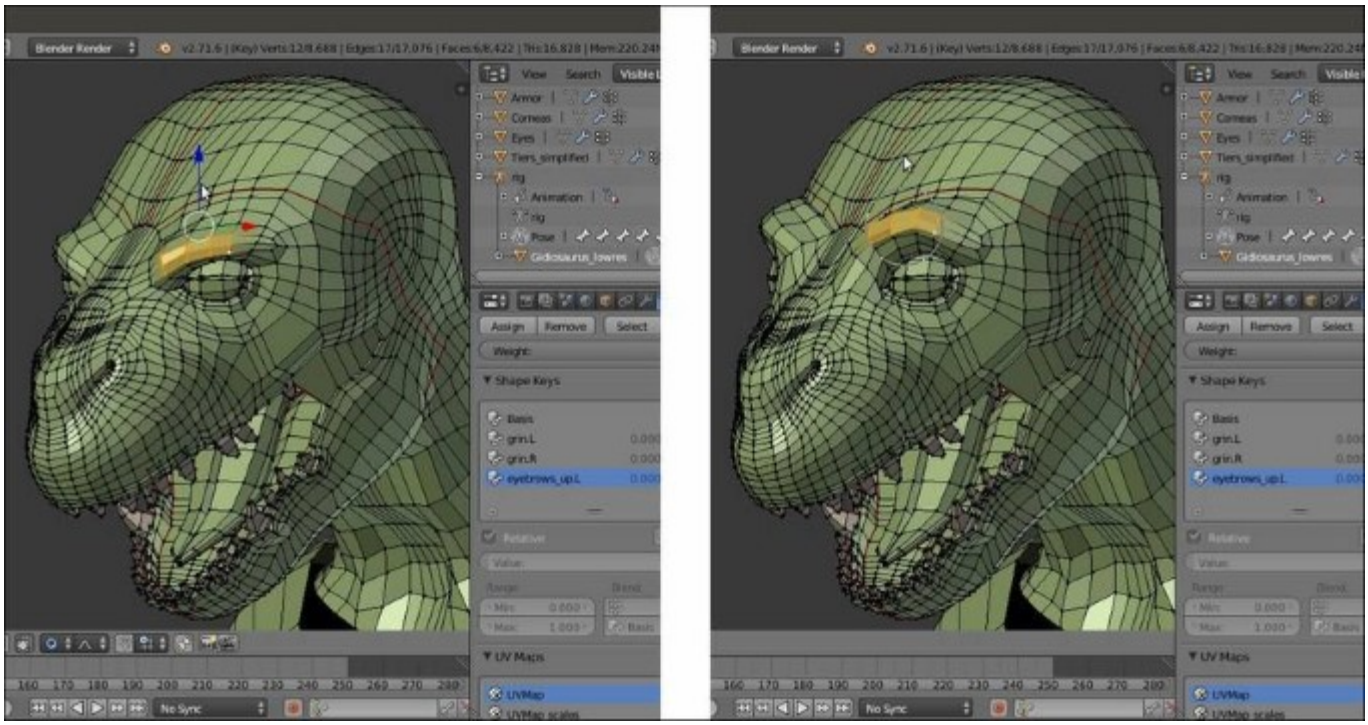


The two "grin" shape keys for the right and the left sides

This was for the **grin** expression; now we need to add at least two or three more kinds of shape keys, namely: two for the **eyebrows** (up and down) and one for the **nostrils**, multiplied for each side.

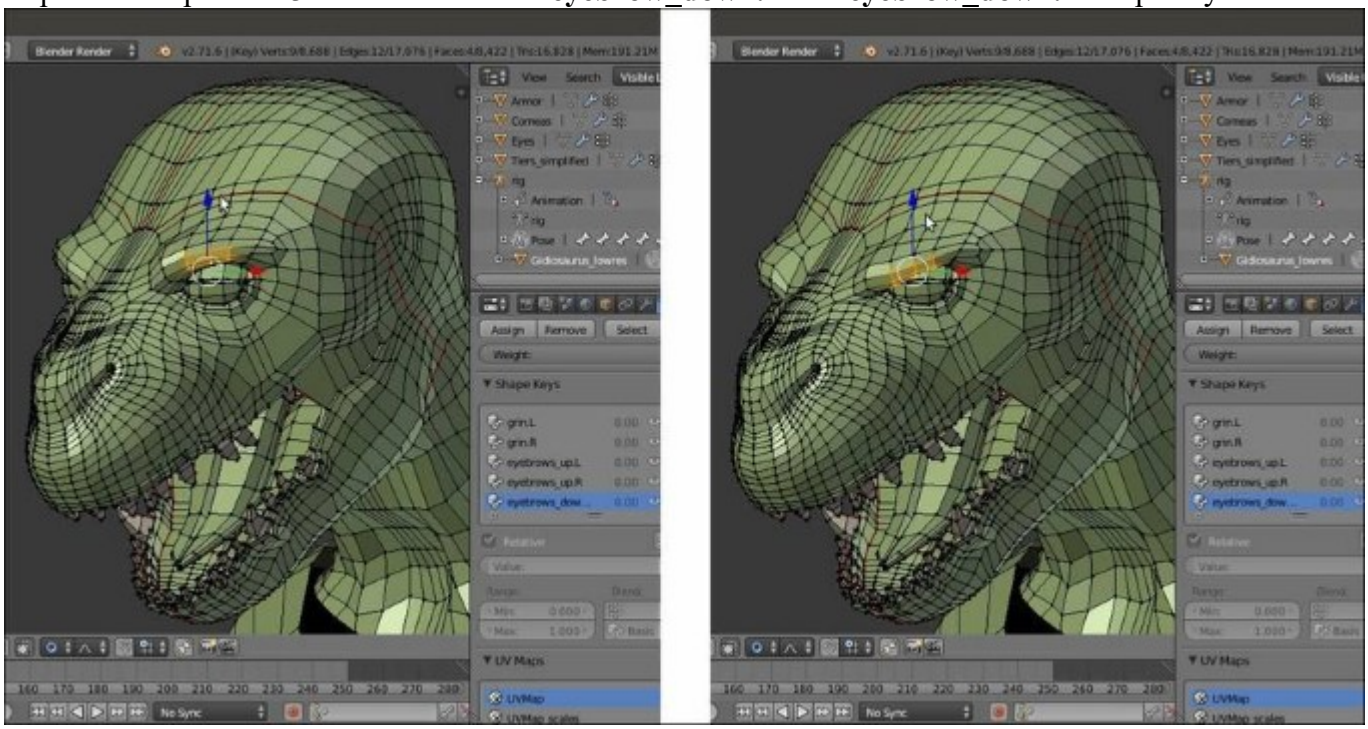
This means **six** more shape keys in total, but as you have seen, the procedure is quite quick and simple.

12. Set the **Value** slider for the **grin.L** and **grin.R** shape keys back to **0.000** and click on the + icon button to add a new shape key.
13. Rename it **eyebrow_up.L** and enter **Edit Mode**; grab some vertices on the left eyebrow and, still with the **PET** activated, move them upward; you can use the mouse wheel to set the influence of the **PET**:



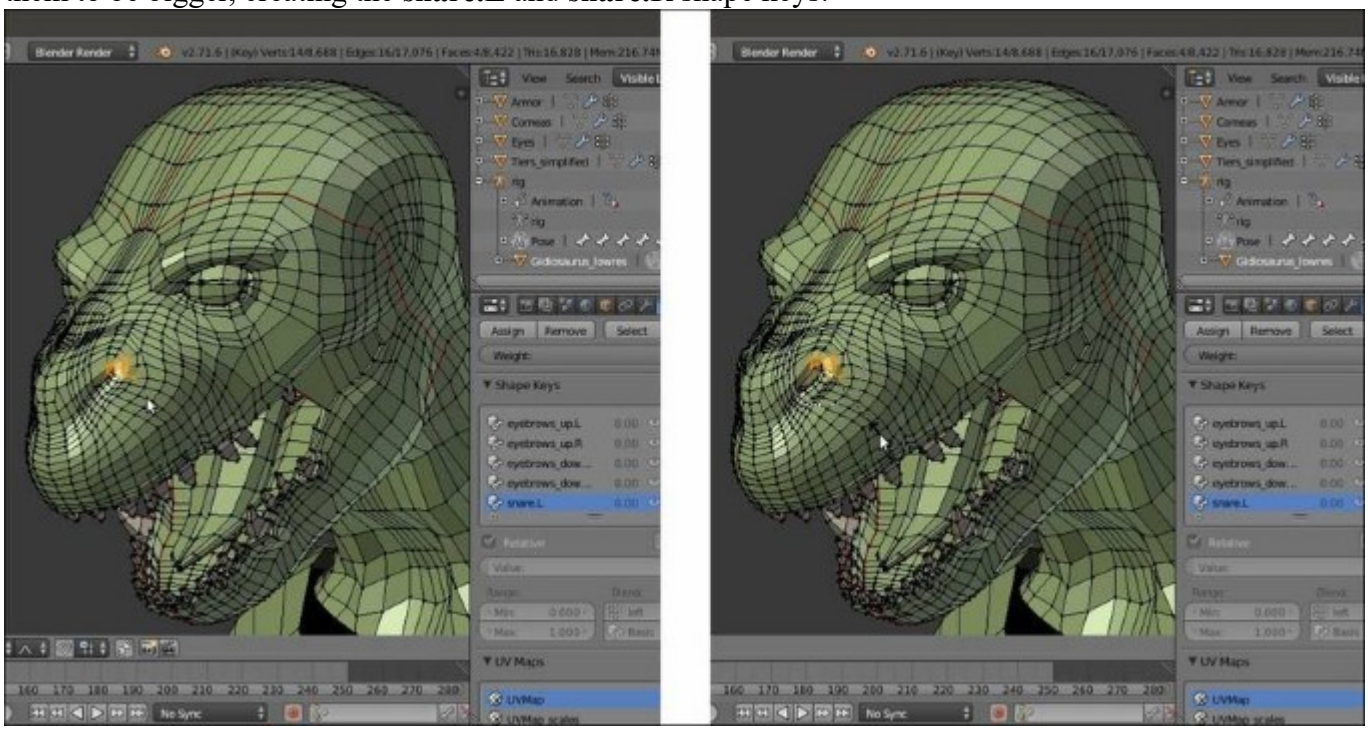
Moving the eyebrow upward

14. Repeat the steps from 9 to 11 to create the **eyebrow_up.R** shape key.
15. Repeat the steps from 3 to 11 to create the **eyebrow_down.L** and **eyebrow_down.R** shape keys:



Moving the eyebrow downward

16. Finally repeat step 3 to step 11, this time selecting the vertices around the **nostrils** and scaling them to be bigger, creating the **snare.L** and **snare.R** shape keys:

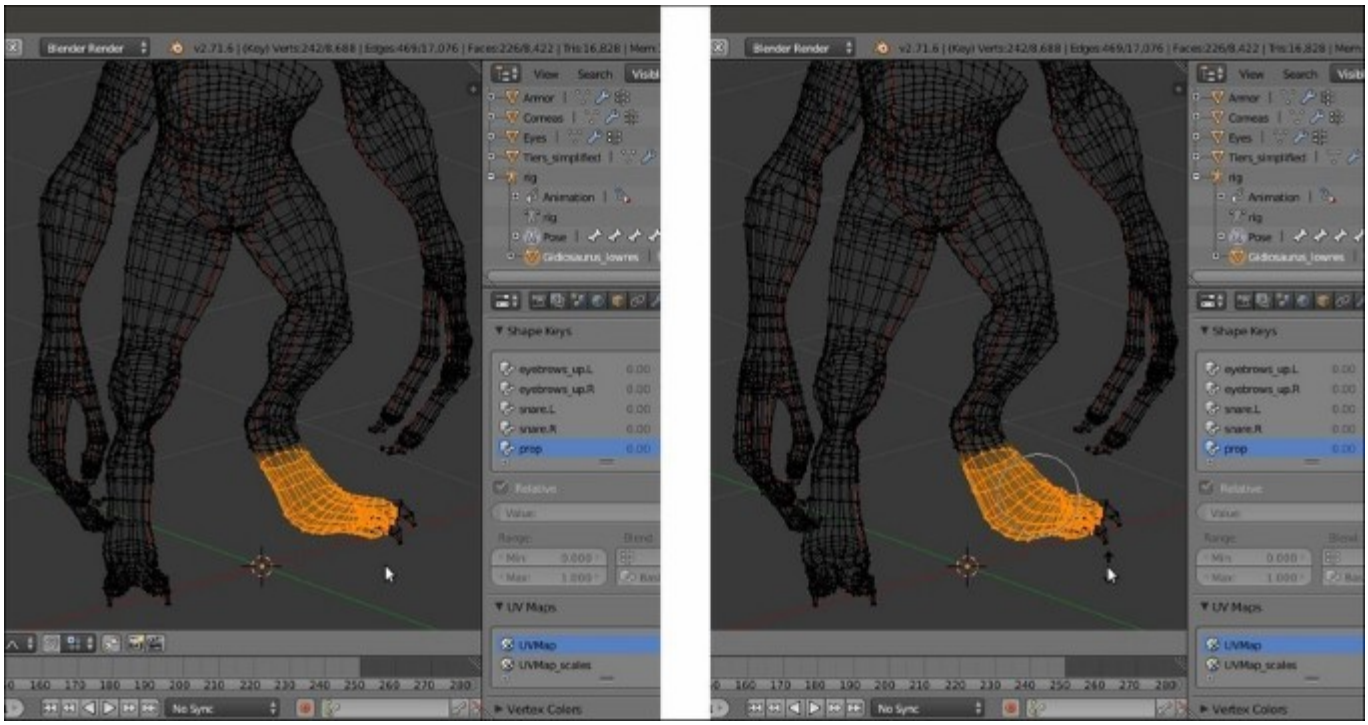


The nostril flaring

Note that, when naming the shape key for the enlargement of the nostrils, I erroneously wrote **snare**; it should have been something like *snarl* or *flaring*, but in the end it's just a naming convention and therefore, this little mistake doesn't pose a real problem.

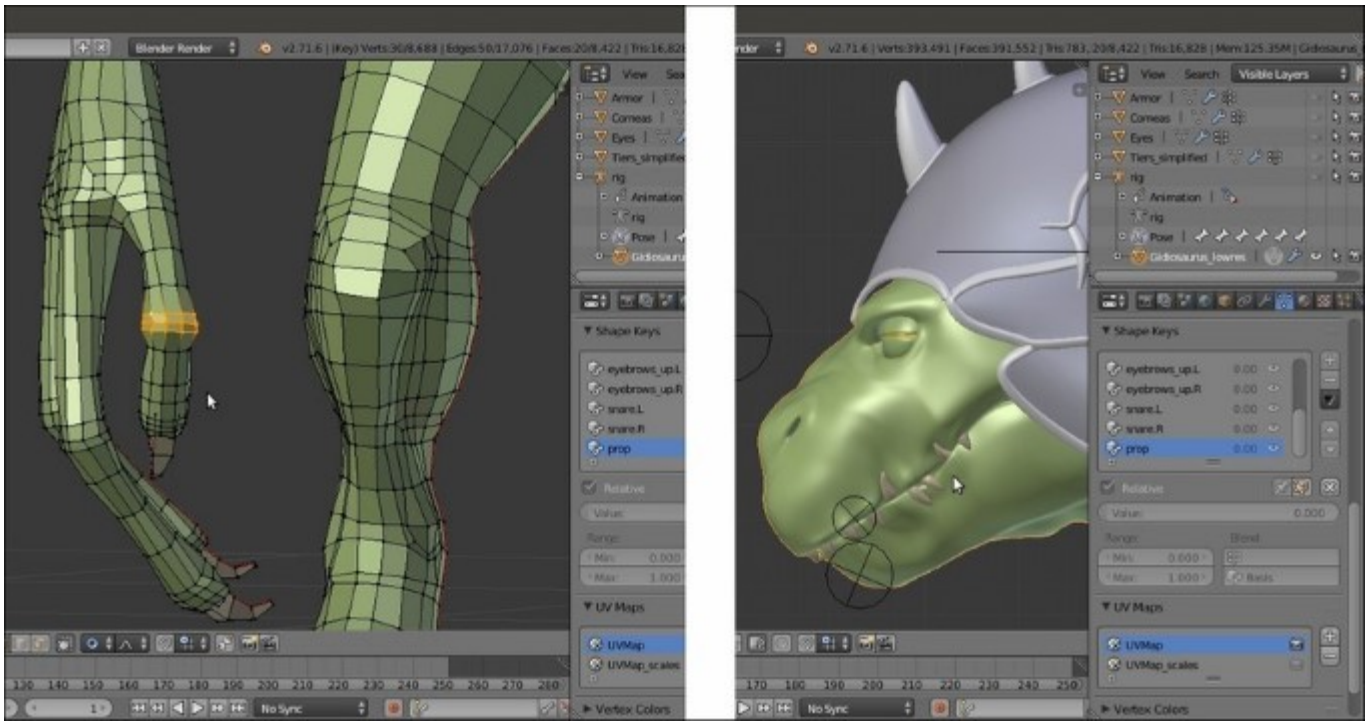
We are done with the facial expressions; now let's add one more shape key to enhance some of the body features of the **Gidiosaurus** a bit; these are not meant to be animated during the animation, but are simply a way to apply non-destructive modifications to the model.

17. Add a new shape key and rename it **prop** (for *proportions*).
18. In **Edit Mode**, select the vertices of the **left foot**, excluding the **feet talons**, and press *Alt* + *S* to scale them *on their normals*; if you are using the **PET**, just be sure to be in the **Connected** mode (so that the **PET** has influence only on the vertices connected to the selected ones, otherwise the unselected **feet talon** vertices will also be modified):



Modifying the feet proportions

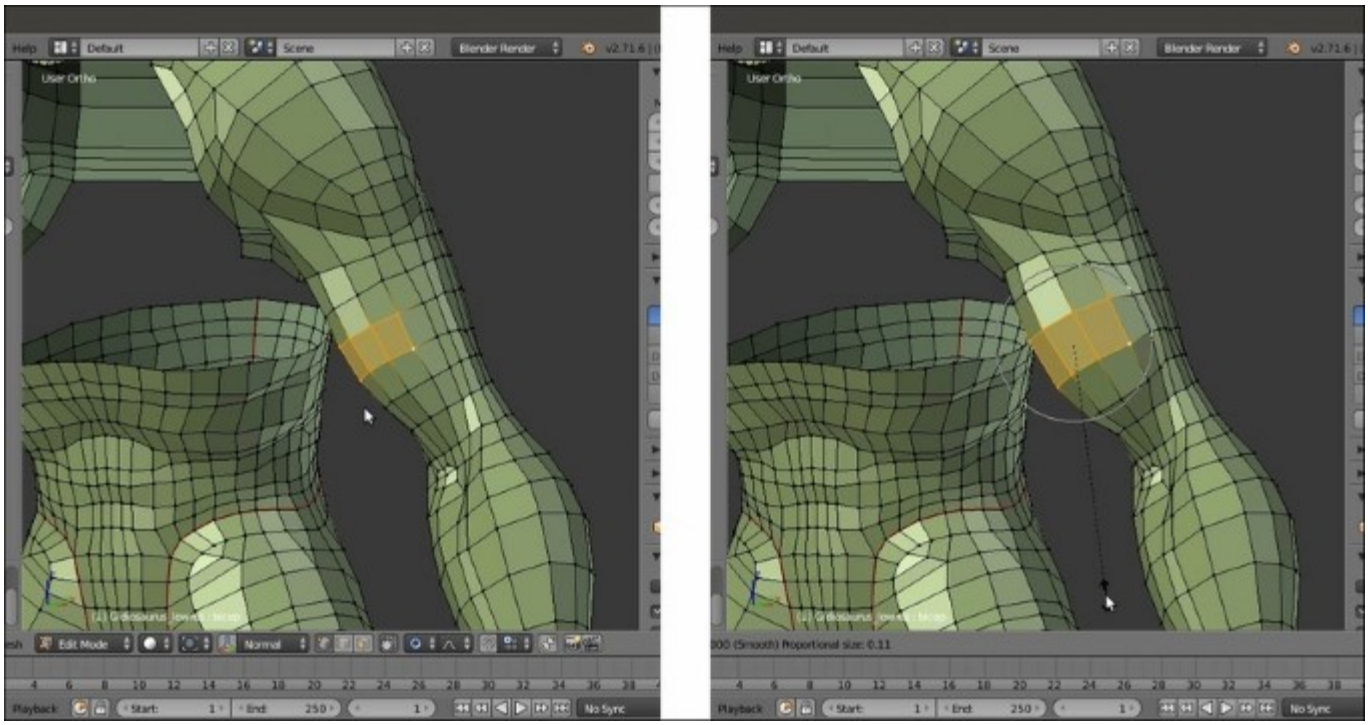
19. Disable the **PET** and adjust the transition between the scaled vertices and the surrounding ones, the area between the two **toes**, and so on.
20. If you wish, you may also make additional modifications; in my case, besides the bigger **feet**, I simply enhanced the **knuckles** on the **hands** and at the **fingers'** joints. Also, I tweaked the rim shape of the upper and bottom borders of the **mandibles** a bit:



Enhancing some of the character's features

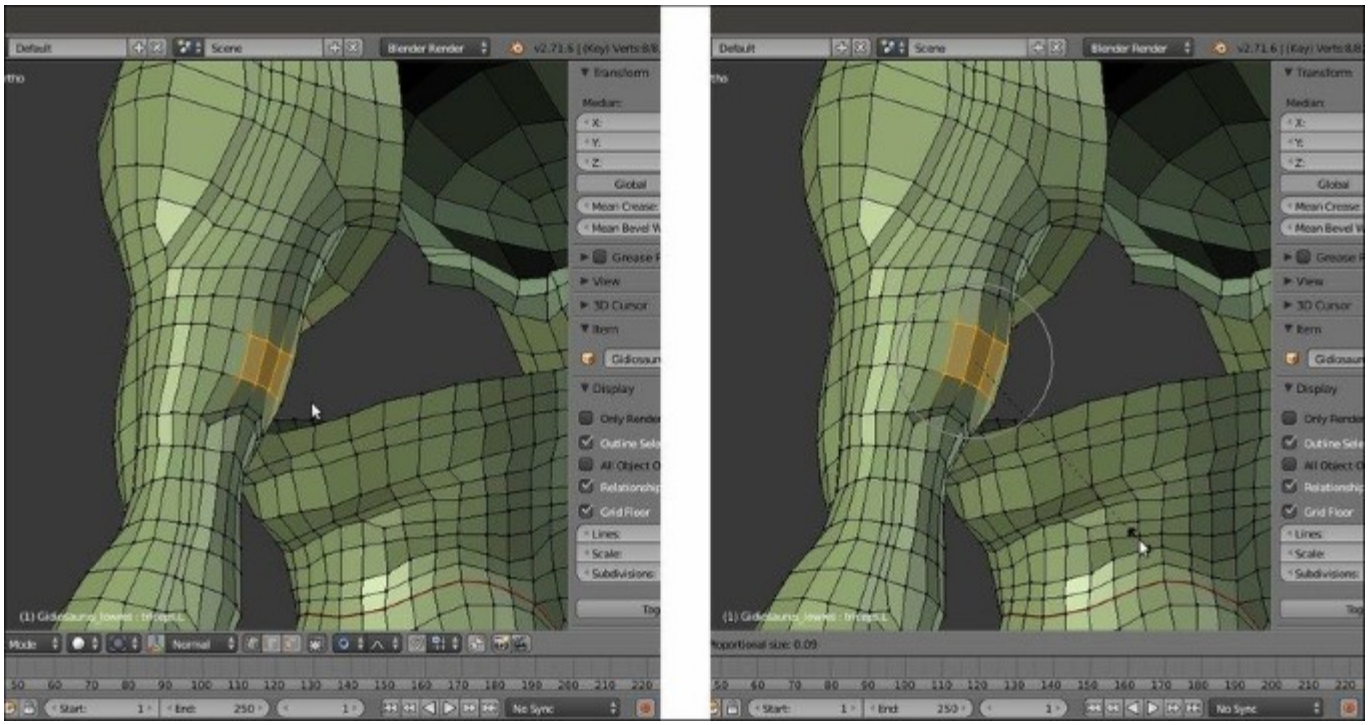
21. When you are done, set the **Value** slider of the **prop** shape key to **1.000**.

    Now, let's add a couple of shape keys to mimic the movement of the main muscles of the arms, specifically of the **biceps** and of the **triceps** muscles:
22. Add a new shape key, then go to the **Gidiosaurus** mesh; select some of the vertices in the middle area of the **bicep** muscle and after enabling the **PET** again, move them forward and also scale them to be bigger, to make the muscle *grow*:



Making the bicep muscle grow

23. Rename the shape key **bicep.L**, and then repeat the steps from 9 to 11 to create the **bicep.R** shape key.
24. Add a new shape key and repeat everything by selecting vertices on the back of the arm, in the **triceps** area, to create the **triceps.L** and the **triceps.R** shape keys.



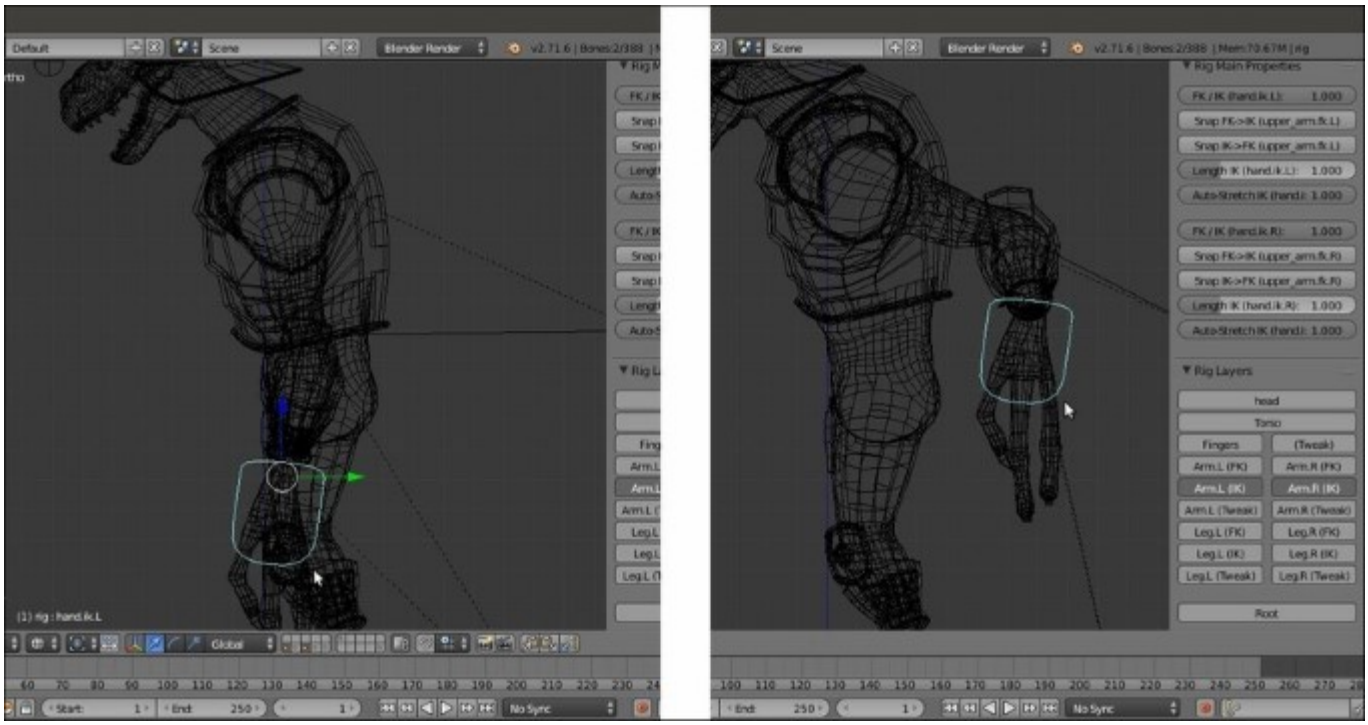
Making the triceps muscle grow

25. Leave the values of these last four shape keys as **0.000** and exit **Edit Mode**.

    More shape keys could be added to simulate a complete muscle system, but in our case we stop here with the **Gidiosaurus** mesh; now let's concentrate on the **Armor**.
26. Go to the **Outliner** and unhide both **Armor** and **rig**.
27. Select the **rig** and then press *N* to call the **Properties** 3D view sidepanel; scroll to the bottom and first go to the **Rig Main Properties** subpanel to set both the slider for **FK/IK (hand.ik.L)** and **(hand.ik.R)** to **1.000**, then go down to the **Rig Layers** subpanel and deselect everything except for the **Arm.L (IK)** and **Arm.R (IK)** buttons.

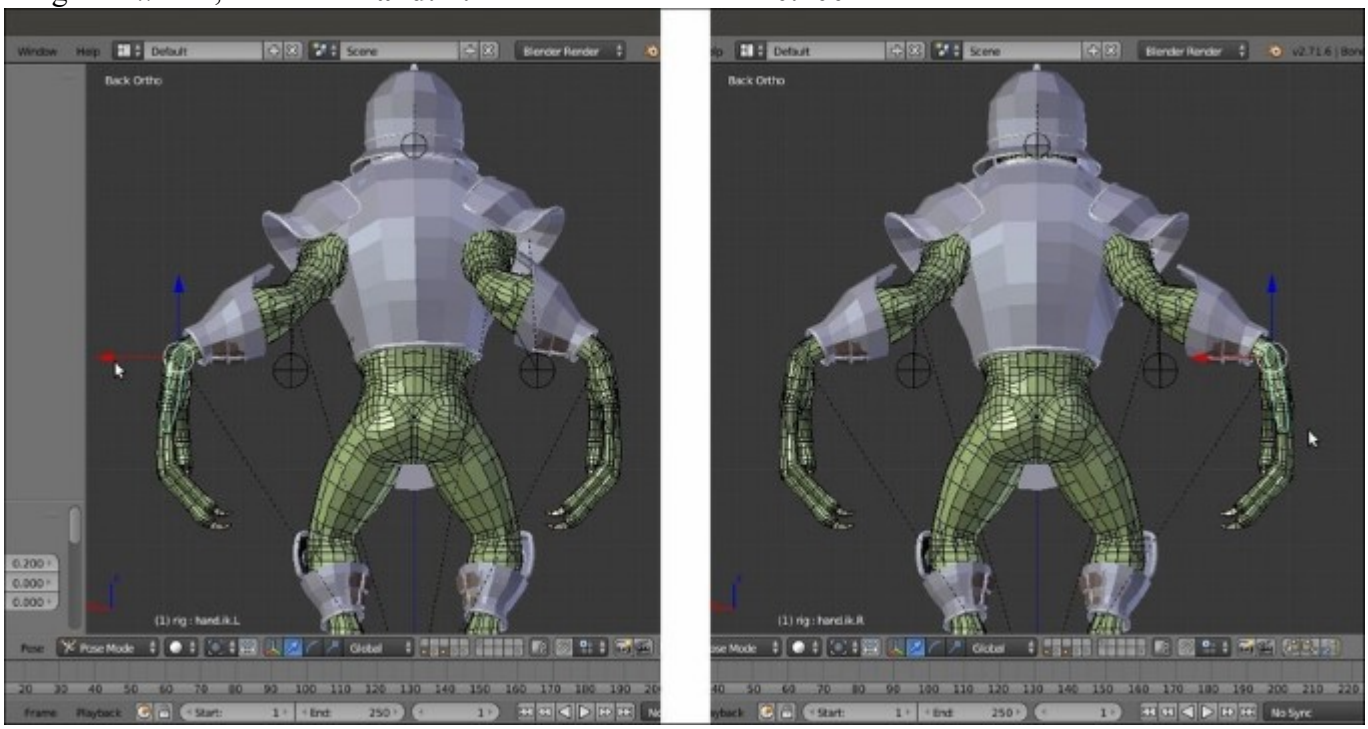
    ## Note

    Note that if the **FK/IK sliders** don't appear, it's because you have to select one of the hand (**ik** or **fk**) bones in the viewport first.
28. Go to the 3D viewport and select the **hand.ik.L** and **hand.ik.R** handle bones, go into the **Side** view and move them towards the upper back.



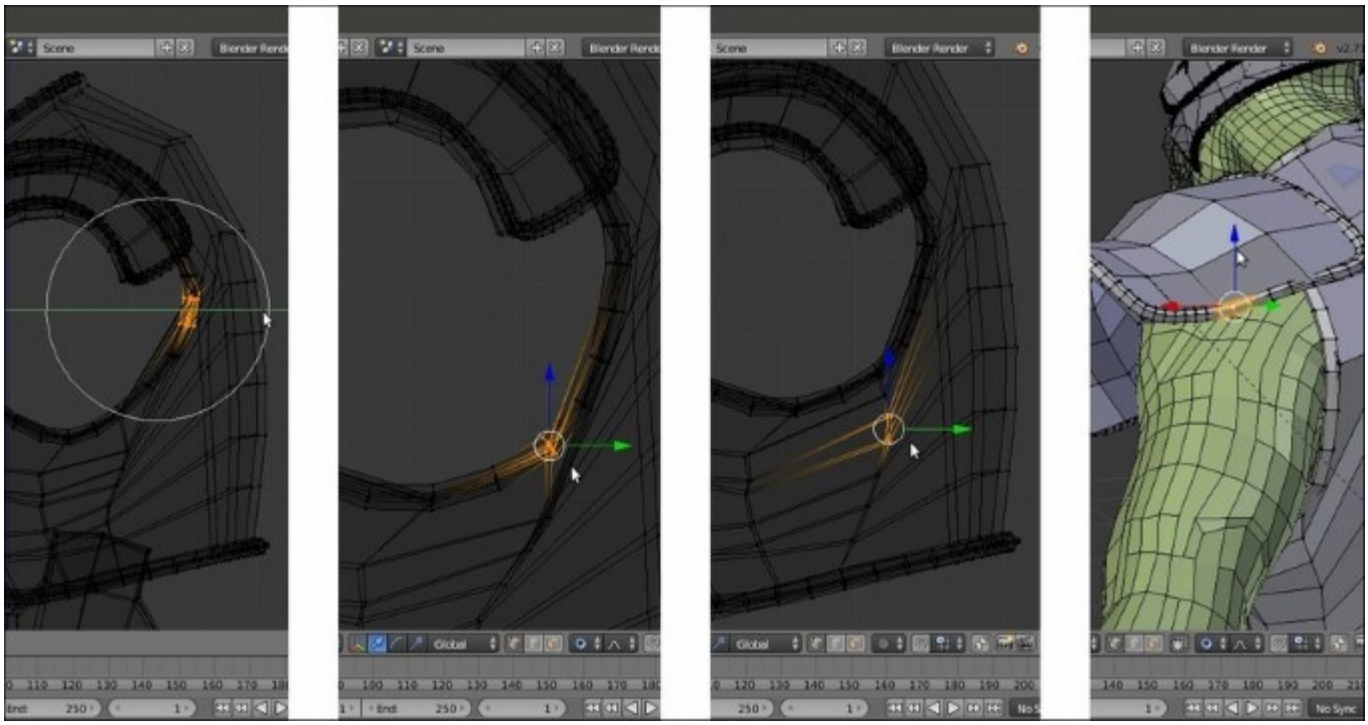
Moving the arm control bones backward using the Inverse Kinematics

29. Now press *Ctrl* + numpad *1* to go in **Back** view and move the **hand.ik.L** bone to **0.200** along the global *x* axis; select the **hand.ik.R** bone and move it to **-0.200**:



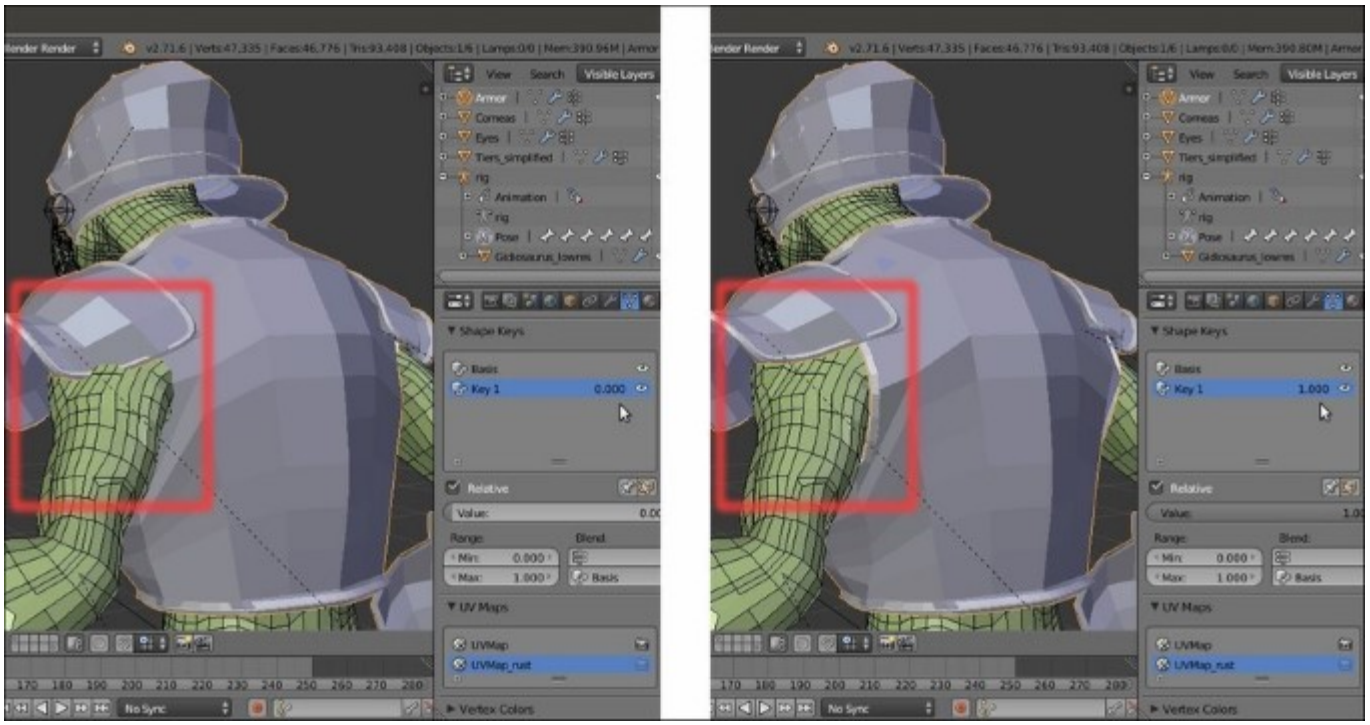
Adjusting the lateral position of the arms

30. Now select the **Armor** object and add the **Basis** shape key in the **Shape Keys** subpanel under the **Object Data** window, and then add **Key 1**.
31. Enter **Edit Mode** and press the slash key (/) on the numpad to go in **Local** view; this way only the selected objects are visible in the viewport, in our case only the **Armor**. Using the **Proportional Editing** mode, start to enlarge the back section of the opening for the **arms**, adjust the position of the surrounding vertices as required, and also raise the back vertices of the **spaulders** a bit, to avoid interpenetration with the borders of the **chest plate** and the **shoulder** as well.
32. Press the numpad slash (/) key again to go out of the **Local** view and check for the correction with the **Gidiosaurus** mesh:



Editing the position of the Armor vertices for a new shape key

33. When you are done, just exit **Edit Mode** and set the **Value** slider of the **Key 1** shape key for the **Armor** to **1.000**:



The shape key working as a fix for the Armor

34. Rename the **Key 1** shape key as **Armor_fix** and save the file.

How it works…

Although technically there are no differences, we could say that we created three different types of **shape key**:

- One type to fix shape errors or make improvements in the mesh, for example, with the **prop** and the **Armor_fix** shape keys
- The second type to modify the mesh only at certain established moments during the animation process, in our case just to animate facial expressions
- The third type to simulate muscle movements in the character

Shape keys work in **linear space**; that means that it's not possible to make vertices rotate around a pivot through a shape key, but only to move them *from point A to point B*. That's why we didn't use shape keys for stuff like the **eyelid** movements, for example, or the opening/closing of the **jaw**, but only for actions including muscles sliding above the bones such as the **eyebrows**, the **grin**, and the **nostrils**, as well as the **bicep/triceps** movements.

Thanks to shape keys, we also made *last minute* modifications and improvements to the **Gidiosaurus** mesh and to the **Armor**; being included inside a shape key, all these modifications are *non-destructive* and can be turned on or off at will, or their influence can be set at an intermediate strength value.

When modifying a mesh using a shape key, be careful not to change too much of the mutual proportions of articulated parts of a *to-be-deformed* mesh; for example, it's usually problematic to scale a whole part such as the **hands** or the **head** of a character, both smaller or bigger, unless you also scale the corresponding bones of the rig and the joints' position accordingly as well.

Beyond a certain threshold, the bones of the **fingers**, or of the **eyes** and **jaw**, start to be *out of register* compared to the respective mesh's edge-loops and you'll have to fix this by re-positioning the joints of the **Armature's** bones (just in case, remember: always do this in **Edit Mode**).

In our example, with the **prop** shape key, we just restricted ourselves to enhance the hands' **knuckles** and to make stronger **feet** by simply making the vertices' positions grow in the direction of their normals.

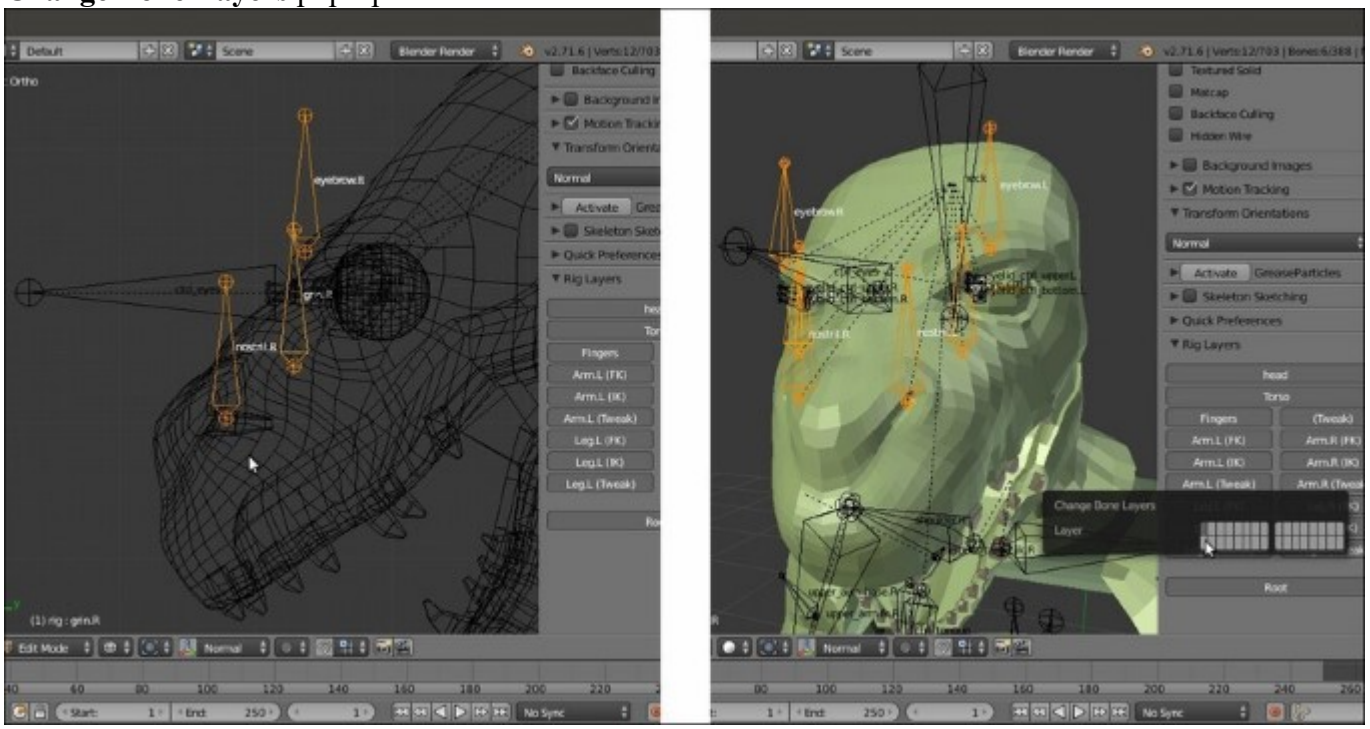
Assigning drivers to the shape keys

In the previous recipe, we created three different types of **shape keys**. Besides the *fixing* shape keys, that have a fixed value (no pun intended), we now need a way to set the amount of influence of the other two types of shape keys, facial expressions, and the muscle movements during the animation. This is accomplished by setting **drivers**, though with different kinds of controls.

Getting ready

Start from the previously saved `Gidiosaurus_shapekeys.blend` file:

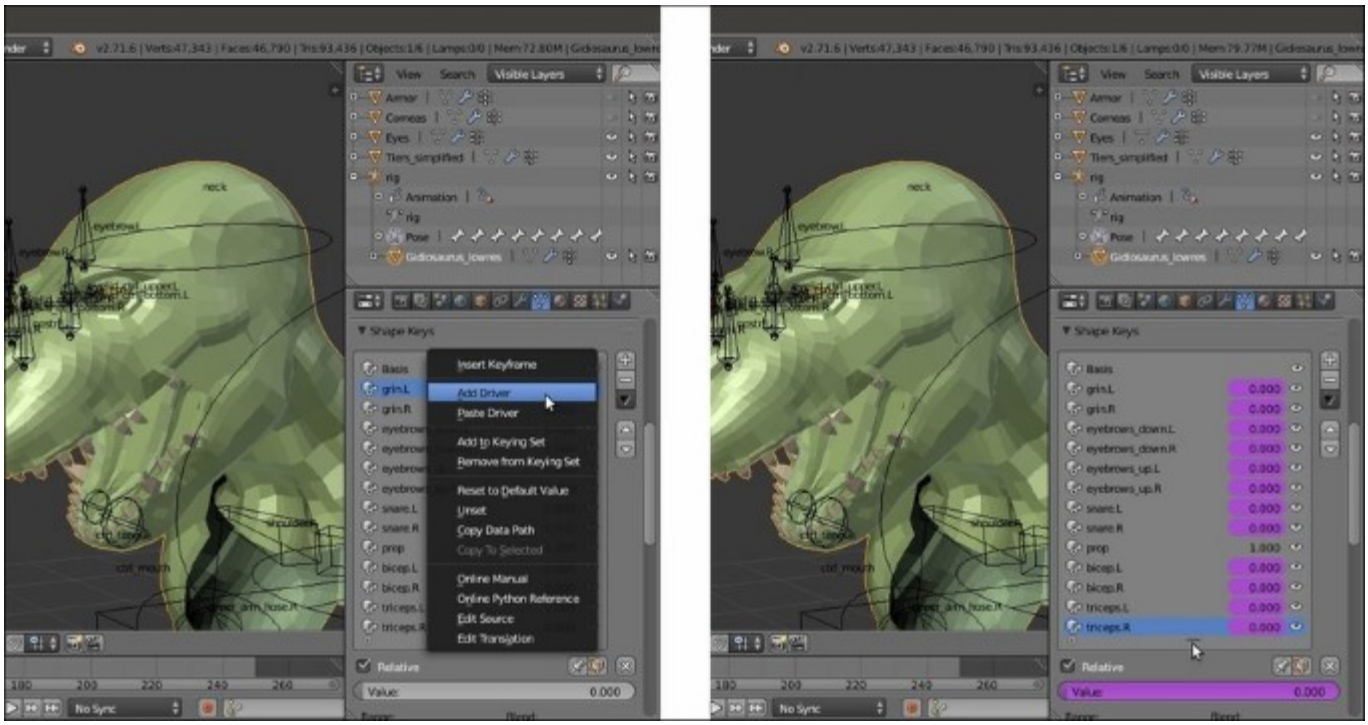
1. Go to the **Outliner** and hide the **Armor** object.
2. Select the **Armature** rig, switch to the **Octahedral** bones draw mode, and press *Tab* to enter **Edit Mode**; zoom to the character **head** and add **six** bones located as follows: **two** bones close to both the right and the left **eyebrows**, **two** bones close to both the sides of the **grin snout** area, and **two** bones close to the **nostrils**. Enable the **Names** item in the **Skeleton** subpanel under the **Object Data** window and rename the bones accordingly and with the correct **.L** or **.R** suffix, then be sure to have them located on the first bone layer by pressing the *M* key to call the **Change Bone Layers** pop-up.



The new bones for the shape key drivers

3. Exit **Edit Mode** and select the **Gidiosaurus** mesh.
4. Go to the **Shape Keys** subpanel under the **Object Data** window and expand the list window by left-clicking on the = icon at the bottom and dragging it downward.

5. Now right-click on the value (**0.000**) at the right side of the name of the first shape key (**grin.L**) and from the pop-up panel, select the **Add Driver** item; the value is enhanced, in violet, to show that now it has a **driver** associated.
6. Repeat the same for all the shape keys in the list except for the **prop** one, which has a fixed value of **1.000**:



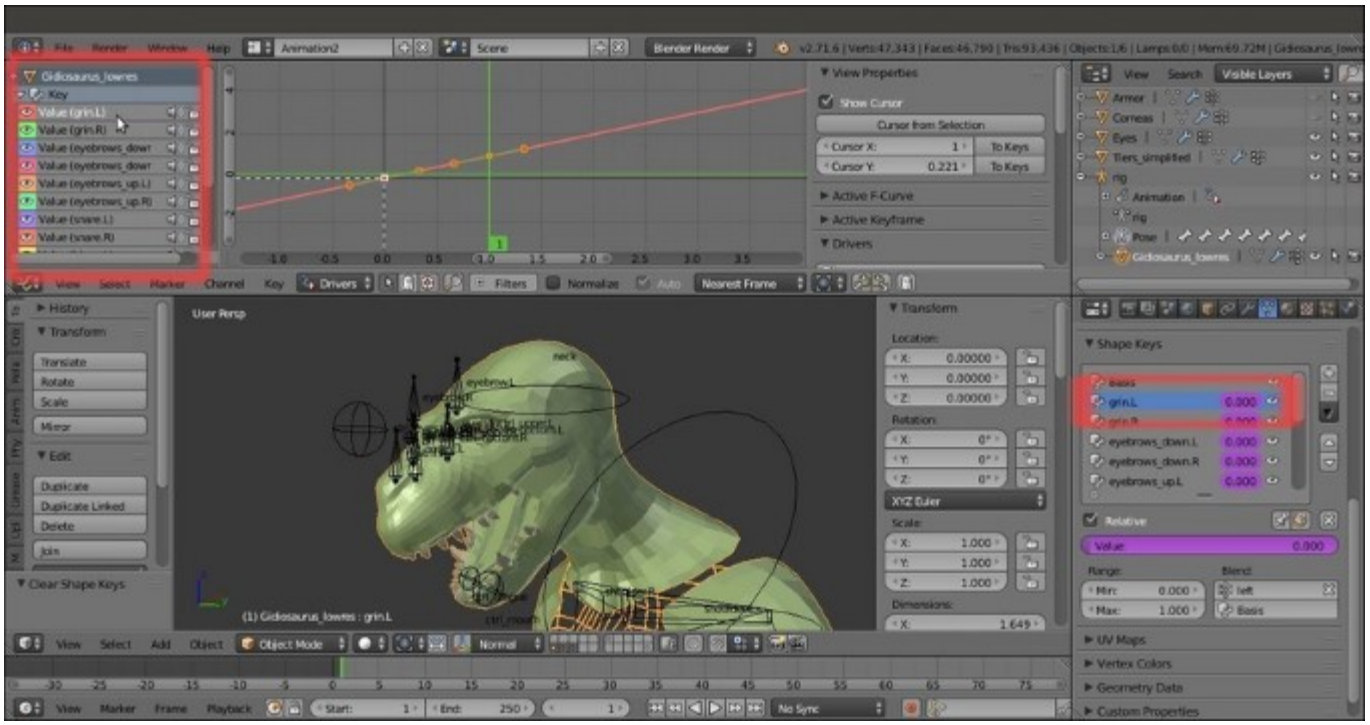
The shape keys list showing they have drivers

7. Split the 3D viewport horizontally into two parts, change the upper part into a **Graph Editor** window, or simply switch the screen to the **Animation** layer (in the **two** files provided with the cookbook, there are actually two prepared animation screens, **Animation1** and **Animation2**). Click on the *Editing context being displayed* button in the toolbar of the **Graph Editor** window and change it from **F-Curves** to **Drivers**.

How to do it…

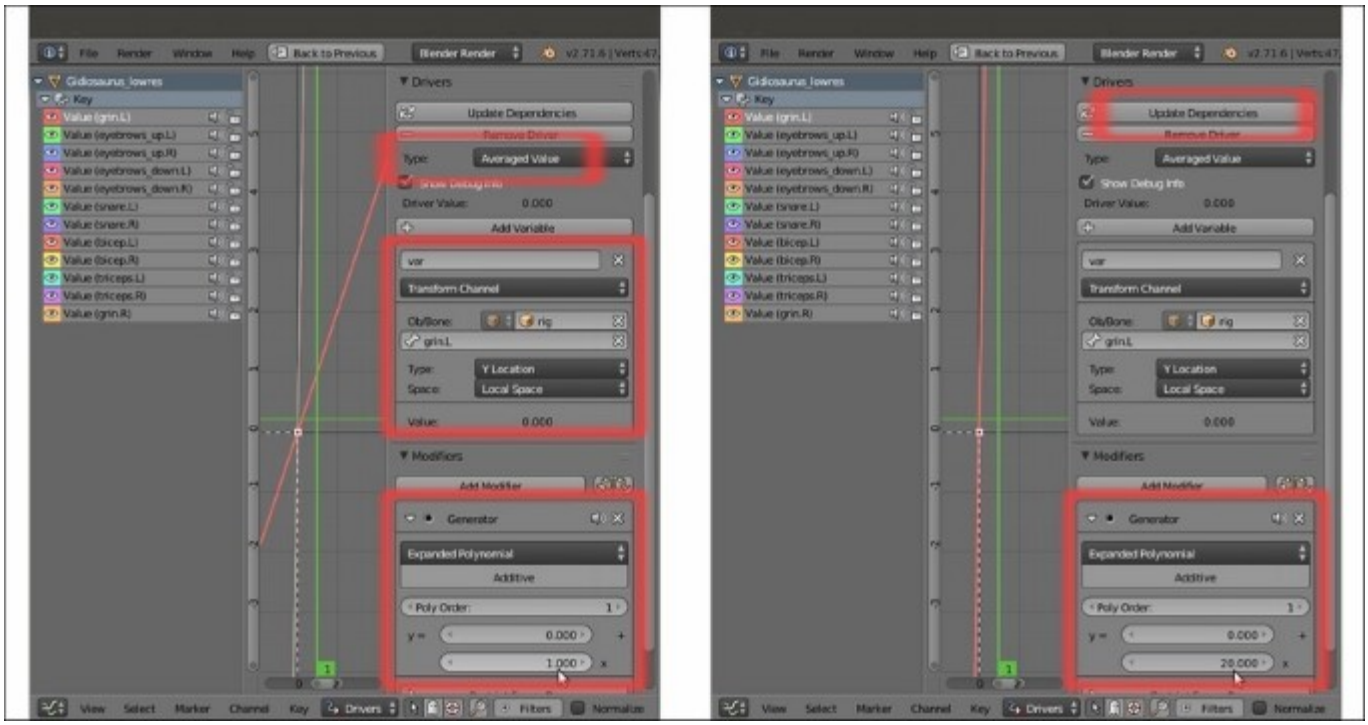
Let's start with the **expressions shape keys**:

1. If not already present, press the *N* key to open the **Properties** sidepanel of the **Graph Editor** window, then click on the **Value (grin.L)** top item in the drivers list at the top-left of the screen:



The Graph Editor window, at the top of the screen, with the driver f-curve

2. Go to the **Properties** panel of the **Graph Editor** and, by scrolling down, find the **Drivers** subpanel. In the *Driver* **Type** slot, switch from the **Scripted Expression** item to the **Averaged Value** one.
3. In the **Ob/Bone** slot, select **rig** and in the under slot (*Name of PoseBone to use as target*), select the **grin.L** bone.
4. Going downward, in the *Variable* **Type** slot, select the **Y Location** item and in **Space**, select **Local Space**.
5. Click on the **Update Dependencies** button at the top of the **Drivers** subpanel (the **Update Dependencies** function works particularly for **Scripted Expression**; it is quite important to use it to refresh the new setups each time).
6. Go even further down and click on the **Add Modifier** button in the **Modifier** subpanel; from the **Add F-Curve Modifier** pop-up menu, select the **Generator** item.
7. In the *Coefficient for polynomial* – **x** slot, change the value **1.000** to the value **20.000** (this is to re-map the declivity of the **f-curve** and therefore the speed of the corresponding shape key):

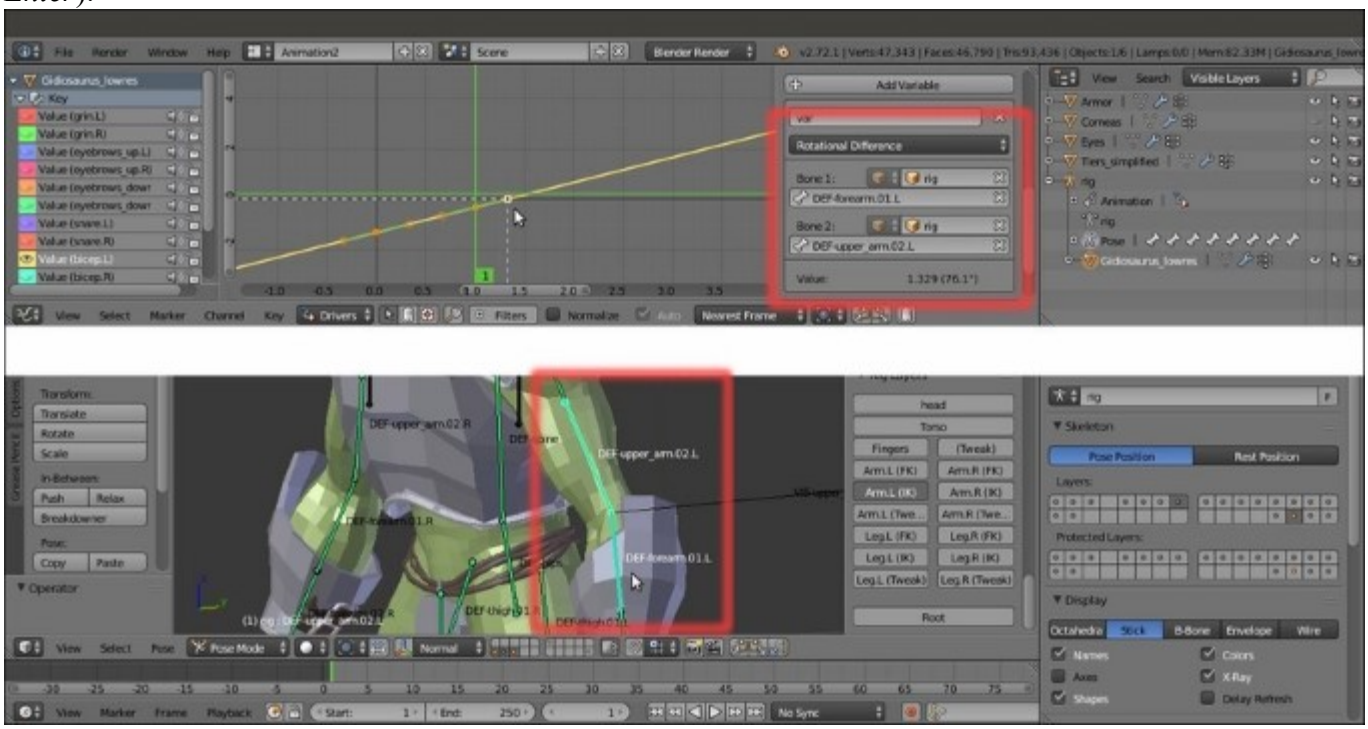


The N Properties Graph Editor sidepanel for the selected driver

8. Now select the **grin.L** bone and in **Pose Mode**, move it upward to see the **grin.L** shape keys being animated on the character's **snout**.
9. Go to the **Shape Keys** subpanel and right-click on the value to the right side of the **grin.L** shape; from the pop-up menu, select the **Copy Driver** item.
10. Select the **grin.R** shape key and right-click on the value to the right; from the pop-up menu, select the **Paste Driver** item.
11. Go to the **Animation** screen and switch the **grin.L** to the **grin.R** bone in the **Ob/Bone** field under the **Drivers** subpanel.
12. Copy and paste the drivers for the **eyebrows_up.L** and **eyebrows_up.R** shape keys, then replace the driver bones names in the **Ob/Bone** field under the **Drivers** subpanel.
13. Go to the **Shape Keys** subpanel under the **Object Data** window and set the **Max** value under the **Range** item to **0.600** for both the **eyebrows_up.L** and **eyebrows_up.R** shape keys; this is to limit the movement of the shape keys to avoid any intersection with the character's **helm**.
14. Copy and paste the drivers for the **eyebrows_down.L** and **eyebrows_down.R** shape keys. This time, leave the same driver bone names and instead change the value of the *Coefficient for polynomial* – **x** to negative and **-20.000** to *invert* the direction of the **f-curve**.
15. Repeat the procedure for the **snare.L** and **snare.R** shape keys, this time switching the *Variable* **Type** from **Y Location** to **X Location** and assigning a negative **-20.000** value to the **snare.L** driver and a positive **20.000** value to the **snare.R** one.

    At this point, all that is left is to assign automatic drivers for the *shape keys to stretch and grow muscles* we created for the character's **arms**.

16. Click on the **Value (bicep.L)** item in the drivers window at the left top of the **Graph Editor** and then go to the **Properties** panel on the right and then to the **Drivers** subpanel. In the *Driver* **Type** slot, select the **Averaged Value** item again; in the *Variable* **Type** slot, switch to **Rotational Difference**.
17. In the **Bone1** slot, select **rig** and in the slot below (*Name of PoseBone to use as target*), select the **DEF-forearm.01.L** bone; in the **Bone2** slot, select **rig** again, and then **DEF-upperarm.02.L**.
18. In the **Graph Editor**, click on a point of the **f-curve** to select it and then press the *L* key to select all the points of the **f-curve**; move them downward, on the *y* axis, by **-1.400** (*G* | *Y* | **-1.4** | *Enter*).



The triceps Rotational Difference driver

19. Copy and paste the driver to the **bicep.R** shape key, then change the **.L** suffixes of the bones to the **.R** ones.
20. Copy the **bicep.L** driver and paste it to the **triceps.L** shape key; click on the **Add Modifier** button under the **Modifier** subpanel; and from the **Add F-Curve Modifier** pop-up menu, select the **Generator** item.
21. In the *Coefficient for polynomial* – **y** slot, write the value **2.300** and in the **x** slot, write the value **-1.000** (remember that all these values in the recipe are not universal and are valid just for this **Gidiosaurus** model in this particular setup; the drivers values could change from character to character, so always test them on your model).
22. Copy and paste to the **triceps.R** shape key, and change the suffixes of the bones.
23. Click on the **Update Dependencies** button at the top of the **Drivers** subpanel and save the file.

How it works…

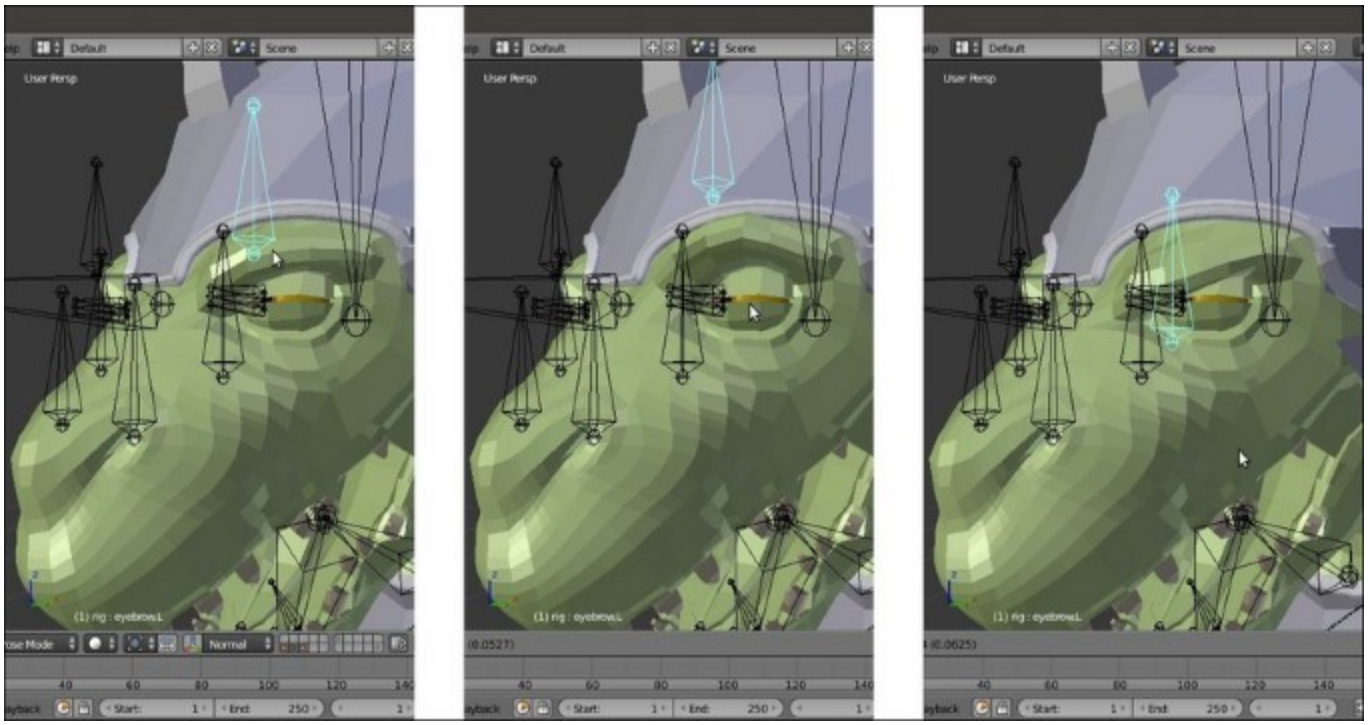
Drivers assigned to bones as controllers for the shape keys are not only an effective way to create a device for animation but also a mandatory technique in the Blender pipeline workflow, where a character is usually linked into the scene from a different file and the rig gets **proxified** (we'll see how to do this in the next chapter). The only possible way to have access to the shape keys in a linked character is through the **drivers** and the **rig**.

As you probably already know, shape keys are often used not only for **facial expressions** but also to mimic the stretching and the growing of the body's **muscles** according to the movement of a character's **limbs**. In this case, their influence is automatically driven by the rotation of the respective bones through the **Rotational Difference** drivers that, as the name itself says, base their influence on the difference of rotation between two bones; more precisely, on the **angle** between them.

The **Generator** modifier we added is a multiplier we used to *virtually* modify the slope inclination of the **f-curves** of the drivers. The default inclination of the **f-curve** wasn't enough to fully map the curve itself to a driver bone movement of (almost) just one or two Blender Units (it was too slow, resulting in a required driver movement of several units to have an appreciable effect), so we increased the declivity by a factor of **20.000** to have a faster correspondence.

However, the same modifier was also used to reverse the direction of the **f-curve**, by using a negative value of **-20.000**, for example to drive the downward movement of the **eyebrows**, or to change the location of the curve along the *y* axis so as to tweak the timing of the driver influence, like in the **triceps** shape keys.

Therefore, by copying and pasting a driver and giving an opposite declivity at the slope of the copied one, it is possible to drive two opposite shape keys through the same bone, as for the **eyebrows** shape keys:

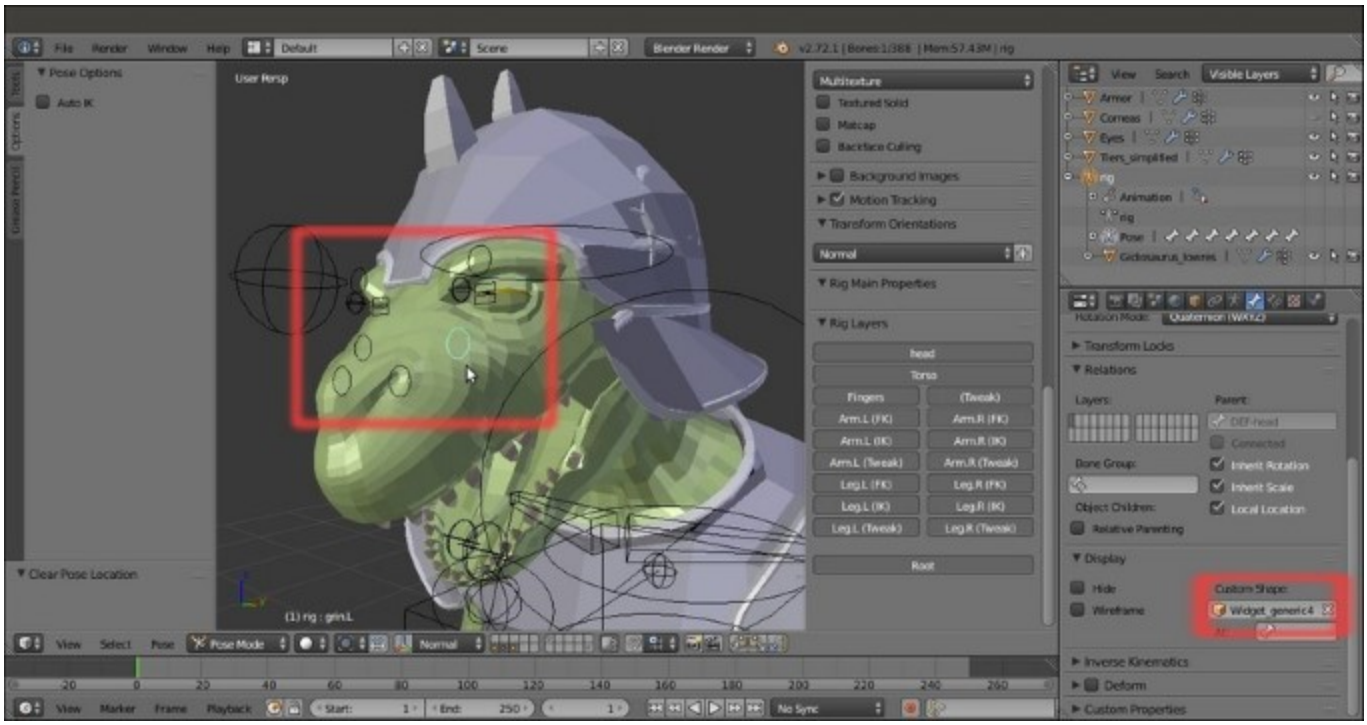


The same bone moving in two opposite directions to drive two opposite shape keys

There's more…

To add shape keys and the respective drivers to the **Gidiosaurus** model, we used the `Gidiosaurus_skinning_rigify.blend` file, with the **rig** created by the **Rigify** addon. The control bones of a **Rigify** rig have pre-made **Custom Shapes** to make their identification and selection easier and are usually located in the last scene layer.

So, for the last step, I just modeled a new simple custom shape, a small **Circle** mesh with **16** vertices. I named it **Widget_generic4** and I assigned it to all the *driver* bones:



The driver bones with the new Custom Shape

See also

- http://www.blender.org/manual/animation/basics/drivers.html

Setting movement limit constraints

Often, it is very useful to put movement limitations on the bones of a rig, for several reasons—usually, to make them easier to work with, but also to establish a maximum range for the rotation of the **limbs** or other parts like in the **mandible** or the **eyelids**.

Two types of limits for the bones are: by the **Transform** locks, and by **bone constraints**.

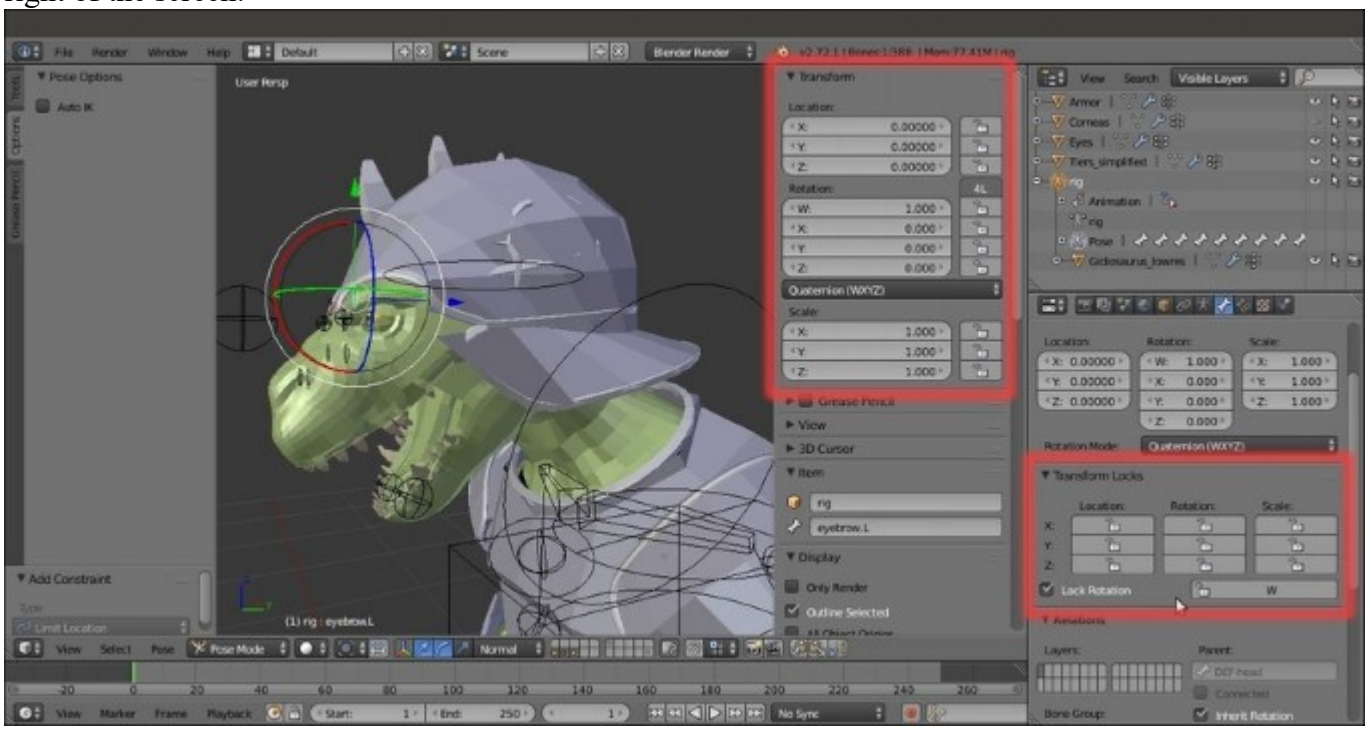
Getting ready

Load the `Gidiosaurus_shapekeys.blend` file, select the **Armature**, and go in **Pose Mode**.

How to do it…

Let's start with the **Transform** locks:

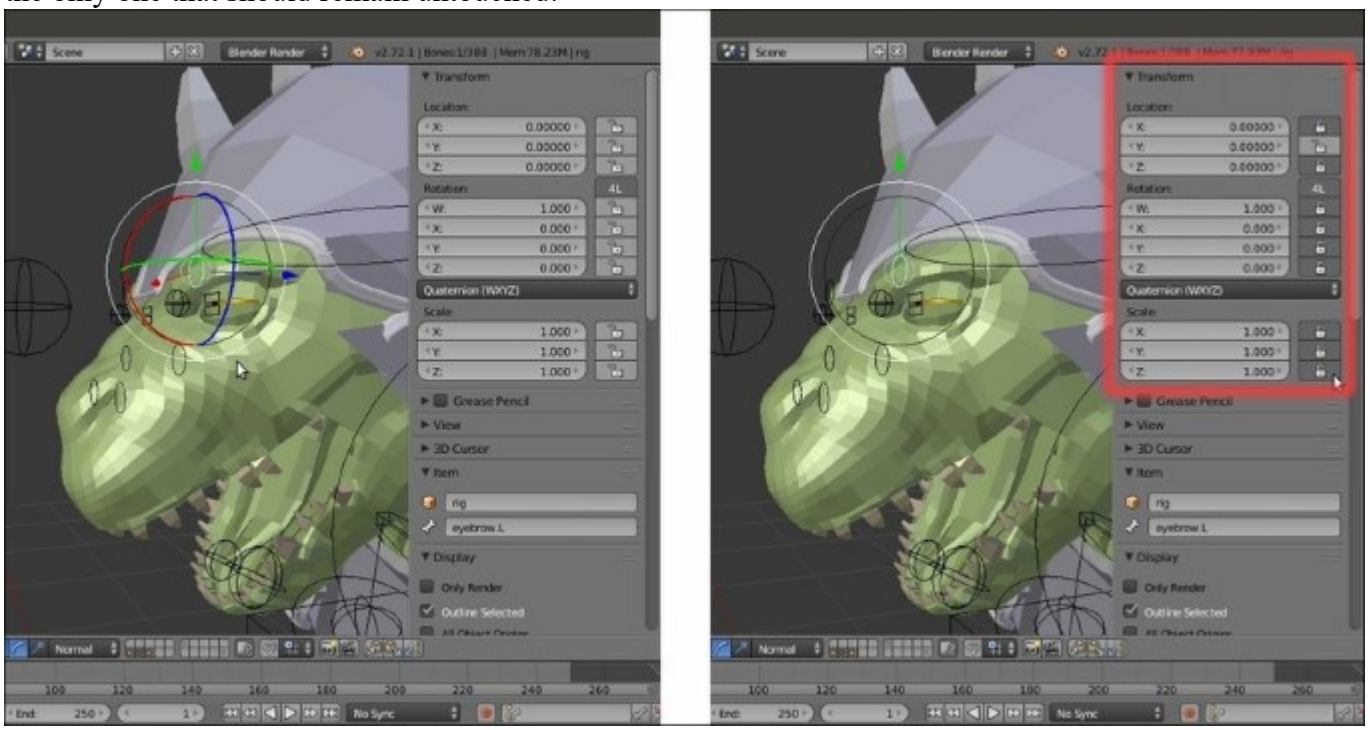
1. Select the **eyebrow.L** bone and if not already present, press the *N* key to call the 3D viewport **Properties** sidepanel. Go to the **Transform** subpanel, which is the first entry at the top, or also to the **Transform Locks** subpanel under the **Bone** window in the main **Properties** panel to the right of the screen:



The Transform subpanel in the N Properties sidepanel and the corresponding Transform Locks subpanel under the main Properties panel

2. Click on the lock icon to the right side of the properties; for this bone (which, if you recall, is the driver control object for the left up and down **eyebrow** shape keys), we want to lock all the

possible transformations *except* for the movement on its *y* axis, so the **Location Y** lock button is the only one that should remain untouched:



Setting the axis Transform locks for the Location, Rotation, and Scale

If you now try to move the **eyebrow.L** bone, you will notice its movement is constrained only to its local *y* axis; the movement is directed by the **Roll** orientation of the bone in **Edit Mode** (and not by the **Normal** item enabled in the **Transform Orientation** button on the viewport toolbar); enable the **Axes** item in the **Display** subpanel under the **Object Data** window to see this.

Having locked the other two axes, it's no longer necessary to use the widget arrow to move the bone on its local *y* axis but it's enough to simply press the *G* key and then move the mouse instead.

And now, let's see limits by **constraints**.

3. Go to the **Bone Constraints** window and assign a **Limit Location** constraint to the **eyebrow.L** bone.
4. Check the **Minimum X** and **Maximum X**, **Minimum Y**, and **Maximum Y**, and **Minimum Z** and **Maximum Z** items. Leave the values for the *x* and *z* axes as they are, change **Minimum Y** to negative **-0.050**, and change **Maximum Y** to positive **0.050** (again, remember that these values are valid just for this file).
5. In the **Convert** slot, change the **Owner Space** item to **Local Space**:

The assigned Limit Location constraint subpanel under the main Properties panel

In Chapter 1, *Modeling the Character's Base Mesh*, we enabled the **Copy Attributes Menu** add-on in **User Preferences** and then we saved the **User Settings**, so I'm taking for granted that you have the script still enabled.

Therefore, we do the following:

6. Select the **eyebrow.R** bone and then *Shift*-select the **eyebrow.L** bone. Press *Ctrl* + *C* and from the **Copy Attributes** pop-up menu, select the **Copy Bone Constraints** item.
7. Select the **grin.L** and **grin.R** bones and then *Shift*-select the **eyebrow.L** bone. Once again, press *Ctrl* + *C* | **Copy Bone Constraints**, and in the two copied constraints, set the **Minimum Y** value to **0.000**.
8. Select the **nostril.L** and **.R** bones and *Shift*-select the **grin.L** bone, then press *Ctrl* + *C* | **Copy Bone Constraints**. This time, set both the **Y** values to **0.000** and **Minimum X** for the **nostril.L** bone to negative **-0.050** and the **Maximum X** for the **nostril.R** bone to positive **0.050**.
9. Save the file.

Several other movement constraints have been added to different bones in the rig, for example the **jaw** bone, or the **eyelid** controllers, but especially to the **eye** bones, to limit the range of possible rotations. To have a look at the various settings, just open the `Gidiosaurus_limits.blend` file.

See also

- http://www.blender.org/manual/animation/techs/object/constraint.html

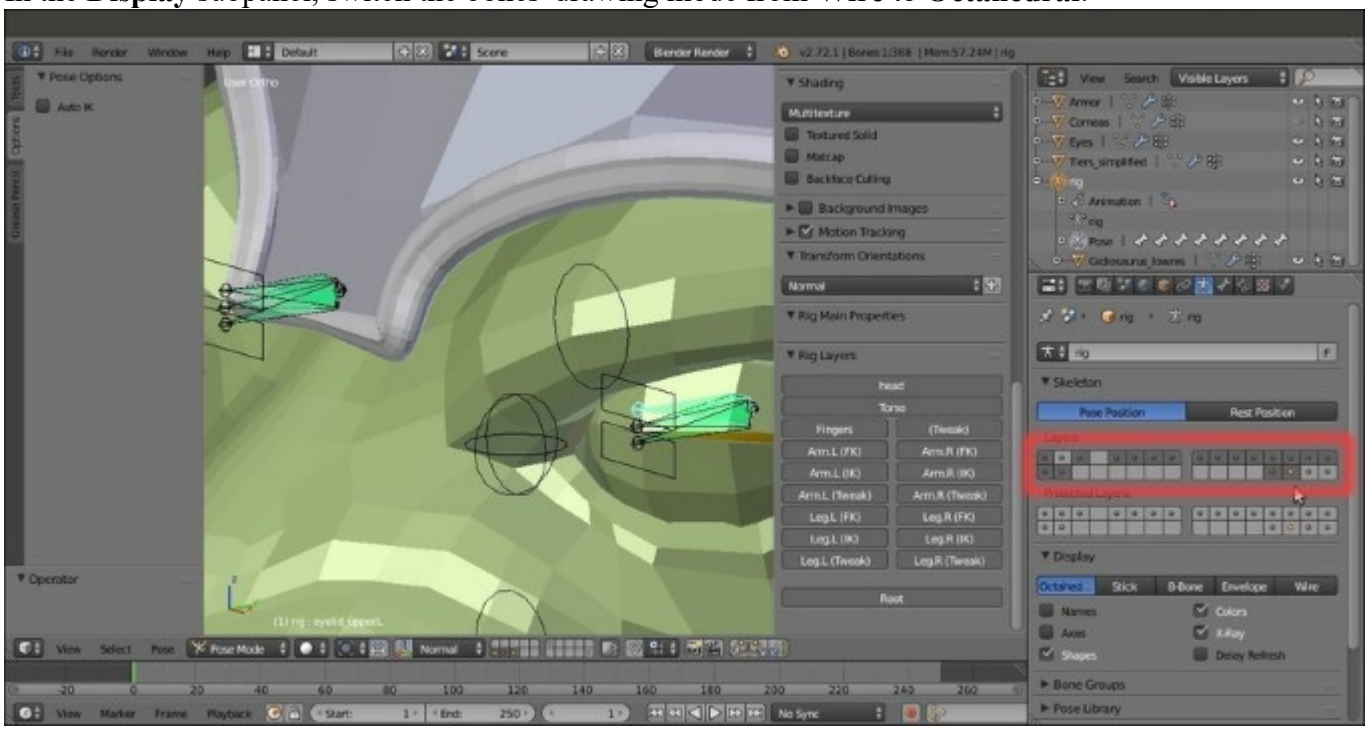
Transferring the eyeball rotation to the eyelids

This is a really simple trick that can add a lot of life to the facial expressions of an animated model, making the **eyelids** follow some of the movement of the **eyeballs**.

Getting ready

Following on from the previous recipes, open the `Gidiosaurus_limits.blend` file:

1. If not already selected, select the **Armature** and enter **Pose Mode**.
2. In the **Object Data** window, go to the **Skeleton** subpanel and enable the **30th** bone layer, to show the deforming bones.
3. In the **Display** subpanel, switch the bones' drawing mode from **Wire** to **Octahedral**:



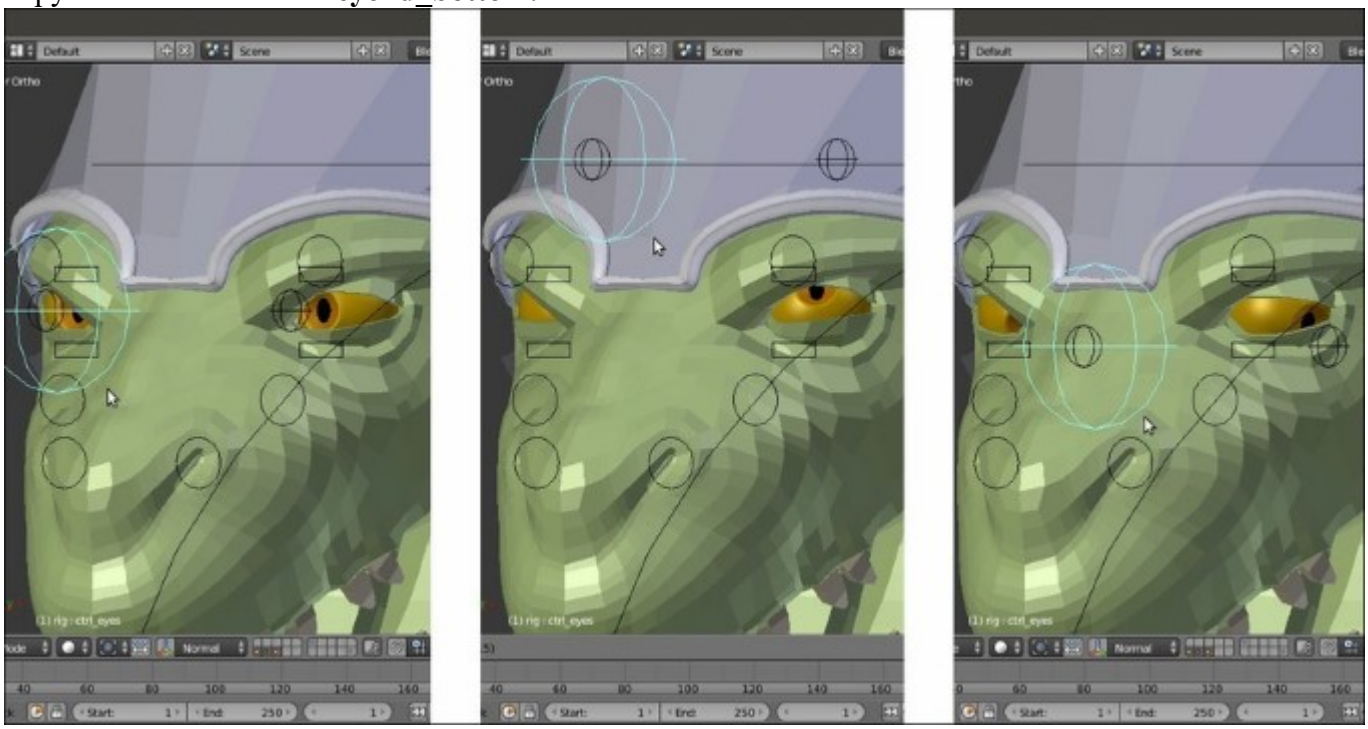
The Skeleton subpanel with the bone layers

How to do it…

Now zoom to the character's **head** and continue with the following steps:

1. Select the **eyelid_upper.L** bone and go to the **Bone Constraints** window; assign a **Copy Rotation** constraint.
2. In the **Target** field, select the **rig** item, and in the **Bone** field, select the **eye.L** bone item. Set **Space** = **Pose Space** to **Pose Space**.
3. Set the **Influence** slider value to **0.300**.

4. Select the **eyelid_bottom.L** bone and *Shift*-select the **eyelid_upper.L** bone, then press *Ctrl* + *C* | **Copy Bone Constraint**.
5. Select the **eyelid_upper.R** bone and repeat the procedure but with **eye.R** as the target bone; copy the constraint to the **eyelid_bottom.R** bone:



The eyelids slightly following the eye movements

6. Save the file.

Detailing the Armor by using the Curve from Mesh tool

In Chapter 3, *Polygonal Modeling of the Character's Accessories*, in the *Using the Mesh to Curve technique to add details* recipe, you already saw how to use this technique as a modeling tool. In this recipe, we'll use the same technique but in the opposite direction—to add **rivets** around the perimeter of the borders of the different **Armor** parts.

Getting ready

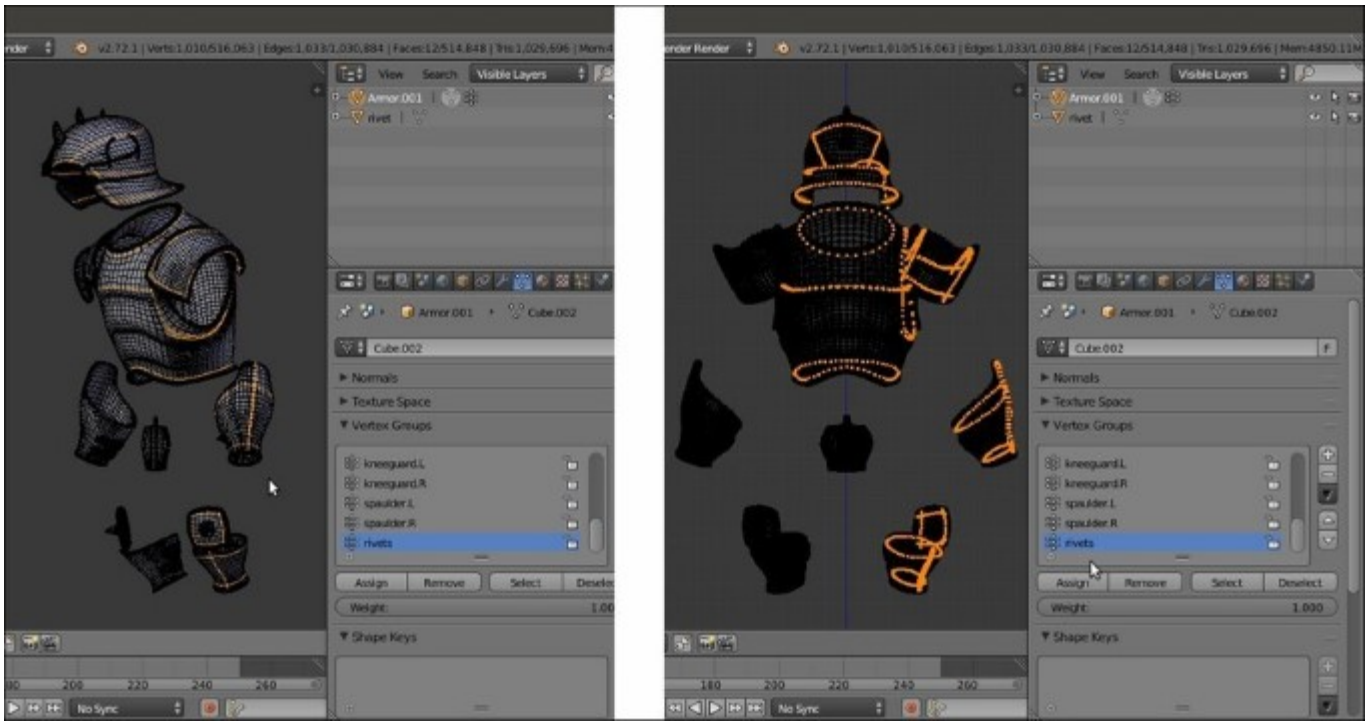
Re-open the `Gidiosaurus_limits.blend` file; the first thing to do is to model a very lowpoly **rivet** object to be duplicated on the **Armor** surface:

1. Switch to an empty scene layer, press *Shift* + *C* to place the **3D Cursor** at the center of the grid, and add a **Cube** primitive mesh. Enter **Edit Mode** and delete the bottom face, then scale the remaining faces by a value of **0.100** twice, then one last time by **0.500**. Move the top face downward to flatten the overall shape a bit and scale the same face by **0.700**.
2. Press *A* to select all the vertices and *W* to choose the **Subdivide Smooth** item from the **Specials** pop-up menu, then delete the middle horizontal edgeloop.
3. Put the pivot on the **3D Cursor** and while still in **Edit Mode**, rotate all the vertices by **90°** on the *x* axis.
4. Select the bottom edgeloop and press *Shift* + *S* | **Cursor to Selected**. Exit **Edit Mode** and click on the **Set Origin** button under the **Tool** tab to select the **Origin to 3D Cursor** item.
5. Click on the **Smooth** button under the **Shading** item and in the **Outliner**, rename the **rivet** object. Once again, place the **3D Cursor** at the center of the grid and the **rivet** at the **Cursor** location; press *Ctrl* + *A* to apply the **Rotation & Scale** option.
6. Enable the scene layer with the **Armor** on it, and in the **Outliner**, hide the **rig**.

How to do it…

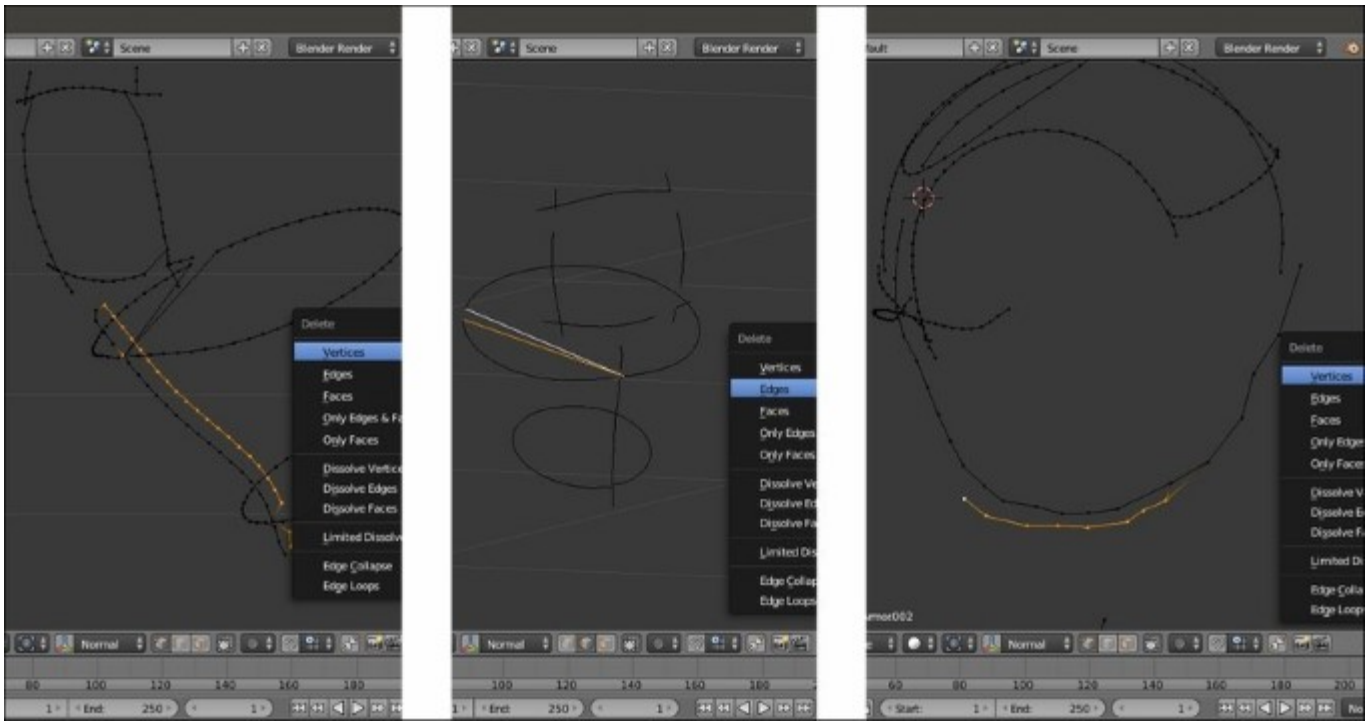
Now, let's create the *guides* to duplicate the rivets on:

1. Select the **Armor** object and press *Shift* + *D* to duplicate it, then place the duplicate **Armor.001** object on the scene layer of the **rivet**. Go to the **Shape Keys** sidepanel under the **Object Data** window and delete the **Armor_fix** first and then the **Basis** shape keys.
2. Go to the **Object Modifiers** window, remove the **Armature** modifier, and apply the **Subdivision Surface** modifier with a **Subdivision** level of **2**.
3. Enter **Edit Mode** and start to select the edgeloops on the different **Armor** parts in areas where you want to add the **rivet** rows (*Alt* + right-click for the first one, then *Alt* + *Shift* + right-click). As usual, it's enough to work only on one half of the mesh:



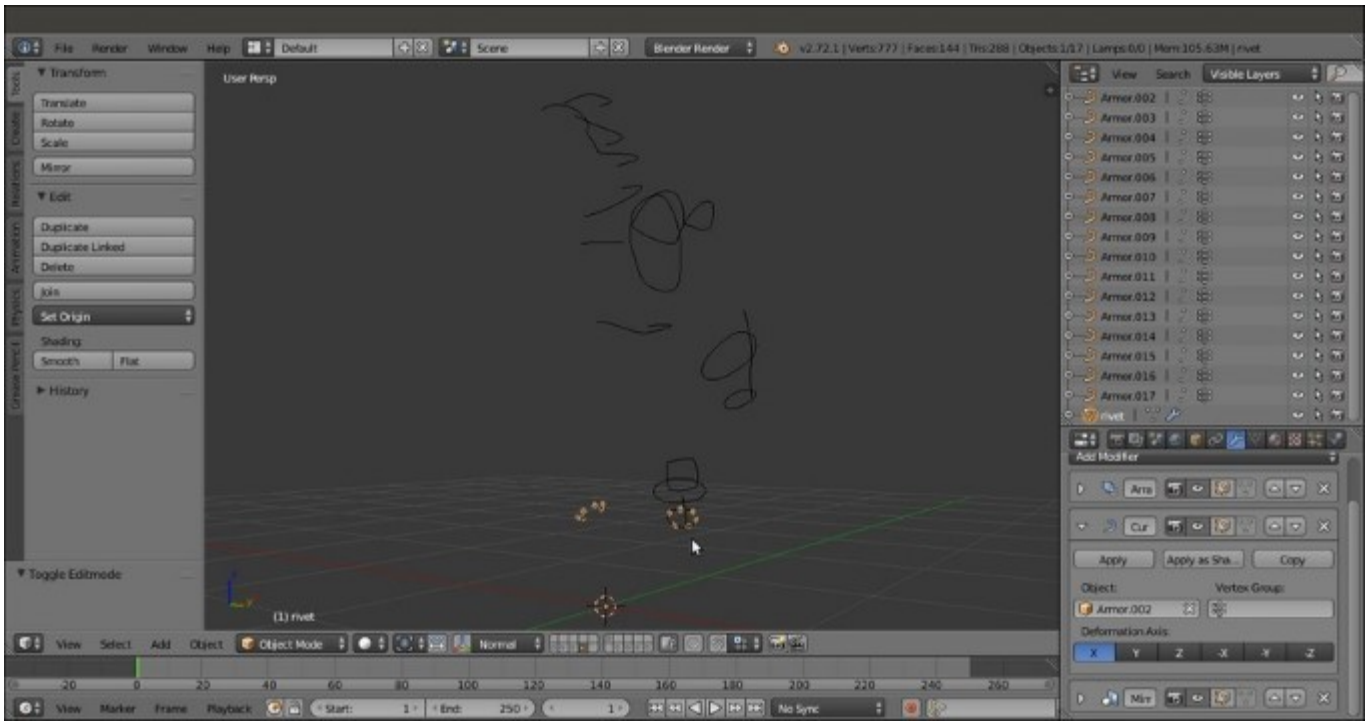
The Armor mesh in Edit Mode with the selected edge-loops

4. Press *Shift* + *D* and soon after, click the right mouse button to duplicate the selected edgeloops without moving them, then press the *P* key to separate them from the **Armor.001** object (in the **Separate** pop-up menu, choose the **Selection** item).
5. Exit **Edit Mode** and delete the **Armor.001** object, or if you don't have problems with big file sizes, move it to a different scene layer to keep it for future refinements. In this case, you can save the edge-loops selection as a vertex group named **rivets**.
6. Select the **Armor.002** object (the duplicated and separated edgeloops) and enter **Edit Mode**; make the necessary adjustments to the edgeloops by deleting the unnecessary vertices, for example the backsides of the plates, and disconnect the welded edgeloops by deleting the common vertices or connecting them where required edges are missing:



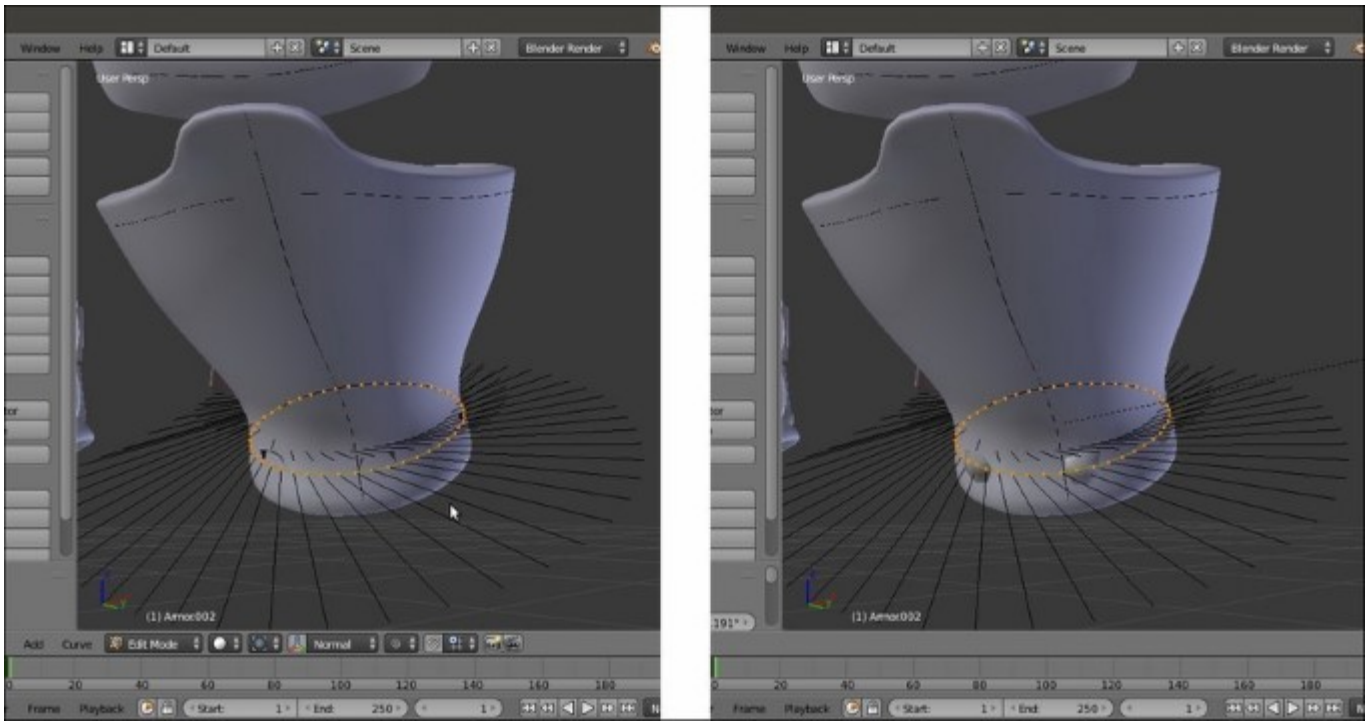
Cleaning the edge-loops of the duplicated Armor.002 mesh

7. Press *A* to select all the vertices and then go to the **Tools** tab under the **Tool Shelf**. Go to the **LoopTools** subpanel and press the **Space** button to evenly space the vertices along the edgeloops.
8. Exit **Edit Mode** and press *Alt* + *C*; in the **Convert to** pop-up menu, select the first item, **Curve from Mesh/Text**. The mesh edgeloops actually get converted into a **Curve** object, as you can see in the **Object Data** window under the main **Properties** panel to the right of the UI. Click on the **Fill** slot to select the **Full** item.
9. Now the tedious part (but not difficult, just a little tedious); in **Edit Mode** again, put the mouse on one of the points and by pressing the *L* key, select each separate part of the **Curve**, then press *P* to separate the whole selected part. This way, you are going to obtain **16** separated **Curve** objects.
10. Select the **rivet** object and go to the **Object Modifiers** window; assign an **Array** modifier with **Fit Type** = **Fit Length**, **Length** = **0.50**, and **Relative Offset X** = **3.000**. Collapse the panel.
11. Assign a **Curve** modifier, then in the **Object** field select the **Armor.002** curve. Leave the panel expanded.
12. Assign a **Mirror** modifier and collapse the panel:



The rivet object instanced on the mirrored curve object

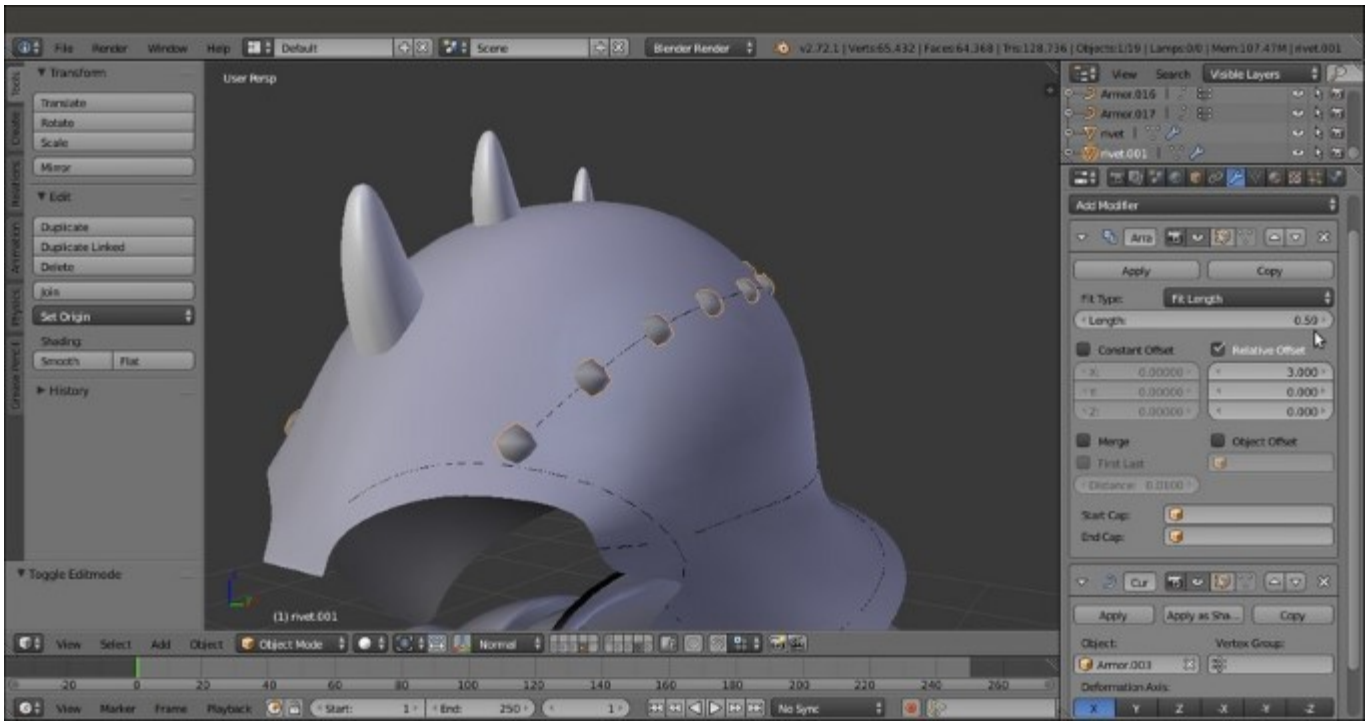
13. In the viewport area, zoom to each curve to check for the correct tilting of the points; if necessary, select the curve, enter **Edit Mode**, select all the points, press *Ctrl* + *T*, and move the mouse to rotate the tilting of the curve's points until the instanced **rivets** are correctly rotated/ aligned with the surface of the main **Armor** mesh:



Tilting the curve's points

If necessary, you can also select individual points of the curve to tweak the orientation of only a part of the instanced rivets, even of single rivets at once; this has been done for part of the **helm** and for the **spaulders**, especially.

14. In the **Outliner**, re-select the **rivet** and press *Shift* + *D* to duplicate it, then in the **Object Modifiers** window, under the **Curve** modifier panel, select the **Armor.003** item in the **Object** field.
15. Once again, zoom to the curve and if necessary, fix the curve tilting and also adjust the **Length** value of the **Array** modifier (for the **Armor.003** curve it has been raised to **0.59**) and the **Relative Offset** value. By selecting all the points and pressing *W*, you can also select the **Switch Direction** item in the **Specials** menu.



The rivets on the helm object

16. Duplicate the rivet and repeat the procedure changing the curve name in the modifier for each curve object and so on. At the end, you should have **16** copies of the rivet as well.

    At this point, if required, we can still make some modification to the rivet mesh; in my case, I just subdivided it a bit more, then deleted some useless edgeloop, made it rounder, and extruded the open side a bit more.

    Now that the rivet is ready, select all the rivet copies (so that the *modified one* is the active object, that is *the last selected*) and press *Ctrl + L* | **Object Data** to share the modifications between them.

    Leaving everything selected, press *U* | **Object & Data** to make them single users again (this is necessary for the next step with the modifiers).
17. When you are done, select all the rivets *one at a time* in the **Outliner** and apply all the **Array** and the **Curve** modifiers.
18. Join all the rivets into a single object (select all and press *Ctrl + J*) and in **Edit Mode**, delete the unnecessary or overlapping ones, keeping only the rivets that really add to the **Armor** look. Then, apply all the **Mirror** modifiers:

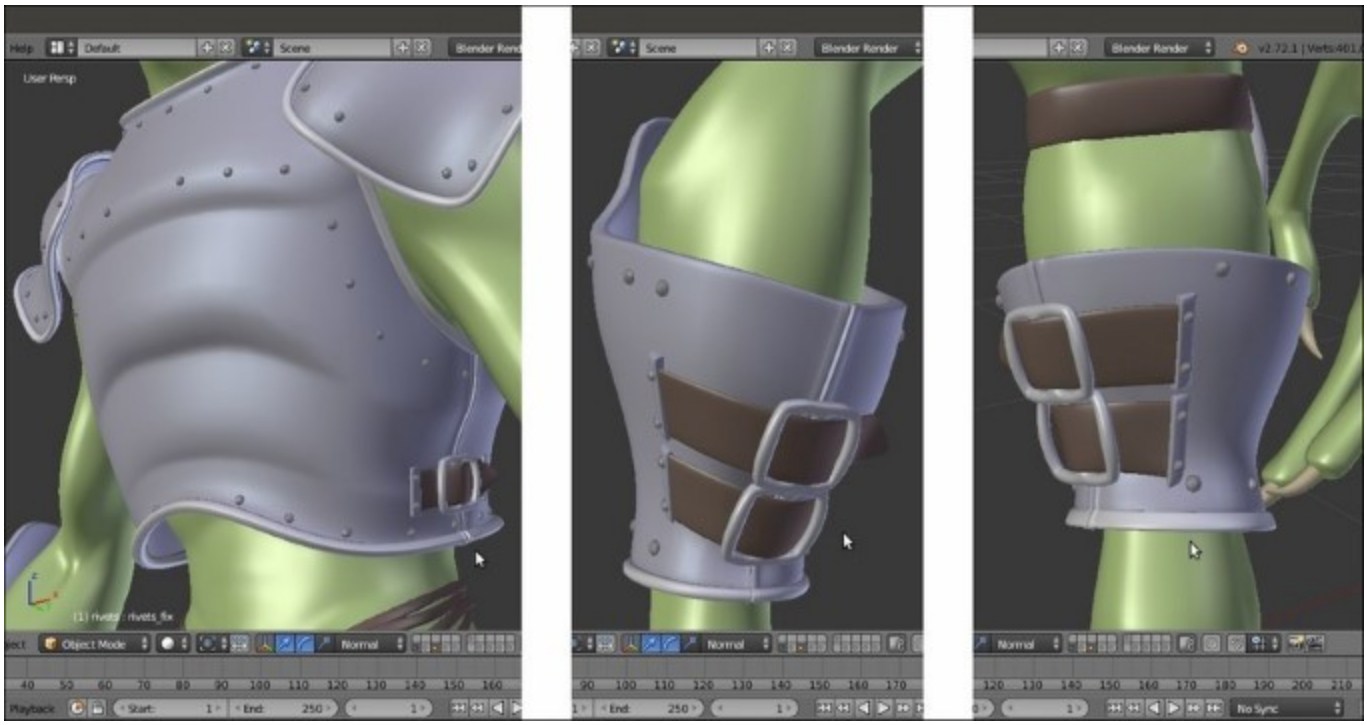
The completed rivets

19. Select the **Armor** object and then *Shift*-select the rivets object, press *Ctrl* + *Tab* to go in **Weight Paint** mode and click on the **Transfer Weights** button under the **Tools** tab.
20. Exit **Weight Paint** mode and assign an **Armature** modifier to the **rivets** object, select **rig** in the **Object** field.
21. Save the file as `Gidiosaurus_final_detailing.blend`.

There's more…

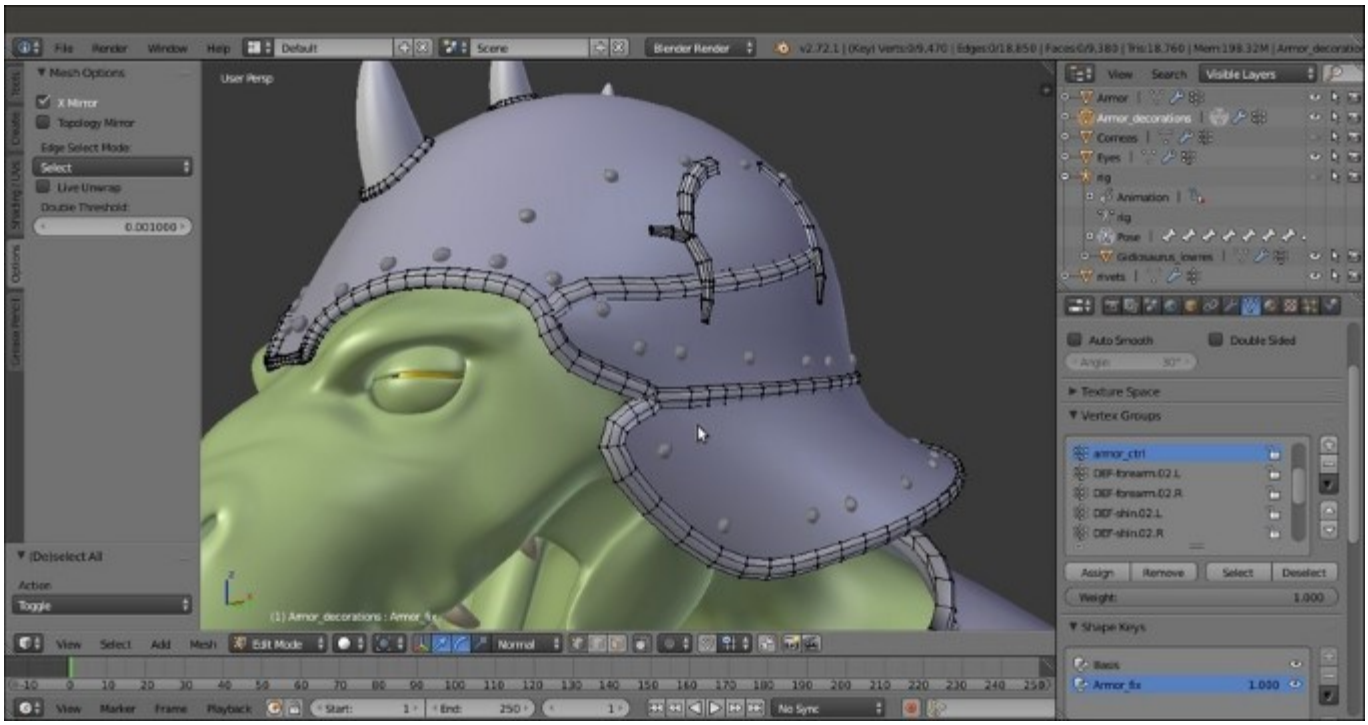
At this point, the **Gidiosaurus** model is ready to be animated, but some minor adjustments are still missing and can be added.

I won't go into the details about these additions, they are all processes you have already seen in the previous chapters and recipes, so this is simply a showcase:



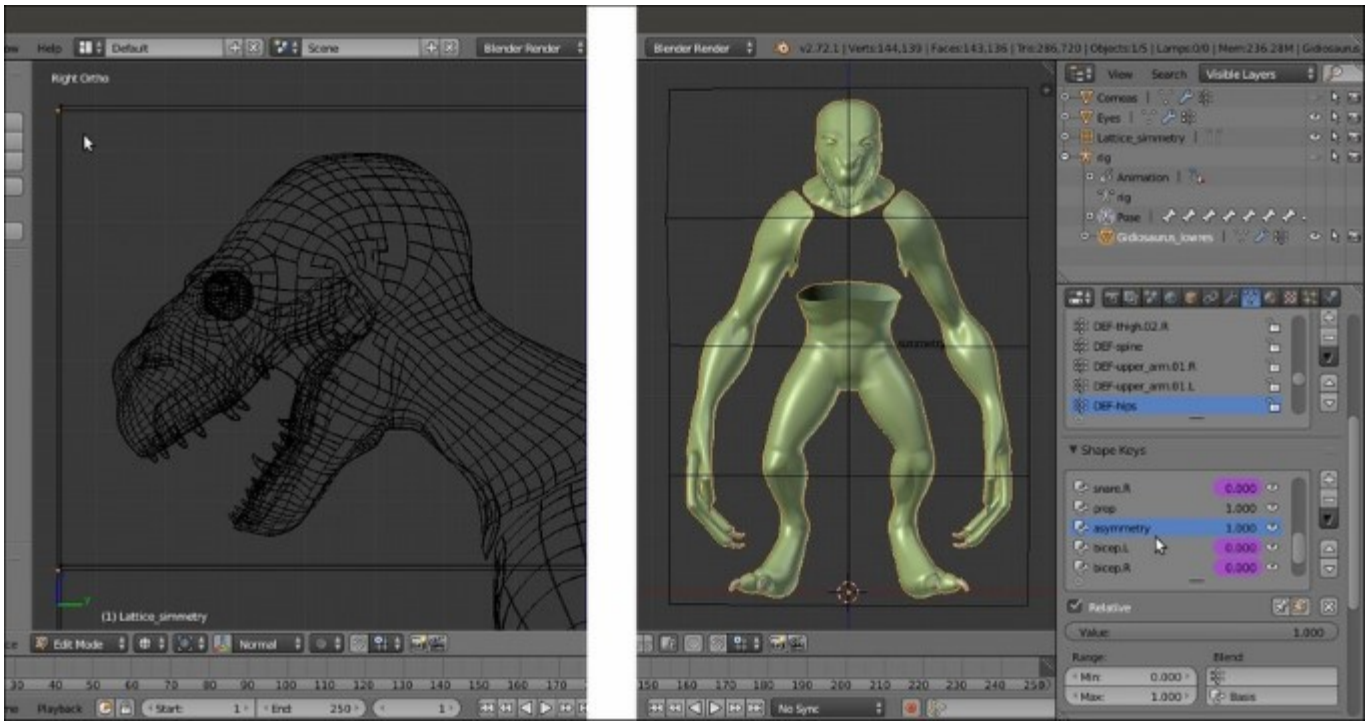
The modeled tiers and the rivets

1. The tier attachments on the **Armor's vambraces** and on the **greaves** have been refined by adding smaller **rivets**, and new tiers have been added to the sides of the **Armor chest plate**. Also, the opening seams in the **Armor** parts have been modeled under each tier location.
2. The **Armor** decorations have been separated as a new object (the **Armor_decorations** item in the **Outliner**) and simplified by deleting as many edgeloops as possible without altering their basic shape:



The simplified decorations in Edit Mode

3. The **Armor's** shape also has been tweaked even further through the **Armor_fix** shape key to adjust some overlaps that were occurring during the movements of the **spaulders** and in the **stomach** area too. The same shape key has been repeated also on the **decorations** and on the **rivets** objects for the areas of interest.
4. A bit of asymmetry has been introduced in the **Gidiosaurus** mesh by assigning a **Lattice** modifier to the character, and slightly modifying the shape on the left side, then applying the modifier as a shape key (the **Apply as Shape Key** button):



The asymmetry lattice

5. Finally, after some test renders, I realized that the **teeth** and the inside of the **mouth** of the **Gidiosaurus** still needed refinements, so I made some more adjustments to the **prop** shape key by making the **teeth** bigger and bolder, and the **inner mouth** more organic-looking and smooth:

The modified teeth and inner mouth

Be aware that almost in every modeled object there is still room for improvement, and that's okay, it's not a sign of a bad job! This sort of improvement is done all the time and is simply part of the working experience.

See also

- http://www.blender.org/manual/modeling/curves/index.html

Chapter 9. Animating the Character

In this chapter, we will cover the following recipes:

- Linking the character and making a proxy
- Creating a simple walk cycle for the character by assigning keys to the bones
- Tweaking the actions in Graph Editor
- Using the Non Linear Action Editor to mix different actions

Introduction

There are literally a *plethora* of tutorials and manuals about animation principles in general, and in Blender in particular, on the Web and in bookstores, so this one is going to be just a very *easy* chapter, mainly about the **technical aspects** of creating a simple animation with the rigged **Gidiosaurus** character, following the most usual pipeline commonly used in Blender (at least for the **open movies**).

Linking the character and making a proxy

The habit of linking assets from library files is the most useful and used, I would say, not only in a Blender based workflow, but also in the *industry*. A linked asset, in our case a creature character, can be placed and animated even if not already completed in all its parts, thus it allows a team to work almost *at the same time* on the different aspects. In our case, the **Gidiosaurus** is still missing **texturing** and **shaders**, but can already be placed *on stage* and animated anyway.

To link an asset in Blender and keep the possibility of animating it through a rig, we must make a proxy of the **rig** itself. A **proxy object** overrides the animation controls of a linked object in a non-destructive way, so that an animator can animate it locally to the `.blend` file the rigged character has been linked to. This way, the linked character object retains all its original information and is *only locally* altered by the proxy object scene.

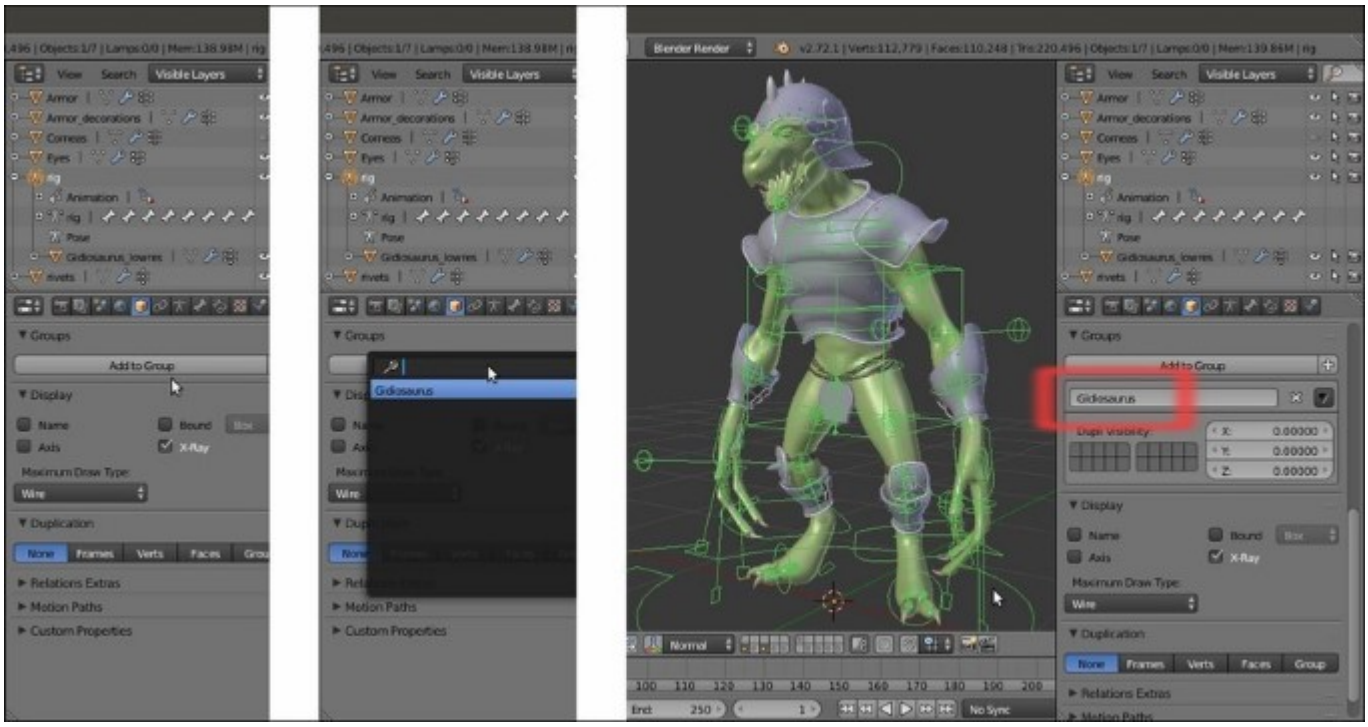
Getting ready

As the first thing, we must prepare the **library**, so open the `Gidiosaurus_final_detailing.blend` file:

1. Go to the **Outliner** and select the **Gidiosaurus_lowres** mesh, then also *Shift*-select the **Armor**, the **Armor_decorations**, the **rivets**, the **Eyes**, and the **Corneas** objects.
2. Press *Ctrl* + *G*, and all the selected objects are outlined in green to show that now they belong to a **group**, in this case, to the same group we created just now.
3. Go to the **Object** window and in the **Groups** subpanel, change the generic default **Group** name to **Gidiosaurus**.

Creating a Group and assigning all the selected objects to it

4. Go to the **Outliner** and click on the eye icon to the side of the **rig** item to make it visible again, and then click on the **rig** item itself to select it.
5. Press *Ctrl* + *Tab* to go out of **Pose Mode** and go to the **Groups** subpanel under the **Object** window again. Click on the **Add to Group** button and in the pop-up menu, select the **Gidiosaurus** item (in this case, the only group already created). The **rig** is outlined in green as well:



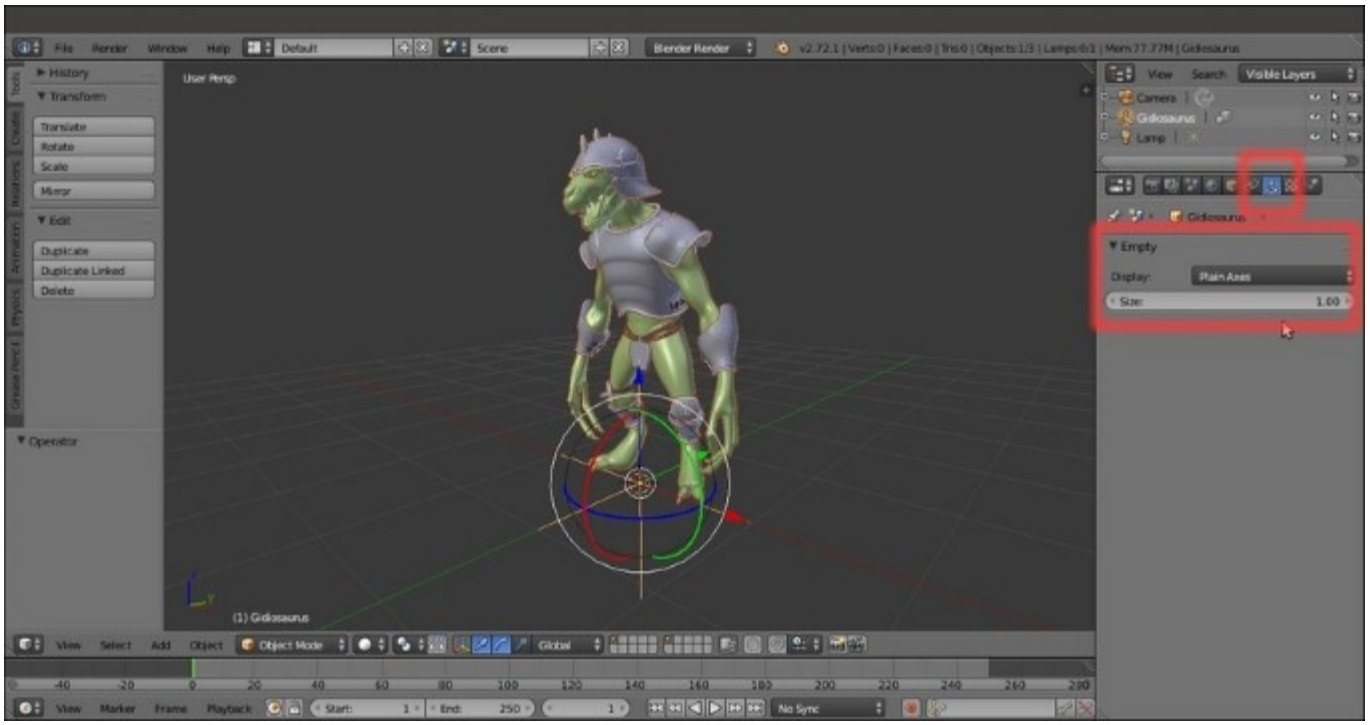
The rig assigned to the group as well

6. Click again on the *Restrict view-port visibility* button (the one with the eye icon) to the side of the **rig** item to hide it and save the file as `Gidiosaurus_library.blend`.

How to do it…

1. Click on the **File** item in the main menu bar, select the **New** item, and confirm by clicking on the **Reload Start-Up File** pop-up (or just press *Ctrl* + *N*).
2. Select the default **Cube** and delete it, then go to **File** | **Link** (or press *Ctrl* + *Alt* + *O*). Browse and click on the `Gidiosaurus_library.blend` file, then click on the **Group** folder item, and finally click on the **Gidiosaurus** item. Click on the **Link from Library** button to the top right of the screen.

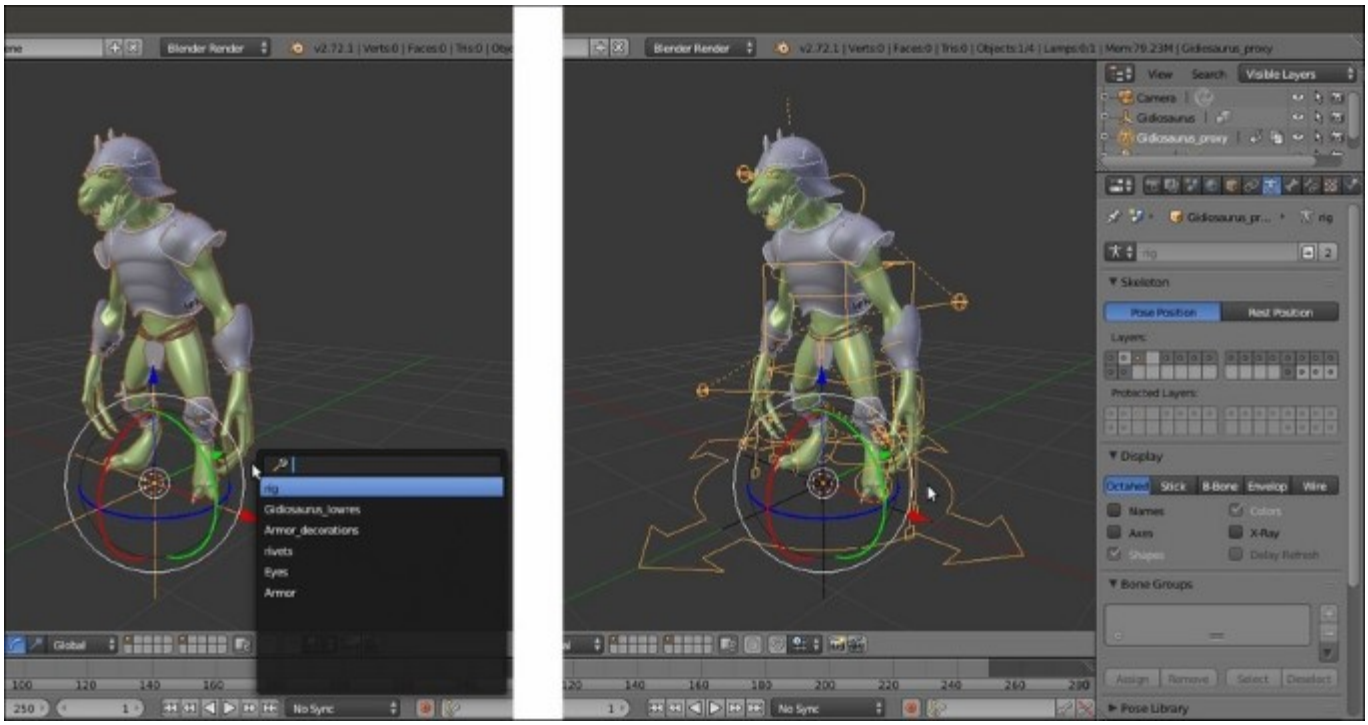
   A new object has appeared at the **3D Cursor** location (that should be placed at the center of the scene), and what we have got at this point is the **linked Gidiosaurus group**; this means that the character and any other object inside the **Gidiosaurus** group in the library file are now linked and *instanced* on an **Empty** that is named **Gidiosaurus** as well:



The Gidiosaurus group linked and instanced on the Empty

Remember that in the library file, inside the **Gidiosaurus** group we put also the **rig**, which for the moment is not visible in the linked group because it is *hidden in the library file*.

3. Press *Ctrl* + *Alt* + *P*, and a new pop-up appears where we can select the item we want to *proxify* (although all the objects inside the group appear in the list, at the moment only an **Armature** can be proxified). Click on the **rig** item:



The proxified rig

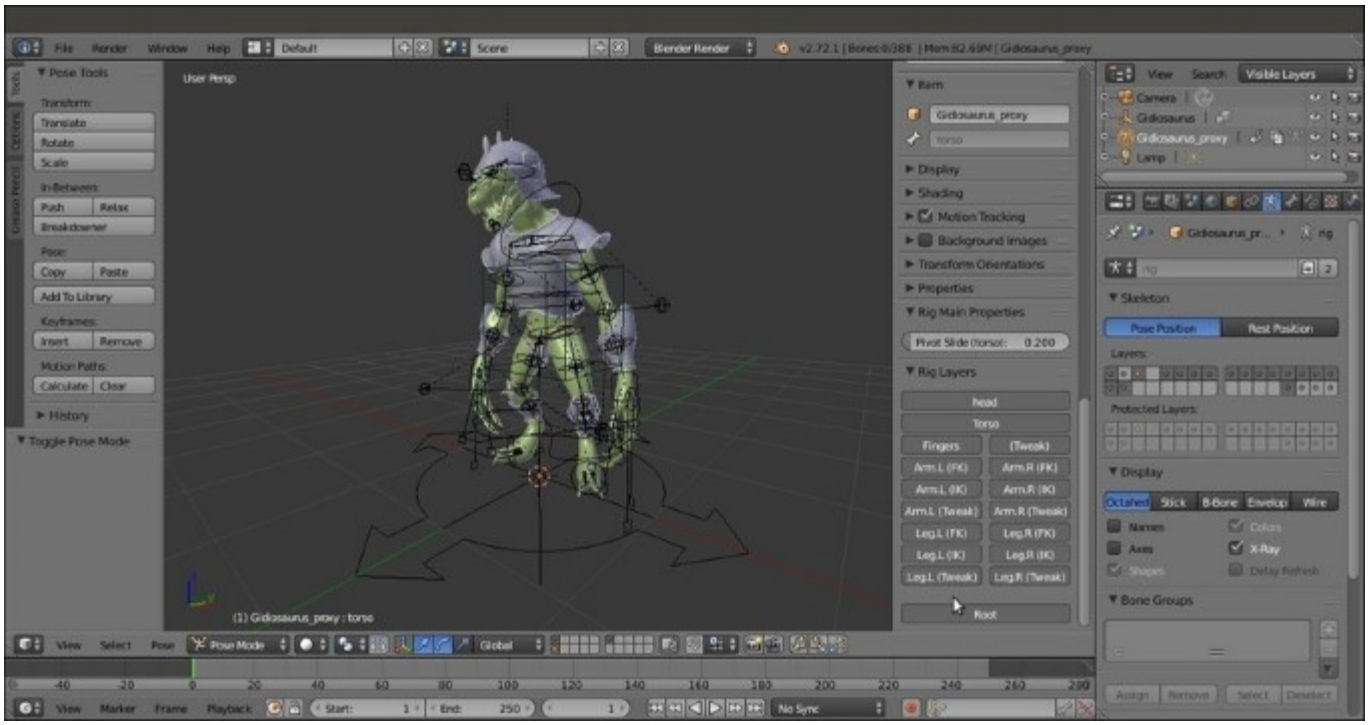
The **rig** appears as a separate object in the **Outliner**, identified by the name **Gidiosaurus_proxy**; at this point, it is possible to only select the **rig** (which is still in **Object Mode**) and move it to a different layer.

4. Select the **Gidiosaurus_proxy** object and move it to the **11th** scene layer (use the *M* key). *Shift*-click to enable the layer and then go to the **Display** subpanel, under the **Object Data** window, to enable the **X-Ray** item.
5. Press *Ctrl* + *Tab* to go into **Pose Mode** and the *N* key to call the viewport **Properties** sidepanel.
6. Save the file as `Gidiosaurus_proxy.blend`.

At this point, looking at the viewport **Properties** sidepanel, we will see the **Rig Layers** interface usually created by the **Rigify** addon, *but* if we save the file and reopen it, the interface is gone.

This is because, at least for the moment, the Python script that draws the rig interface doesn't get automatically linked with the rig, so it's something we must do by hand. This is not a big issue, and by the way, the procedure is incredibly simple:

7. Click again on **File** | **Link** in the main header menu (or press *Ctrl* + *Alt* + *O*).
8. Browse to the `Gidiosaurus_library.blend` file, click on it, and then click on the **Text** item. Click on the **rig_ui.py** item (the Python script for the interface) and then on the **Link from Library** button.
9. Save the file and reopen it; the rig interface is visible again on the viewport **Properties** sidebar:



The rig interface at the bottom of the Properties sidepanel

See also

- http://wiki.blender.org/index.php/Template:Release_Notes/2.43/Animation/Proxy_Objects
- http://www.blender.org/manual/data_system/linked_libraries.html?highlight=proxy#proxy-objects

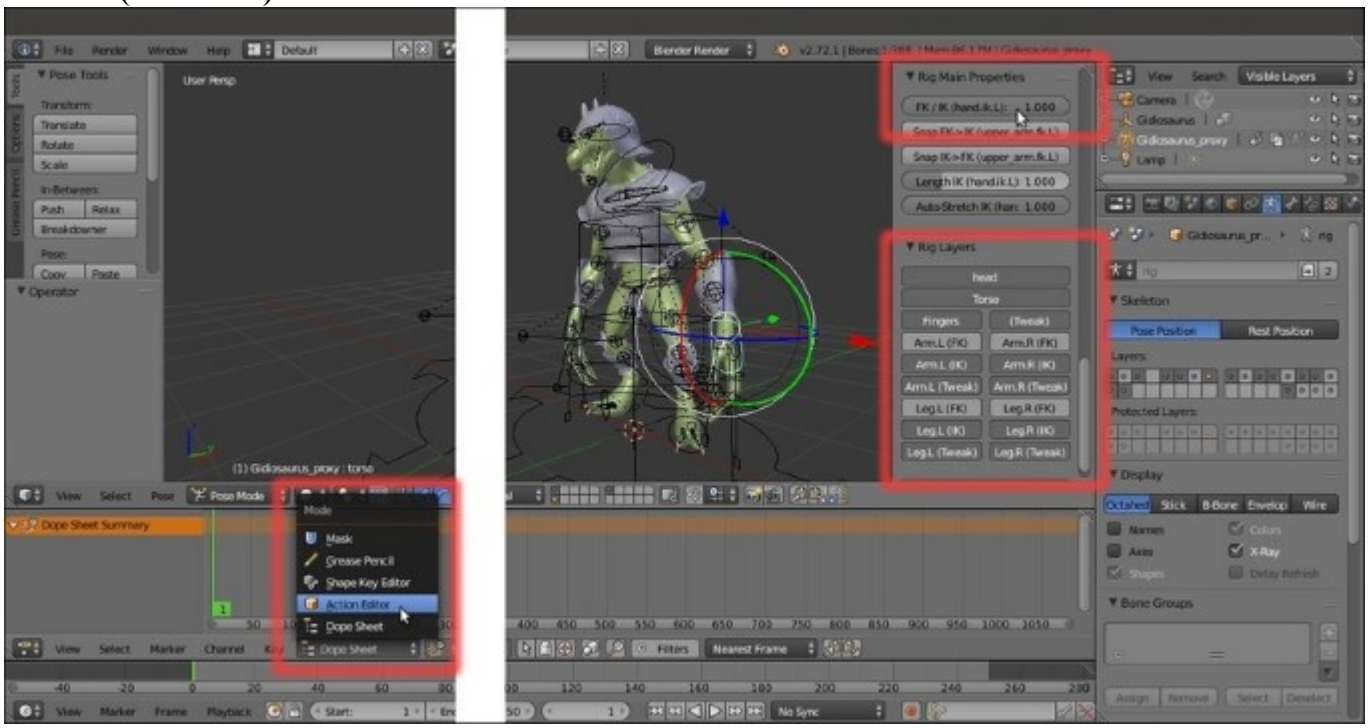
Creating a simple walk cycle for the character by assigning keys to the bones

We are now going to create a simple **walk cycle** for the **Gidiosaurus** character by assigning **position** and **rotation** (and in some cases, also **scaling**) keys to the **control bones** of the **rig**.

Getting ready

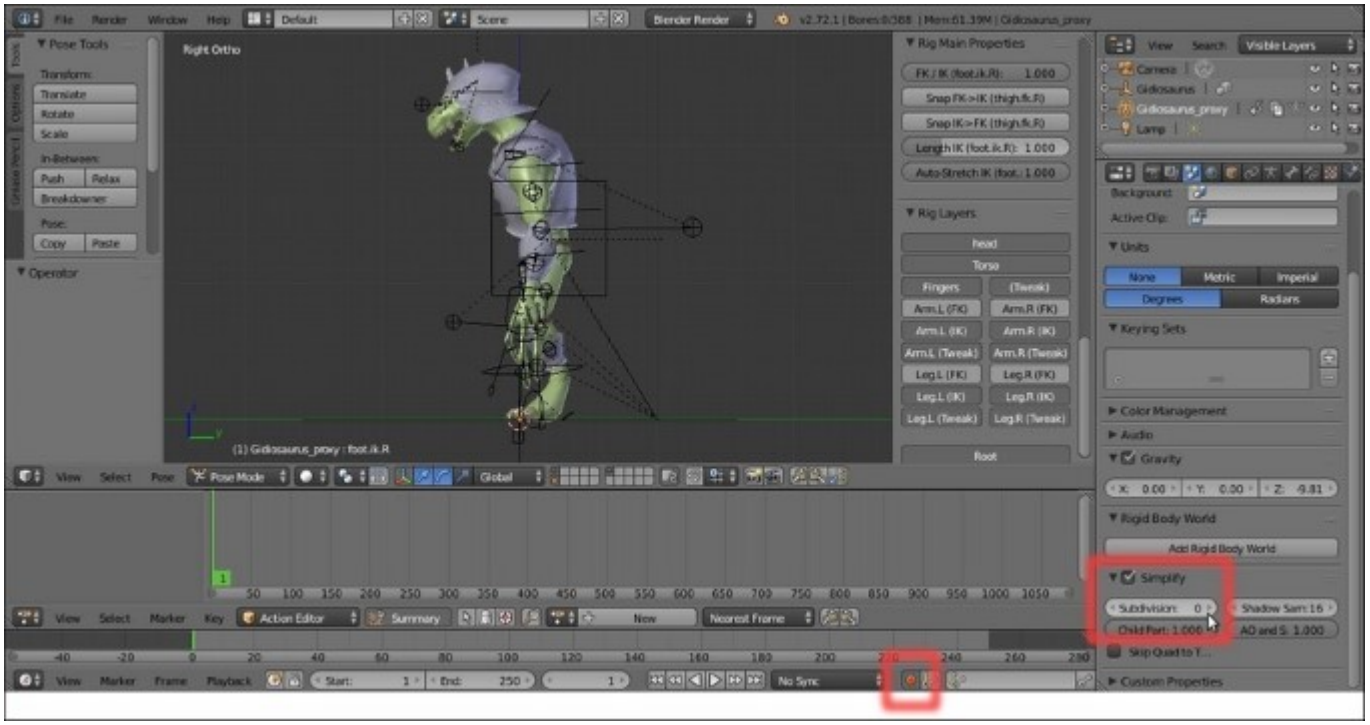
In Blender, there is already a **preset screen** layout named **Animation** that you can switch to and start animating. By the way, I usually prefer to set up my screen layout for the required task, and animating is no exception, so let's first prepare the scene and the screen for the job:

1. Open the `Gidiosaurus_proxy.blend` file.
2. If necessary, enable the **3D manipulator widget** in the toolbar of the 3D view (press *Ctrl* + Spacebar), click on the **Translate** icon button, and set **Transform Orientation** to (just for the moment) **Global**.
3. Split the 3D view horizontally into two windows and change the bottom one into a **Dope Sheet** window. Click on the *Editing context being displayed* button on its toolbar to switch from **Dope Sheet** to the **Action Editor** context **Mode**.
4. Go to the **Properties** sidepanel of the 3D viewport (use the *N* key to make it appear if necessary) and under the **Rig Layers** subpanel, disable the **Arm.L (FK)**, **Arm.R (FK)**, **Leg.L (FK)**, and **Leg.R (FK)** buttons.
5. Select the **Gidiosaurus_proxy** rig, making sure you're in **Pose Mode**, and select the **hand.ik.L** control bone. Go to the **Rig Main Properties** subpanel under the **Properties** panel and set the **FK / IK (hand.ik.L)** slider to **1.000**:



Switching from the Graph Editor to the Action Editor and setting the Inverse Kinematics in the Rig Layers subpanel

6. Repeat for the **hand.ik.R** bone and for the **foot.ik.L** and **foot.ik.R** control bones as well.
7. Go to the **Scene** window, enable the **Simplify** subpanel, and set the **Subdivision** level to **0** (or, if you have a more powerful machine than my laptop, also to **1**).
8. Go into the **Side** view and press the *5* key on the numpad to go into the **Ortho** view.
9. Click on the **red button icon** (*Automatic keyframe insertion for Objects and Bones*) in the **Timeline** toolbar.



The red button icon and the Subdivision Surface modifier subpanel

10. Save the file as `Gidiosaurus_walkcycle.blend`.

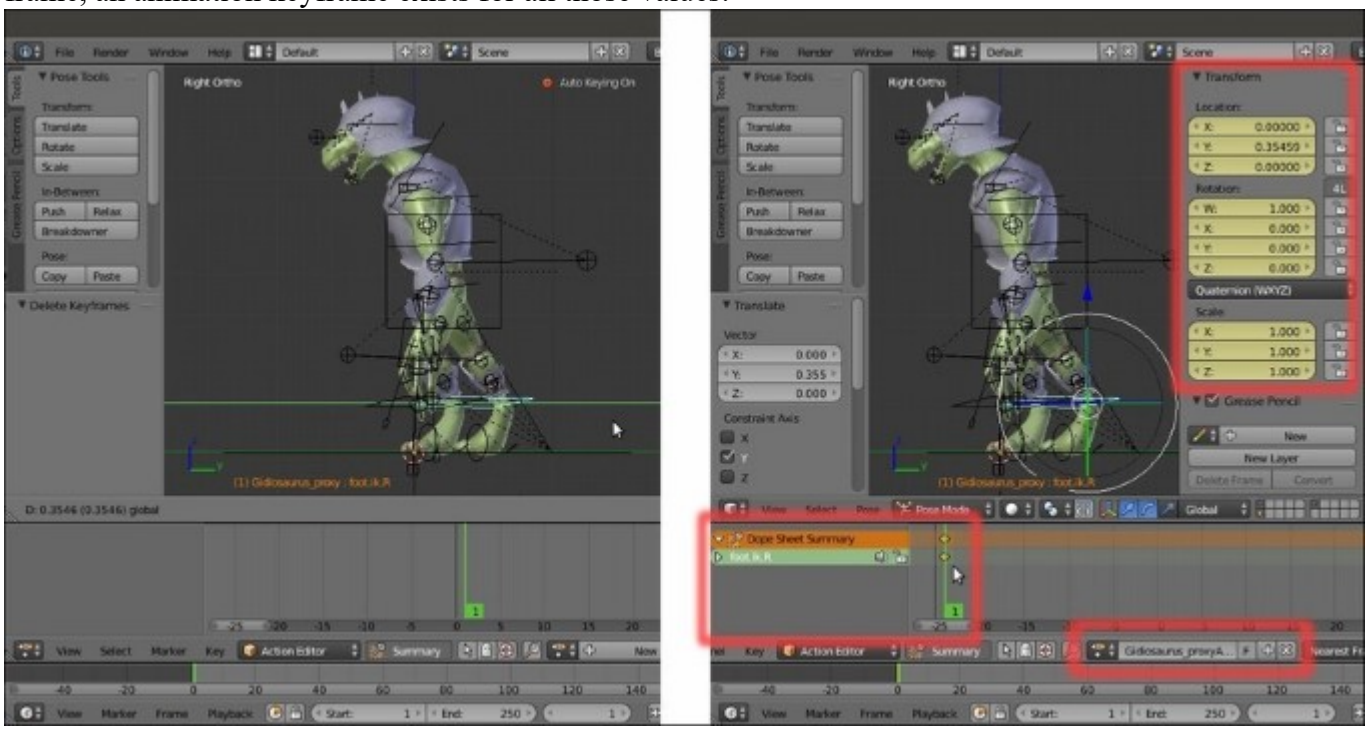
How to do it…

To create a walk cycle, it's important to first establish the start and the end poses of the walk, so let's pose our character for his first step:

1. Be sure to be in the first frame (which in Blender is frame **1** and not **0**), both by clicking on the *Jump to first/last frame in frame range* left button on the **Timeline** toolbar or by pressing the *Shift* + Left Arrow keys.
2. Select the **foot_ik.R** control bone and, by using the widget, move it backward on the global *y* axis to around **0.350**.

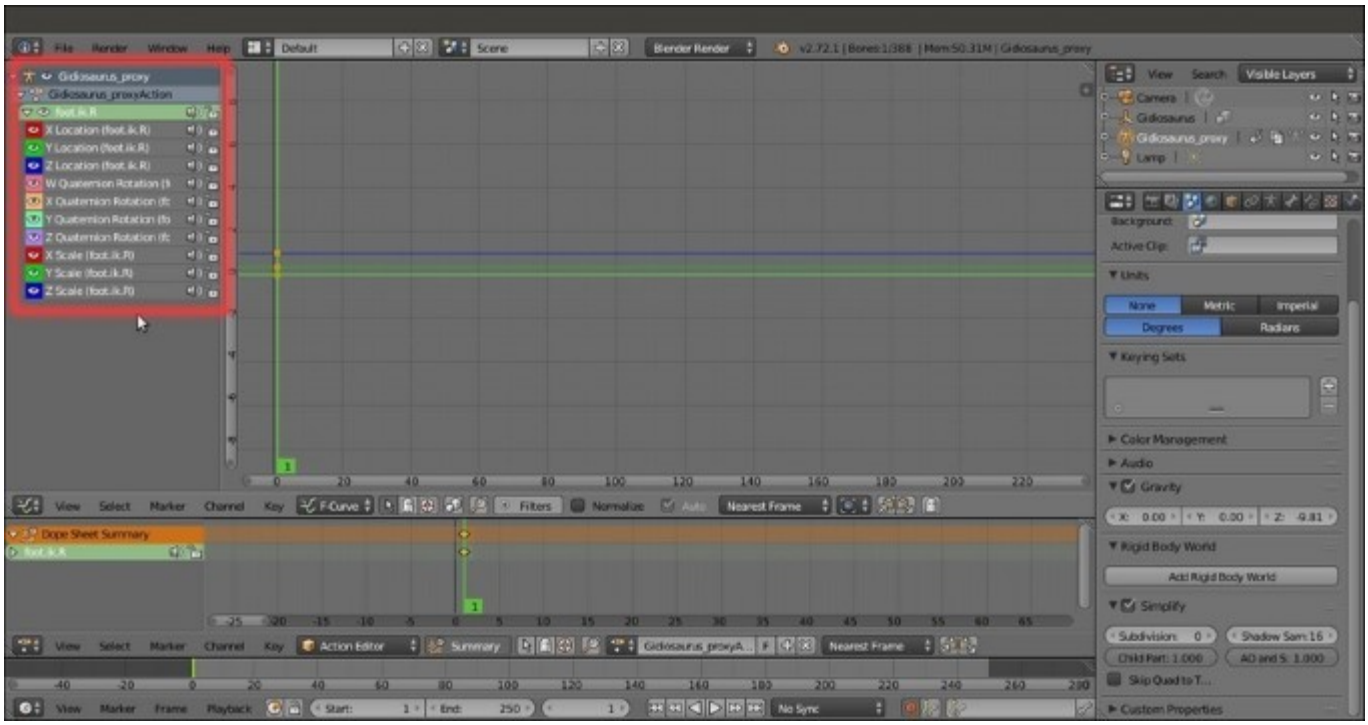
As you release the mouse button, an **Action** datablock, automatically named **Gidiosaurus_proxyAction**, is created and a keyframe for the **foot_ik.R** bone is automatically added in the first frame in the **Action Editor** window. We can also see the value for the movement on the *y* axis in the **Transform** subpanel.

Note that all the transformation value slots turned yellow; this is to show that at the current frame, an animation keyframe exists for all those values:



Setting the first key at frame 1

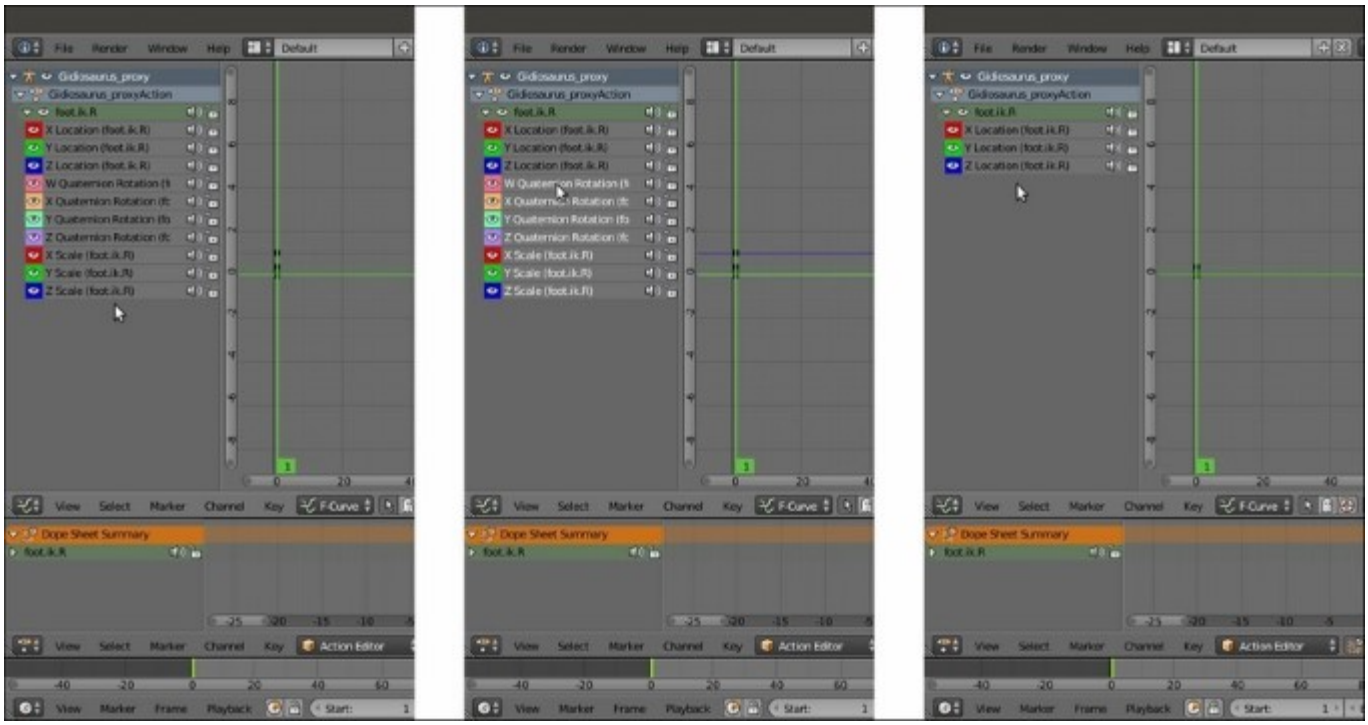
3. Temporarily, switch **3D View** to the **Graph Editor** window:



The Graph Editor window

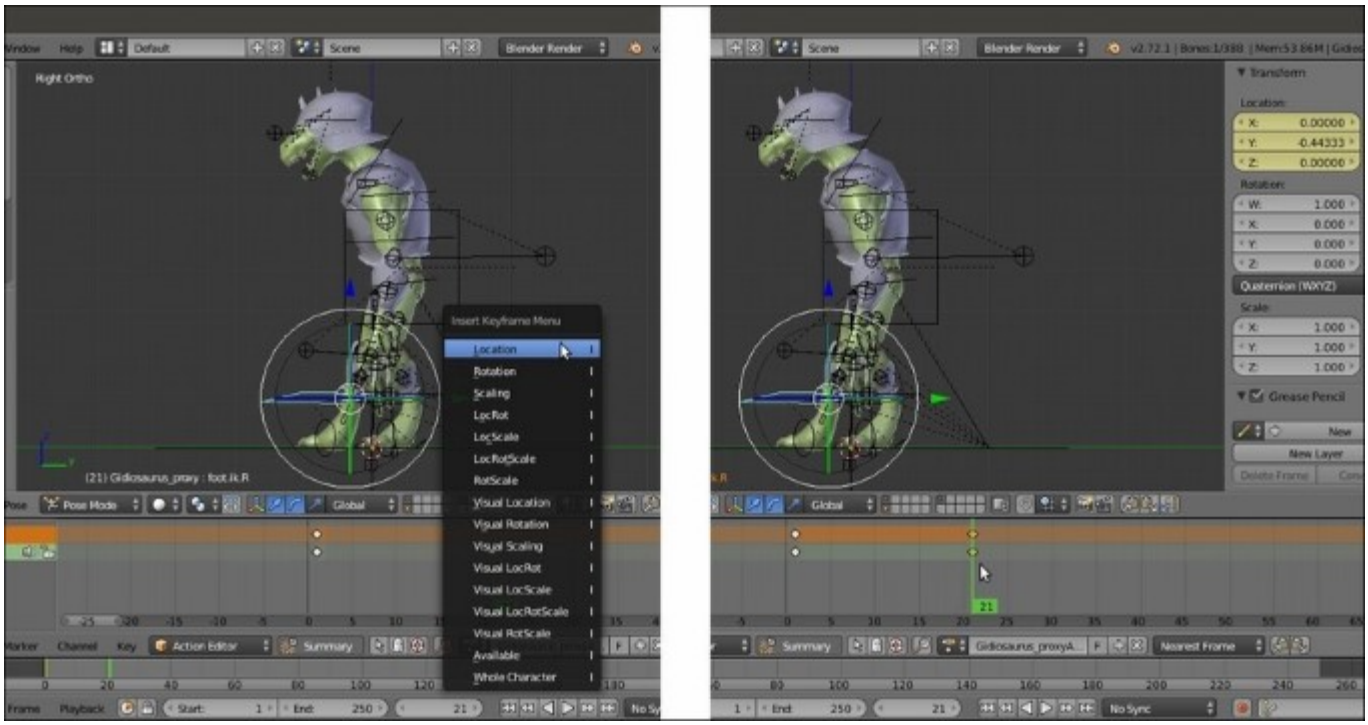
As you can see, because we enabled the **red button icon** (*Automatic keyframe insertion for Objects and Bones*) in the **Timeline** toolbar, every time we move, rotate, or scale a bone, a keyframe for **Location**, **Rotation**, and **Scaling** is automatically added to the **Action**. This can be handy, but also results in a lot of useless keyframes, for example, for most of the rig bones, we need to set keys for the **Location** and/or the **Rotation**, but very rarely for the **Scaling**.

4. Put the mouse cursor inside the **Curve Editor** area of the **Graph Editor** and press the *A* key to deselect everything.
5. *Shift* + left-click on the **Scale** and **Quaternion Rotation** items in the **Gidiosaurus_proxy** Channel Region to select them, then press *X* to delete them:



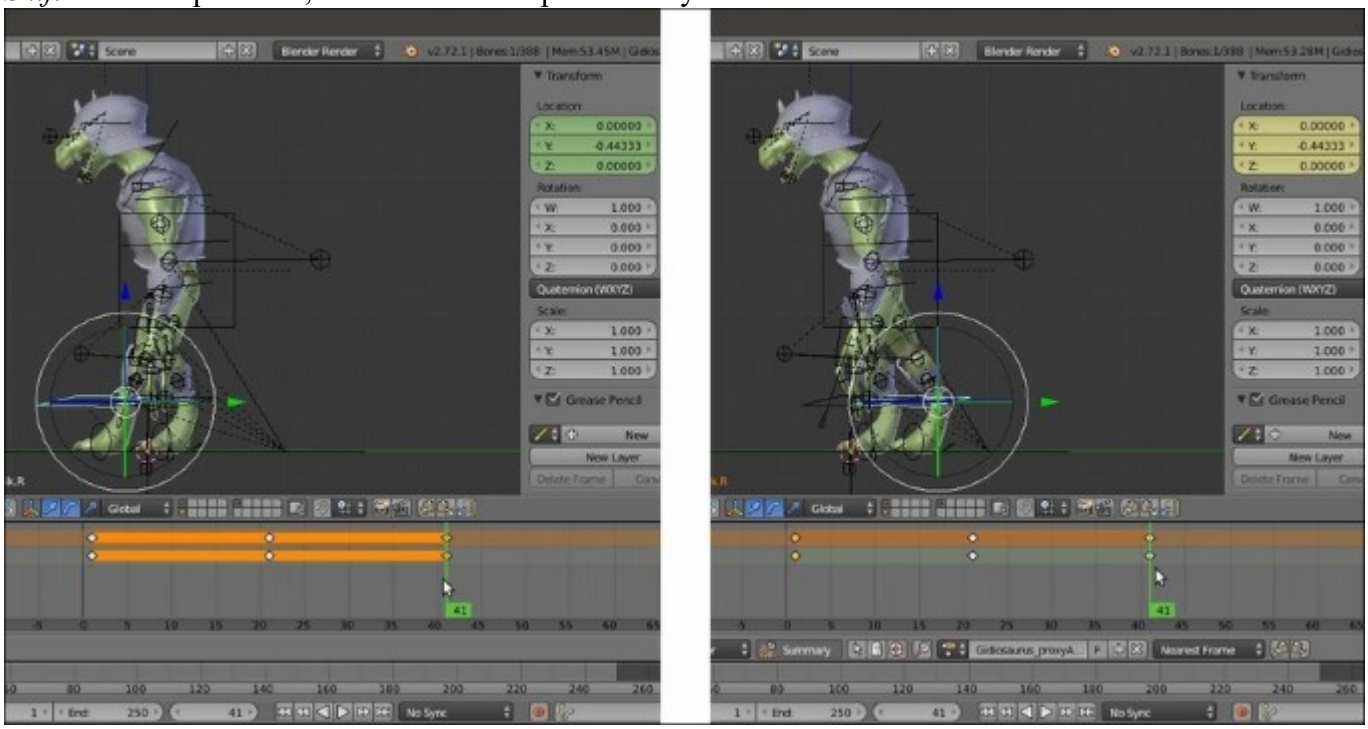
Deleting the useless transformation channels

6. Switch back to the **3D View**, and in the **Transform** subpanel in the **Properties** sidebar (and in the **Transform** subpanel under the **Bone** window in the main **Properties** panel), now only the **Location** slots are highlighted in yellow.
7. Disable the **red button icon** (*Automatic keyframe insertion for Objects and Bones*) in the **Timeline** toolbar.
8. Go to frame **21** by grabbing and moving the **Time Cursor** inside the **Timeline** window or the **Action Editor** window, or by typing the frame number inside the *Current Frame* button on the **Timeline** toolbar.
9. Select the **foot_ik.R** control bone and by using the widget, move it forward on the global *y* axis for around **-0.440**.
10. Press *I* and in the **Insert Keyframe Menu**, select the **Location** item; this adds a second key to the **foot_ik.R** bone at frame **21**, but this time only for **Location**:



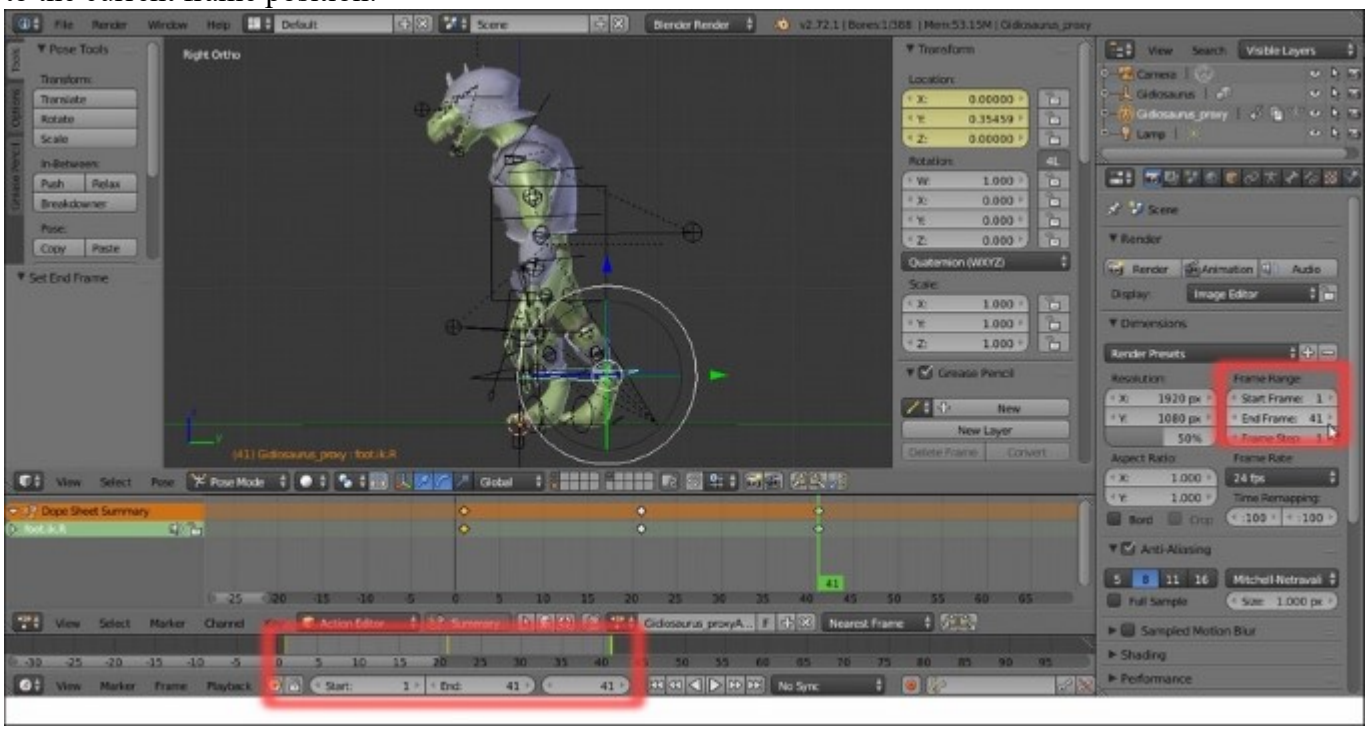
Setting a Location only key through the Insert Keyframe Menu pop-up

11. Go to frame **41**, right-click to select the key at frame **1** in the **Action Editor** window, and press *Shift* + D to duplicate it, then move the duplicated key to frame **41**.



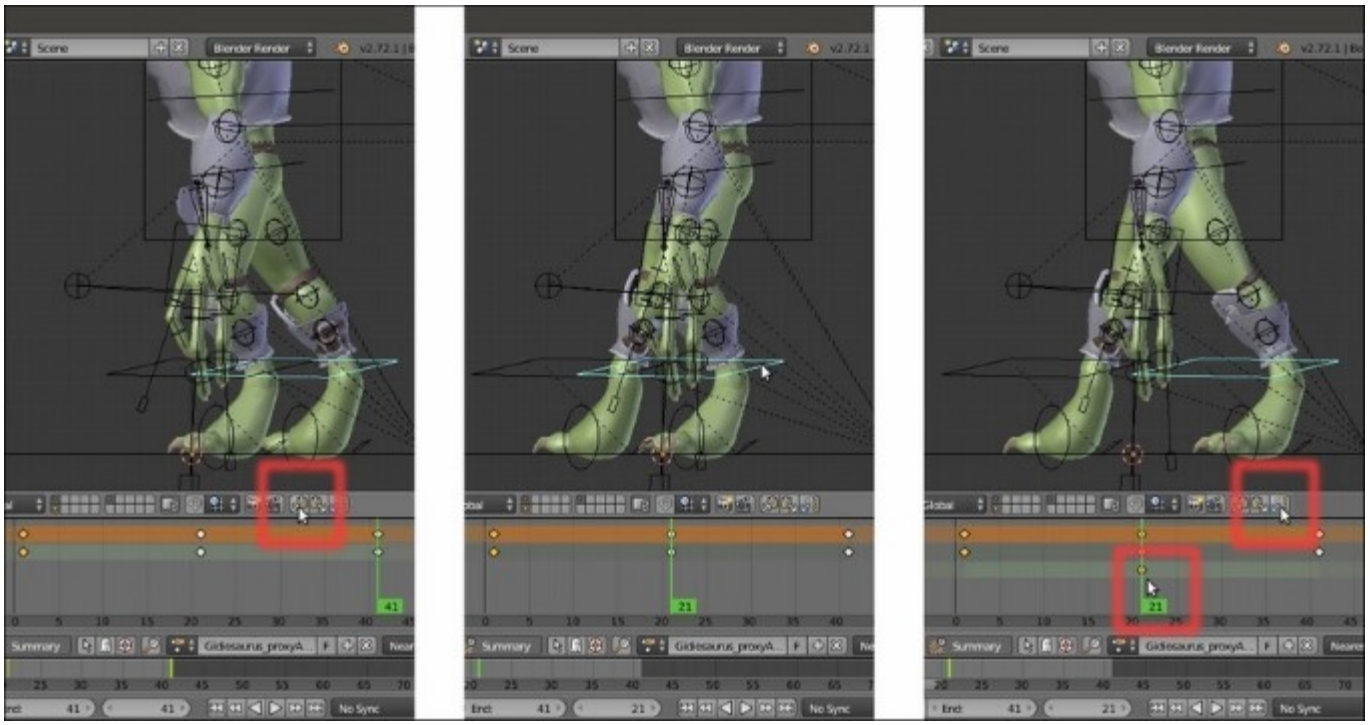
Creating poses at different frames by duplicating keys

12. Put the mouse cursor in the **Timeline** and press the *E* key to set the total length of the animation to the current frame position:



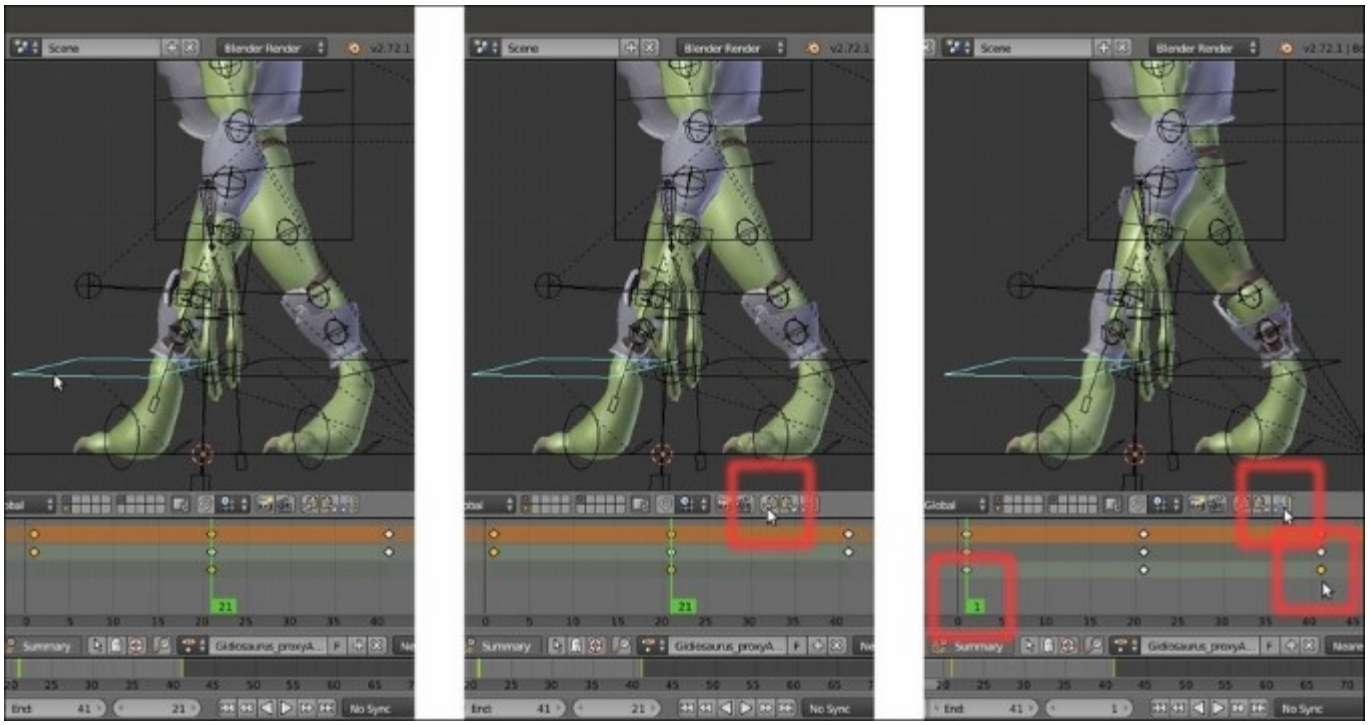
Setting the action total length in frames

13. Still at frame **41** (but this being a cycle, frame **1** could also be fine) and with the **foot_ik.R** bone selected, click on the *Copy the current pose of the selected bone to copy/paste buffer* button on the 3D viewport toolbar.
14. Go to frame **21** and select the **foot_ik.L** bone, then click on the *Paste the stored pose on to the current pose* button at the extreme right side of the 3D viewport toolbar to paste a **mirrored pose**.
15. Press the *I* key and in the pop-up menu, click on the **Location** item to add a new key:



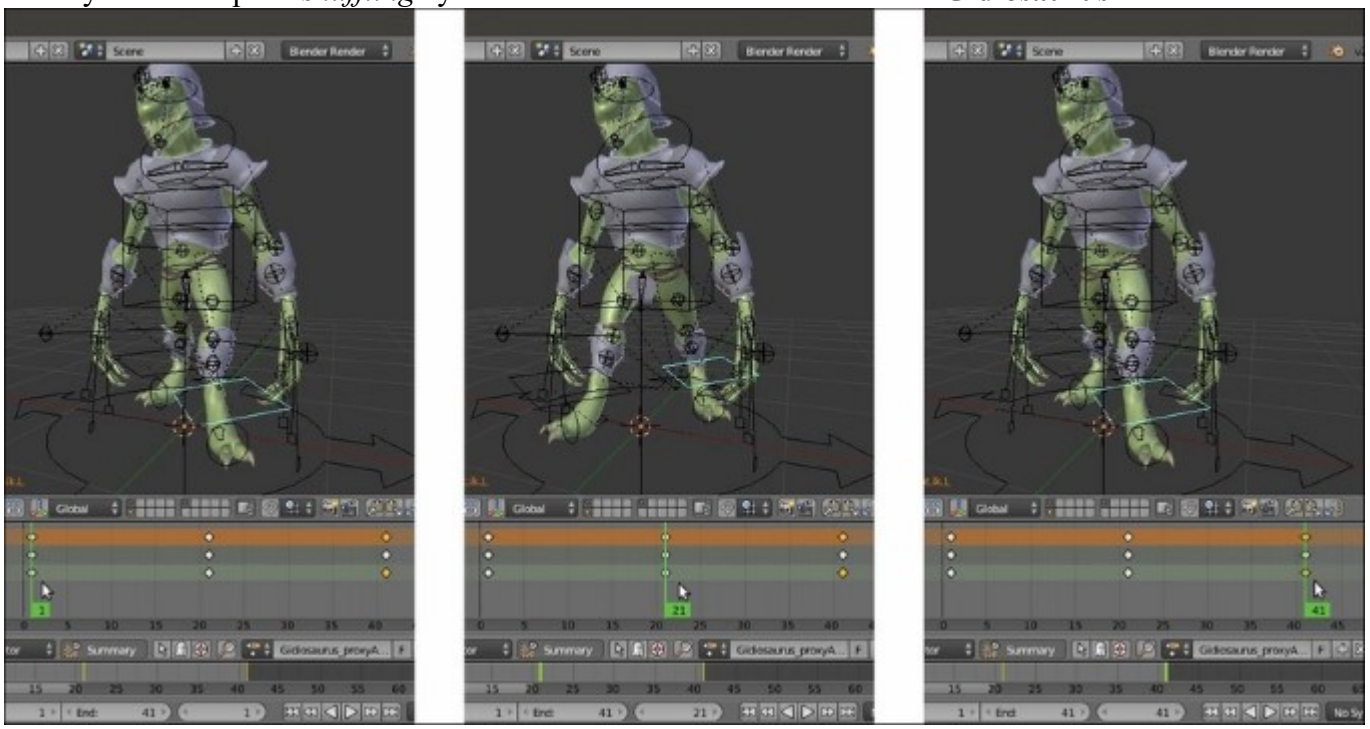
Copying a pose and pasting it reversed

16. Now, still at frame **21**, select the **foot_ik.R** bone and click on the *Copy the current pose of the selected bone to copy/paste buffer* button on the 3D viewport toolbar.
17. Go to frame **1**, select the **foot_ik.L** bone, and again click on the *Paste the stored pose on to the current pose* button to paste the reversed pose, then press *I* and insert a **Location** key.
18. Select and duplicate the new key at frame **1** for the **foot_ik.L** bone and move the duplicated one to frame **41**:



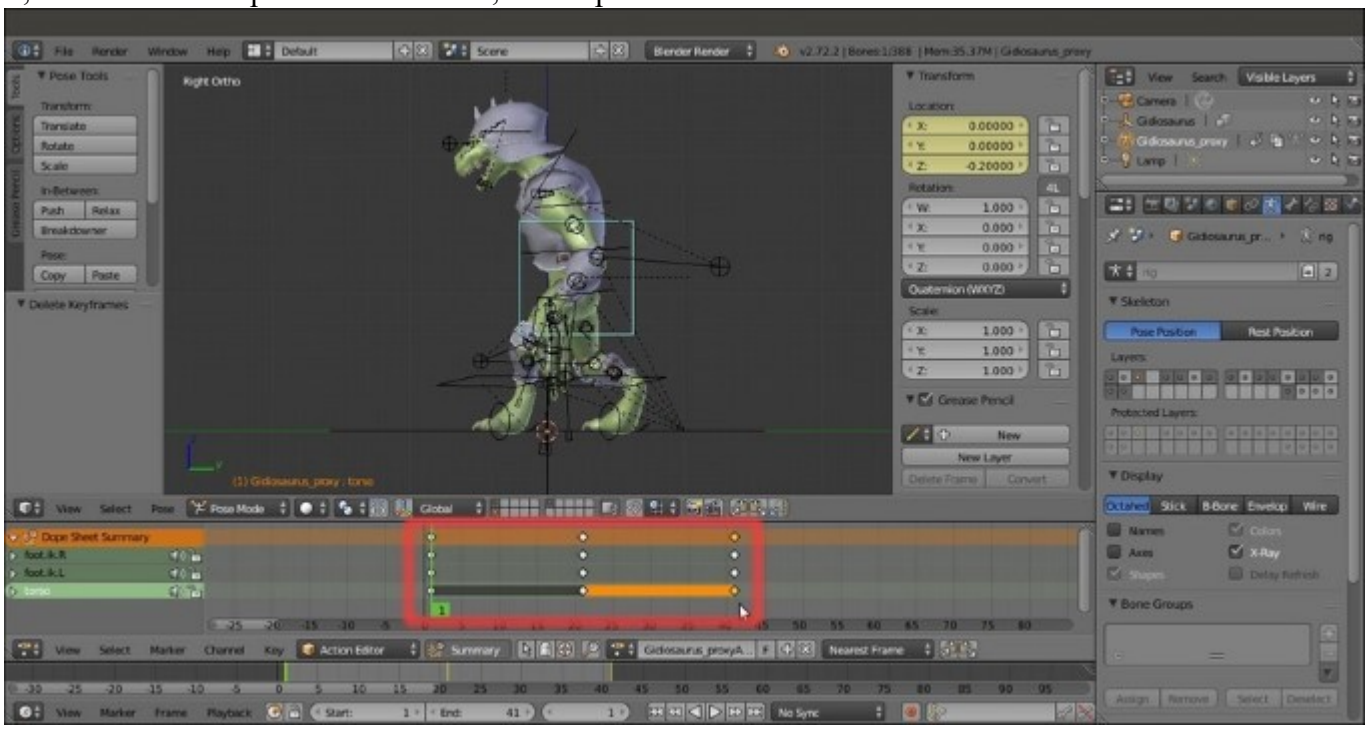
Creating new keyframes by copying, pasting, and duplicating pose keys

At this point, by scrolling the **Time Cursor** in the **Timeline**, in the **Action Editor** window, or by clicking on the *Play Animation* button in the **Player Control** on the **Timeline** toolbar, we can already see a complete *shuffling* cycle of the movement of the feet of the **Gidiosaurus**:



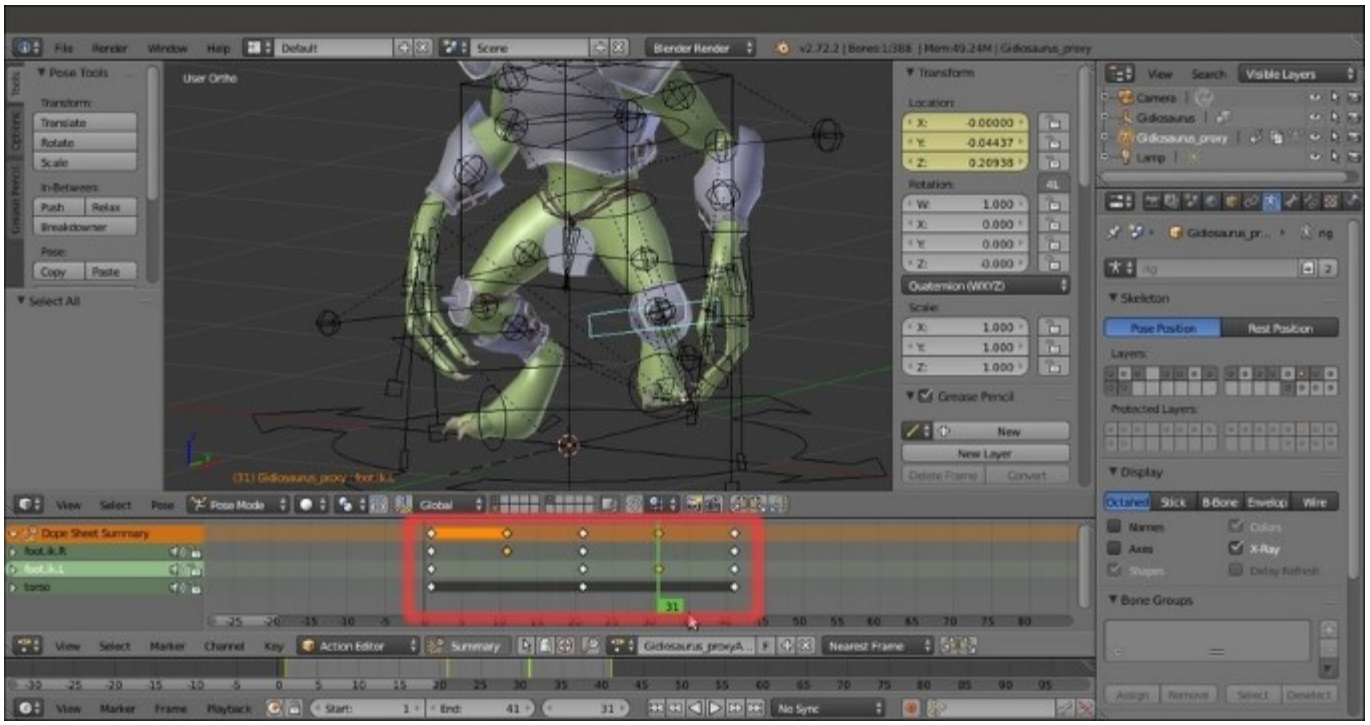
The Gidiosaurus' walk cycle with sliding feet

19. Now go to frame **1**, select the **torso** bone, and lower it on the *z* axis for almost **-0.200**, then assign a position key.
20. Select the just added **torso** bone key in the **Action Editor** window, press *Shift* + *D* to duplicate it, and move the duplicate to frame **21**, then repeat for frame **41**:



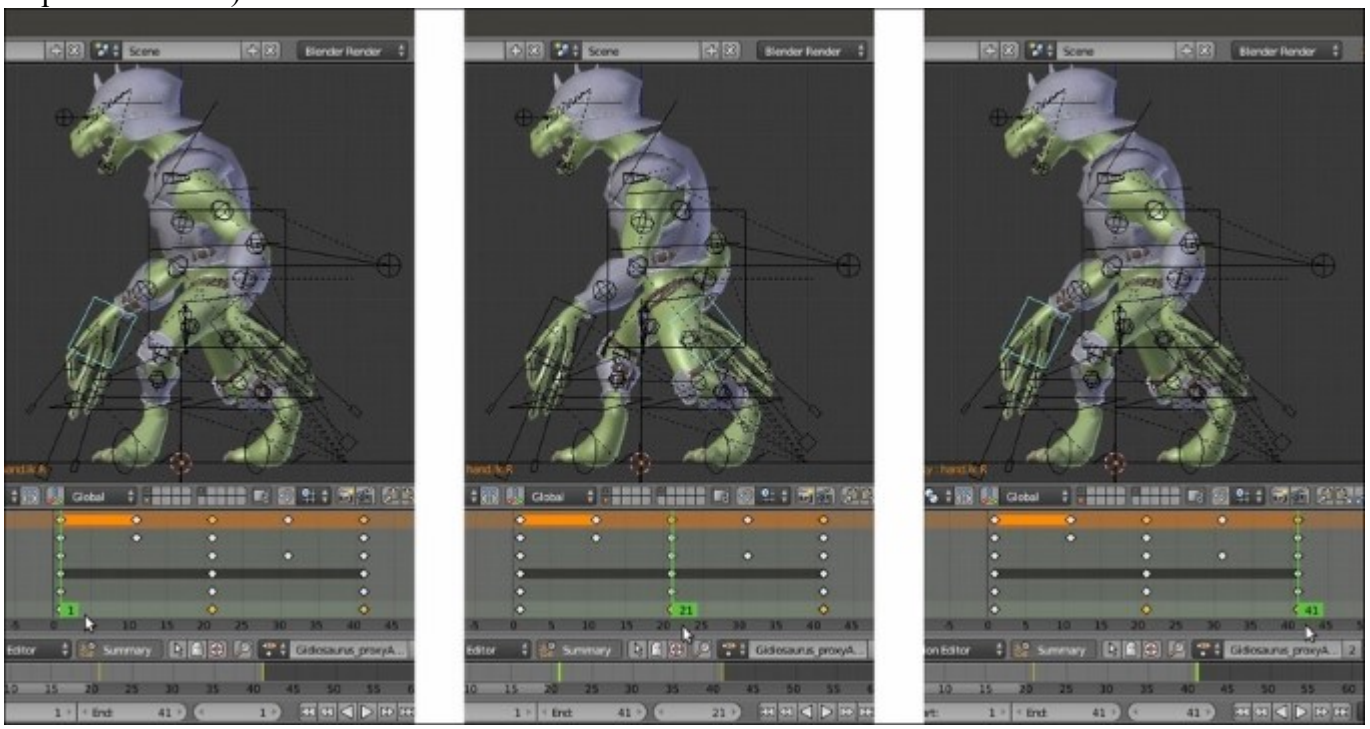
Animating the torso

21. Go to frame **11**, select the **foot_ik.R** bone, and move it on the *z* axis for **0.200**, then assign a position key.
22. As we already did at steps 16 and 17, copy the bone pose, go to frame **31**, and paste it reversed, then assign a position key to the **foot_ik.L** bone.



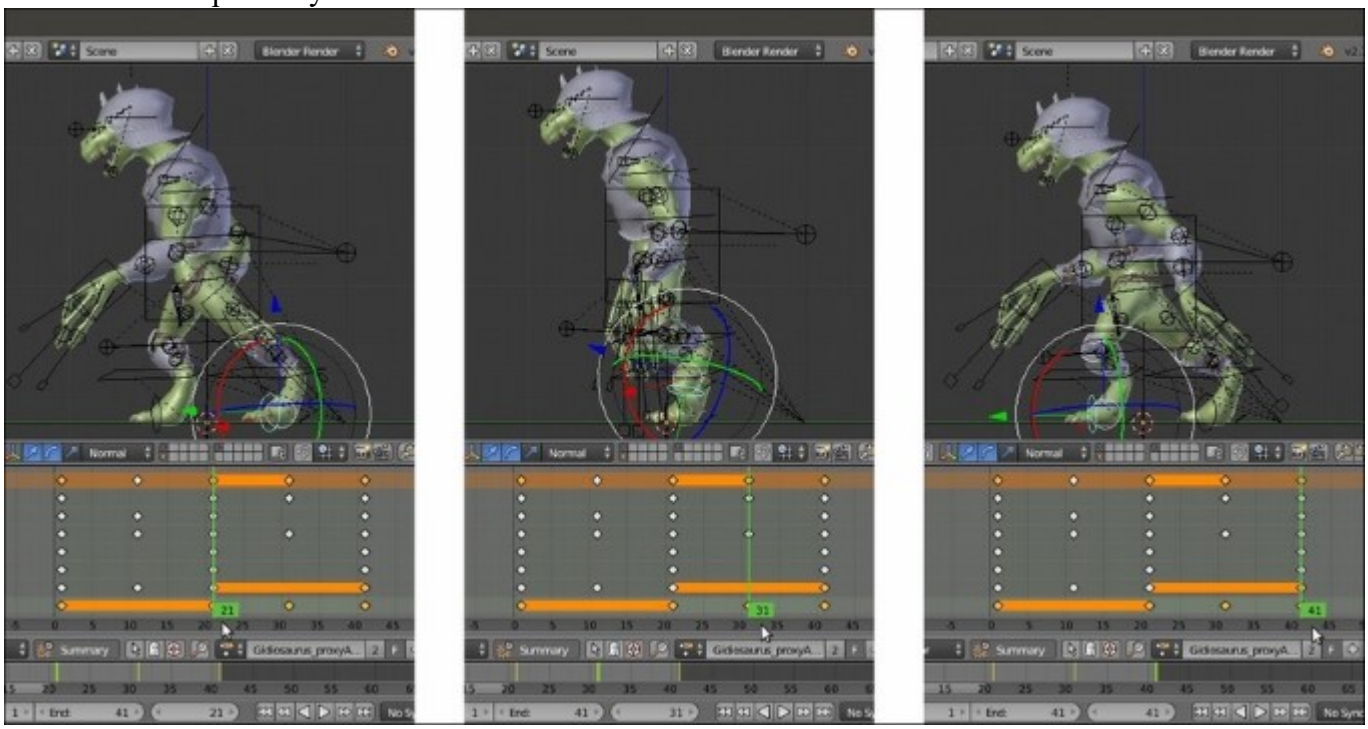
Assigning more translation keys

23. Working in the same manner, select the **hand_ik.R** and **.L** bones and animate them according to the **Gidiosaurus'** walk (note: as for any average walk cycle, in the opposite position with respect to the feet):



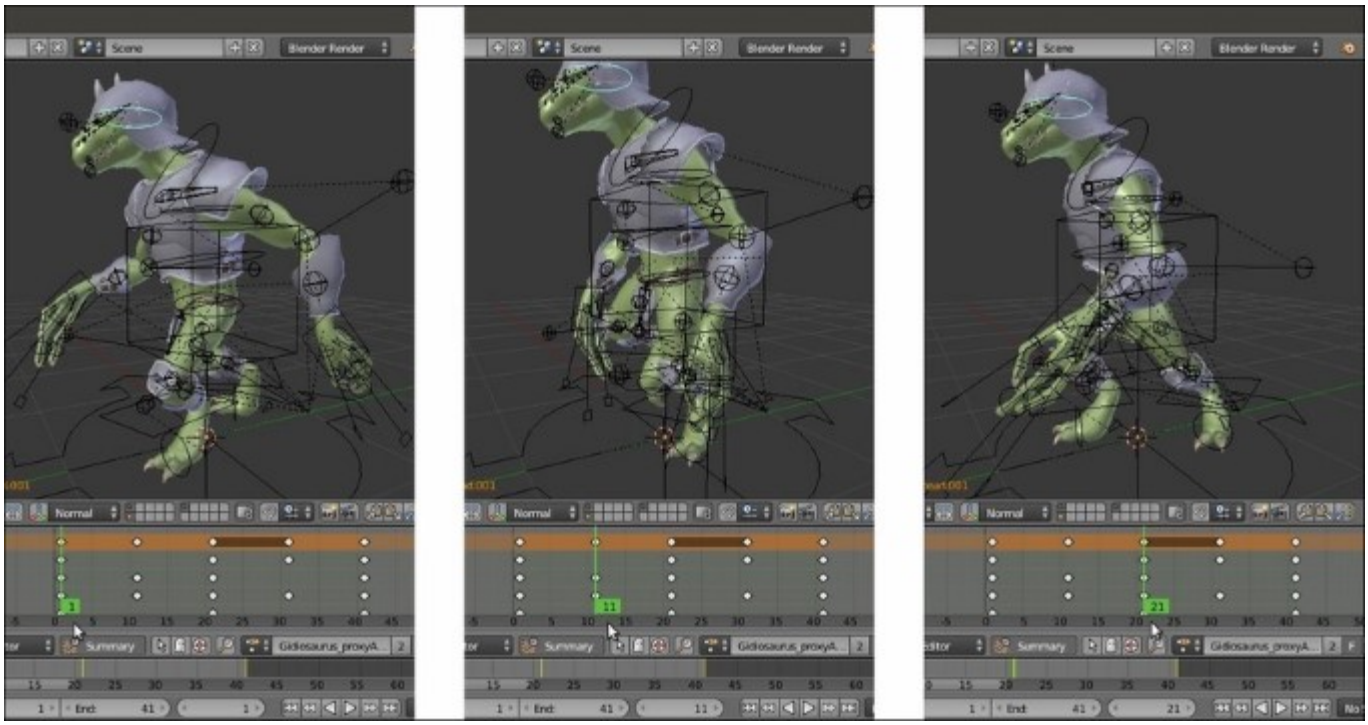
Animating the arms to complete the walk cycle

24. Reselect the **torso** bone, go to frame **1**, and move it forward for **0.240** on the *y* axis. Assign a new position key (to overwrite the old one), then delete the keys at frames **21** and **41** and substitute them with duplicates of the new frame **1** key.
25. Go to frame **11** and move the **torso** bone for almost **0.200** upward on the *z* axis. Duplicate the key for frame **31**.
26. Go to frame **1** and select the **toe.R** bone, then assign a rotation key. Go to frame **11** and rotate the bone on the normal *x* axis (the red circle in the widget tool with **Transform Orientation** set to **Normal**) for **75°**. Go to frame **21** and press *Alt* + *R* to clear the rotation pose and assign a rotation key. Use *Shift* + *D* to duplicate the last added key and move the duplicated one to frame **41**.
27. Select the **toe.L** bone and assign a rotation key at frame **1**, then go to frame **21** and repeat. Copy the **toe.R** pose at frame **11** and paste it reversed for the **toe.L** bone at frame **31**, then assign a cleared rotation pose key at frame **41**:



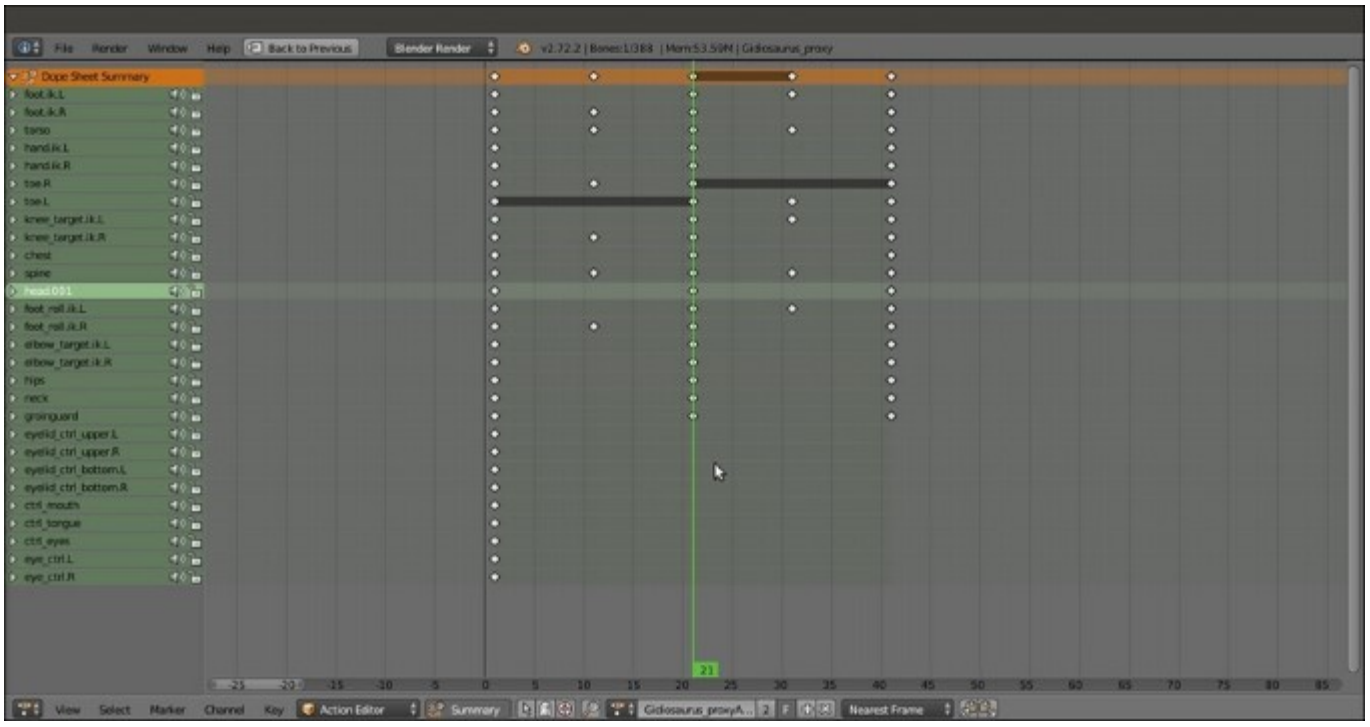
Adding the in-between poses for the feet

28. Following the previous procedures, set keys for the position and/or the rotation of all the affected bones, also adding movements such as the rotation of the **torso** and of the **hips**, the position of the **pole target** for **legs** and **arms**, the swinging of the **head** to compensate for the body's lateral movements, the closed **mouth** and the open **eyelids**, and so on:



The first phase of the walk cycle animation is almost done

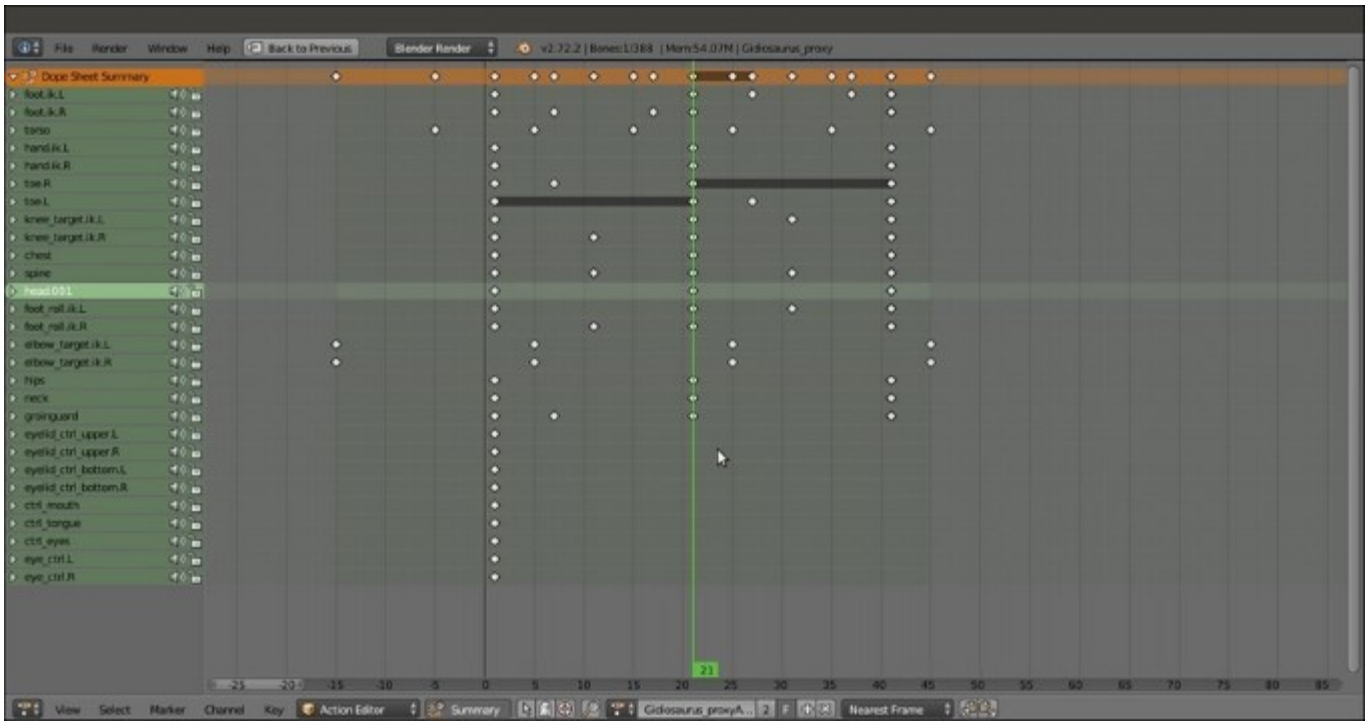
The animation cycle, at this point, looks really stiff and *robotic*. This is simply because everything *happens at the same time*, that is, in the same frame, as you can easily see in the **Action Editor** window (to enlarge a window, put the mouse cursor inside it and press *Ctrl* + Up Arrow; to go back, press *Ctrl* + Down Arrow):



The maximized Action Editor window with the walk cycle action

To make the animation look more realistic and natural, we must offset some of the keys to make the different actions *happen at different times;* for example, the **torso** bone goes down a few frames later than the **foot** touching the ground, and goes up a few frames later as well, the same for the **head** swinging, and so on.

29. To offset the affected keys, simply select and/or *Shift*-select and move them for the required frames, forward or backward in the **Action Editor** window. Here, a bit of testing is needed to reach the right number of frames (usually in the range of **3-5** frames, by the way).
30. Where a *hole* happens at frame **1** in the action channel for a bone because of the dislocation of the keys, simply duplicate the last right side key of that bone and move it to the appropriate **negative frame** position. That is, to the left side of frame **0**, and be sure that the relative item, **Allow Negative Frames**, is enabled in the **Editing** tab of the **User Preferences** panel, as you can see in the following screenshot for the **torso** and for the **elbow_target_ik** bones:



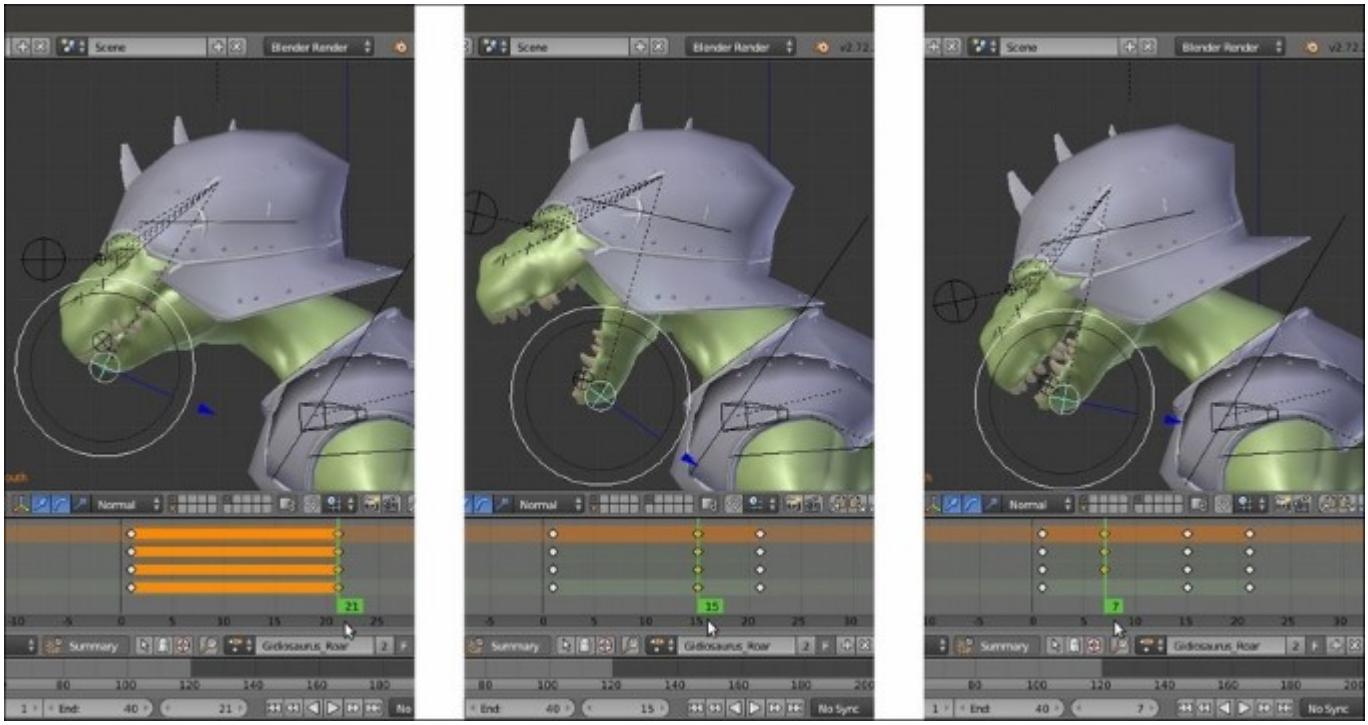
Duplicated keys moved to the Negative Frames space

31. Rename the action `Gidiosaurus_Walkcycle`. To better check the playing animation, go to the **Timeline** toolbar to set the end frame for the total length of the animation to **40** frames, because frame **1** and frame **41** are the same poses.
32. Save the file.

    At this point, we have made our first action with the **Gidiosaurus** character, and it's a **41** frame-long walk cycle meant to be repeated in loops for longer animations.

    Because in the next recipe we are going to use the **Non Linear Action Editor** (**NLA Editor**) to re-use the **action datablocks** to build the final animation, we need now to create some more actions to be mixed with the walk cycle one.
33. Activate the **Fake User** for the **Gidiosaurus_Walkcycle** action by clicking on the **F** icon button to the side of the action datablock on the **Action Editor** toolbar, then click on the **X** icon button to unlink the action datablock.
34. Put the mouse cursor in the 3D viewport and press the *A* key to select all the control bones, then press *Alt* + *G*, *Alt* + *R*, and *Alt* + *S* to clear any position, rotation, or scale and restore the rig default pose (actually, the only control bones using the **scale** operator for the animation are the **fingers**, which we haven't animated so far).
35. Be sure to be at frame **1** and zoom to the character's **head**, select the **head.001** and **neck** bones, and assign a rotation key, then select the **ctrl_mouth** bone and assign a position key.
36. Rename the action `Gidiosaurus_Roar`, then enable the **Fake User**; use *Shift* + *D* to duplicate the keys and move the duplicated ones to frame **21**.

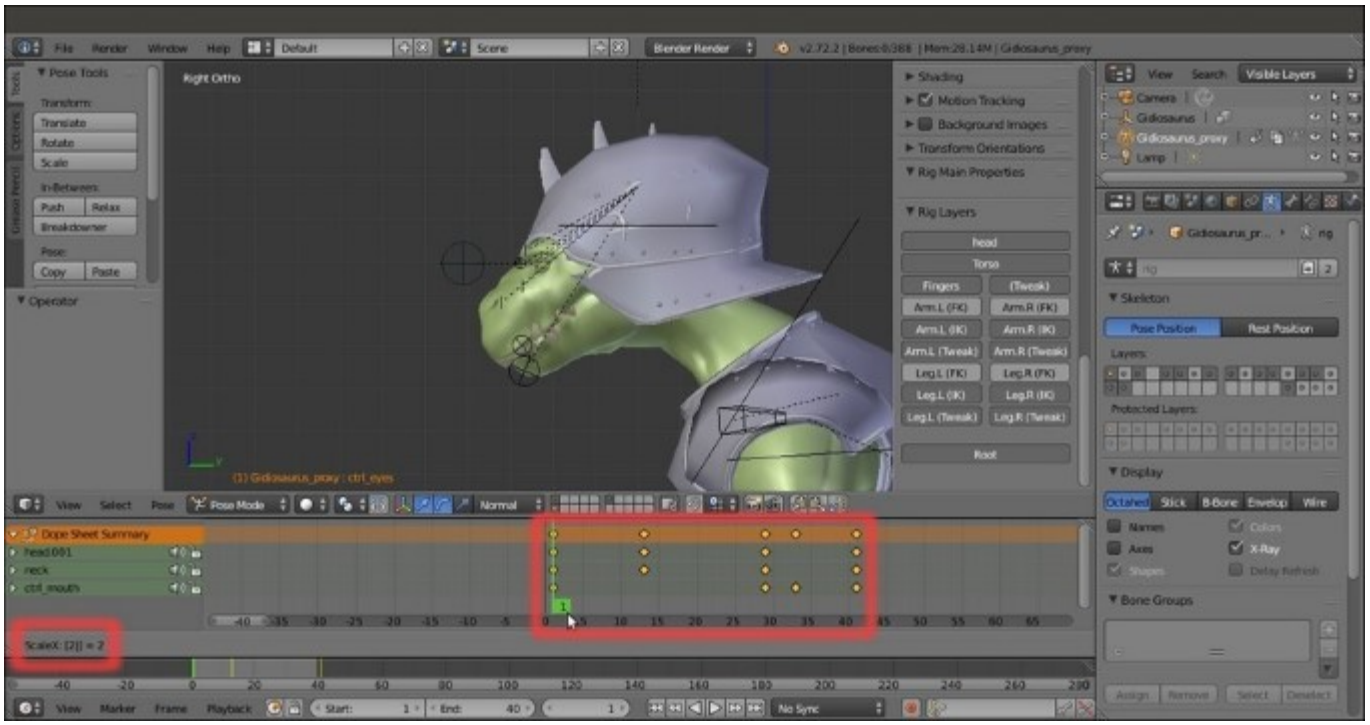
37. Go to frame **15** and rotate the **head.001** and **neck** bones clockwise to raise the **head**, then open the **mouth** wide by moving the **ctrl_mouth** bone down.
38. Go to frame **7** and rotate the **head.001** and **neck** bones counterclockwise a bit to lower the **head**:



The Gidiosaurus_Roar action

We have now built a roar action for the **Gidiosaurus**, but it happens in only **21** frames, so it's really too fast. Although it is possible to scale any action strip in the **NLA Editor** window, in this case it's better to do it directly in the basic action itself.

39. In the **Action Editor** window, put the **Time Cursor** to frame **1**, then press the *A* key to select all the keys of the action. Press *S* | *X* | **2** | *Enter* to scale the action of the double to frame **41**:



Scaling the action on the position of the Time Cursor

40. Now that the action length has been doubled, we can move some keys of a few frames and also animate the movement of the character's **tongue** a bit during the *roar*:

Animating the tongue

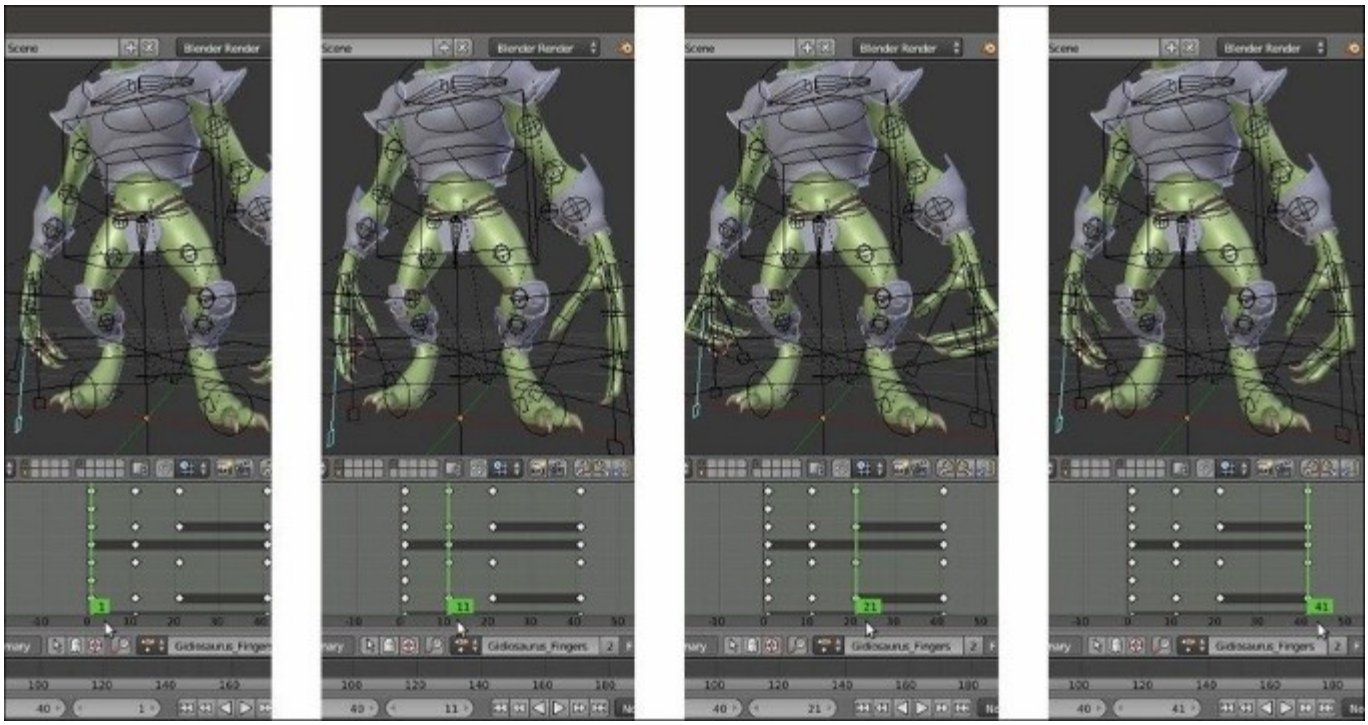
41. Again, click on the **X** icon button to unlink the action datablock, select all the bones, then press *Alt* + *G* and *Alt* + *R* to clear the poses.
42. *Shift*-select the **thumb.R** and **.L**, **f_index.L** and **.R**, and **f_middle.L**, and **.R** control bones and add a **Scaling** key. Rename the newly created action `Gidiosaurus_Fingers` and enable the **Fake User**:

Renaming the fingers action and enabling the Fake User

43. Now *Shift*-select for both the **.L** and **.R** bones, **thumb.01**, **thumb.02**, and **thumb.03**, **f_index.01** and **f_index.02**, **f_middle.01** and **f_middle.02**, and the **palm** control bones, then add a **Rotation** key:

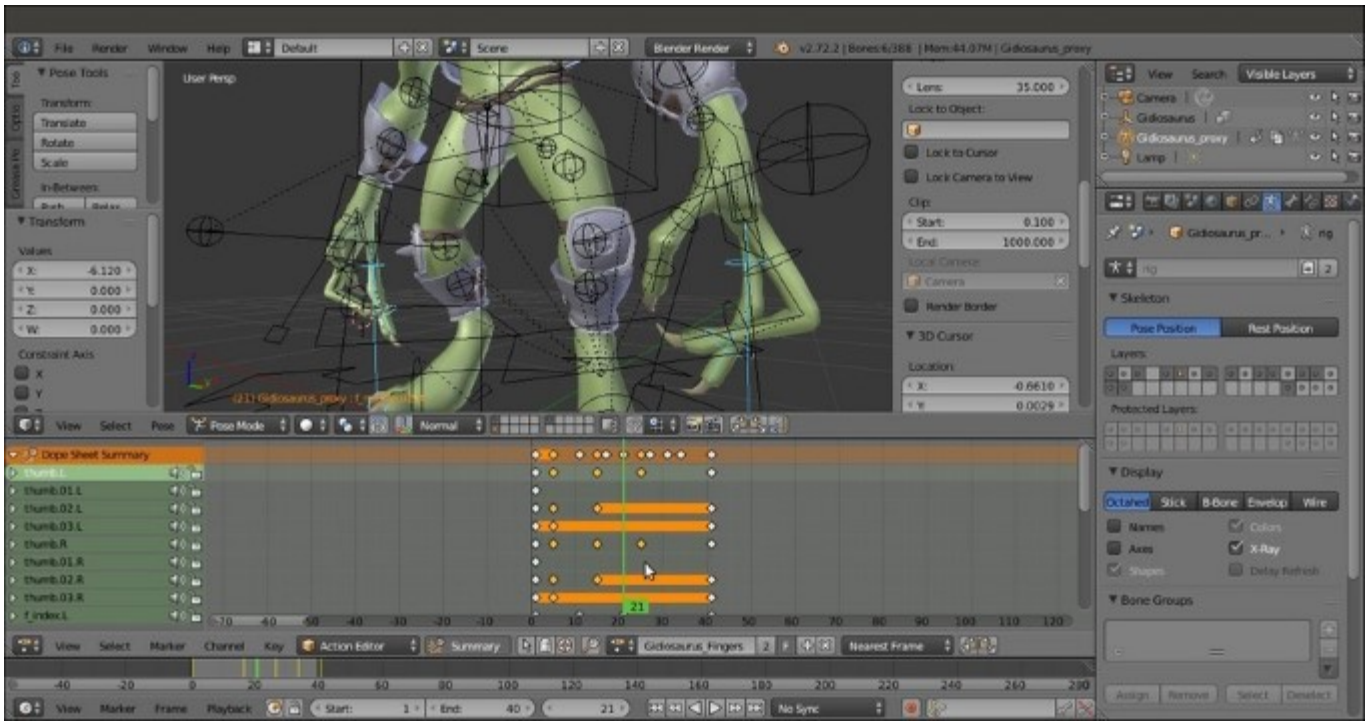
Adding a first rotation key for all the fingers at the same time

44. Now that we have all the finger bones' names in the **Action Editor** list-tree to the left (the **Channel Region**), start to click on the bone names to highlight them, for example, click on the **thumb.L** item, then press *Shift* + *PageUp* keys to move it to the top of the list.
45. Then highlight the **thumb.01.L** bone and by pressing the *PageUp* arrow, move it right after the **thumb.L** bone (press *Shift* + *PageUp* to eventually go directly to the top). Repeat with the **thumb.02.L** and the **thumb.03.L** bones, then go to the **thumb.R** bone, and so on. To move an item downward in the list-tree, simply press the *PageDown* key instead (or *Shift* + *PageDown* to go directly to the bottom).
46. Repeat the ordering until you have *grouped* the bones' names *by finger* in the list-tree, to make it easier to individuate them in the **Action Editor** window, then use *Shift* + *D* to duplicate all the keys and move the duplicated ones to frame **41**.
47. Go to frame **21**, select **thumb.L**, **.R**, **f_index.L**, **.R**, **f_middle.L**, and **.R** bones and press *S* to scale them to **0.900**. Assign a **Scaling** key, then select and rotate the other control bones, and assign **Rotation** keys (be aware that the previous scaling bones can also be rotated). Also, by using the *Copy/Paste* technique already shown, build a kind of *creepy hands* animation:



The "creepy hands" animation made by rotating and scaling the bones controls

48. When you are done, thanks to the re-ordering we made in the **Channel Region**, go to the **Action Editor** window and move groups of keys based on their finger group; in short, to avoid the *everything-at-the-same-time* issue, dislocate the timing of one finger with respect to the others:



Offsetting the finger' keys

49. After this, click on the *Display number of users of this data* button to create a new copy of the action and change the name to **Gidiosaurus_Fingers.L**. In the **Channel Region**, *Shift*-select all the **.R** bones items and delete them (*X* key), then enable the **Fake User**.
50. Click on the double arrows icon to the left side of the datablock name (*Browse Action to be linked*) and reselect the **Gidiosaurus_Fingers** action.
51. Again, click on the *Display number of users of this data* button to create a new copy of the action and change the name to **Gidiosaurus_Fingers.R**. *Shift*-select all the **.L** bones items and delete them. Enable the **Fake User** and click on the **X** icon button to unlink the action datablock.
52. Save the file.

To have a look at the completed walk cycle of the **Gidiosaurus** and the other actions, open the `Gidiosaurus_walkcycle_final.blend` file provided with this cookbook.

How it works…

An **Action** is a bones **F-Curves** datablock created at the same moment any animation key is added through the **Insert Keyframe Menu** (*I* key) or the **red button icon** (*Automatic keyframe insertion for Objects and Bones*) in the **Timeline** toolbar. The newly created **Action** automatically takes the name from the object itself (**Gidiosaurus_proxy** in this case) plus the **Action** suffix.

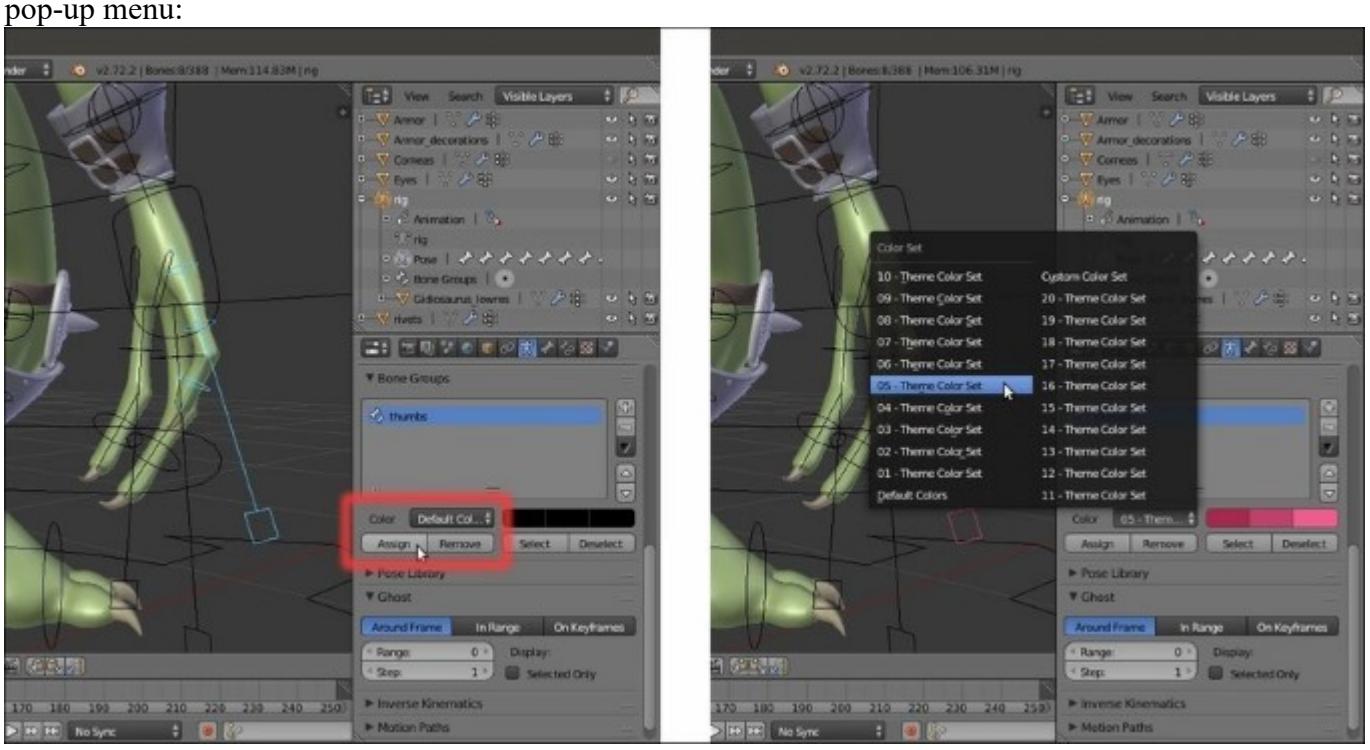
The **Actions** are stored inside the `.blend` file, but thanks to the **Fake User** they don't necessarily need to be linked to the rig to be preserved after saving and closing the file.

Note that the scaling operation for the selected keys of an **Action** in the **Action Editor** window (and the same for the **Graph Editor** and the **NLA Editor**) use the **Time Cursor** position as the pivot point. Also note that even though we did it in our recipe, it wasn't mandatory in this case to declare the *x* (horizontal) axis for the scaling.

There's more…

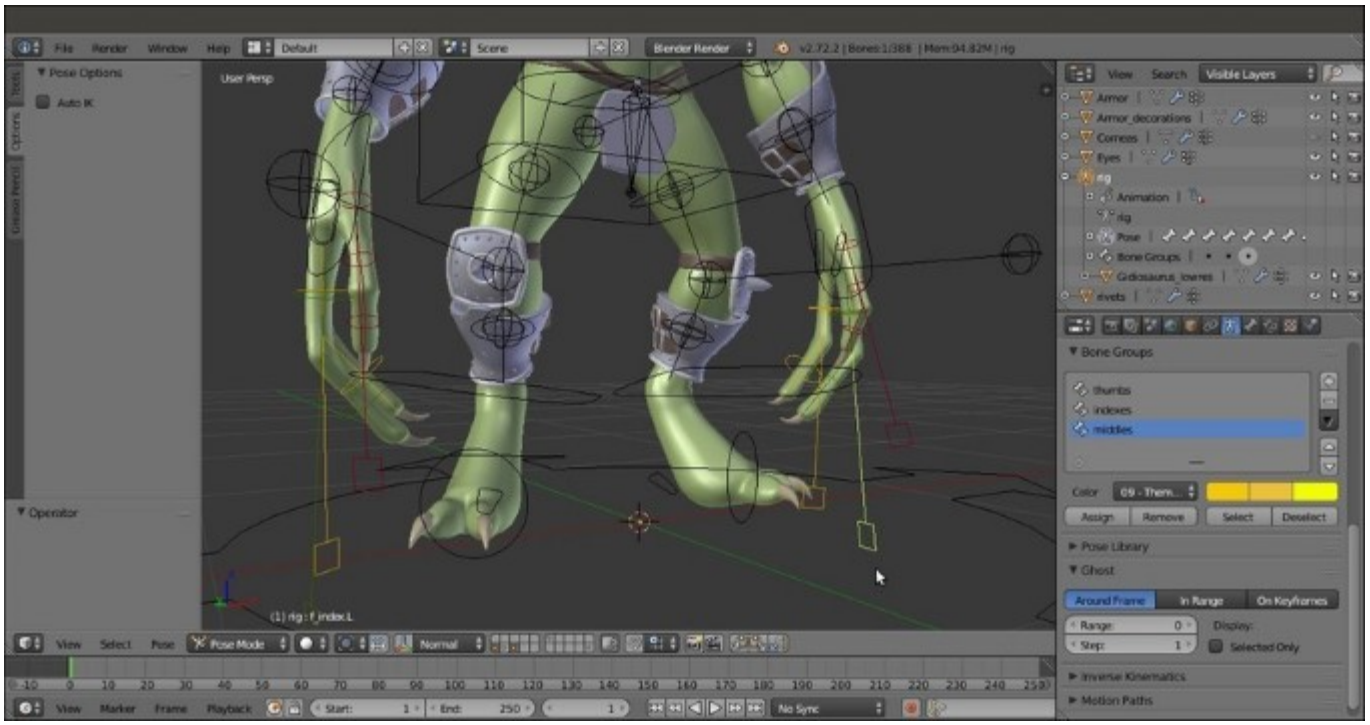
Organizing the bones' names in the list inside an action in the **Action Editor** window is a good way to quickly find the required item, but it can be improved even further by **Bone Groups**:

1. Open the `Gidiosaurus_library.blend` file and go to the **Outliner**; click on the eye icon to the side of the **rig** item to unhide it.
2. Select the **rig** and go to the **Object Data** window, then in the **Bone Groups** subpanel, click on the + icon to add a **bone group**.
3. Double click on it to rename it **thumbs**, then go into the 3D viewport and *Shift*-select all the **thumbs'** bones.
4. Click on the **Assign** button, then click on the **Color** slot to choose a **Theme Color Set** from the pop-up menu:



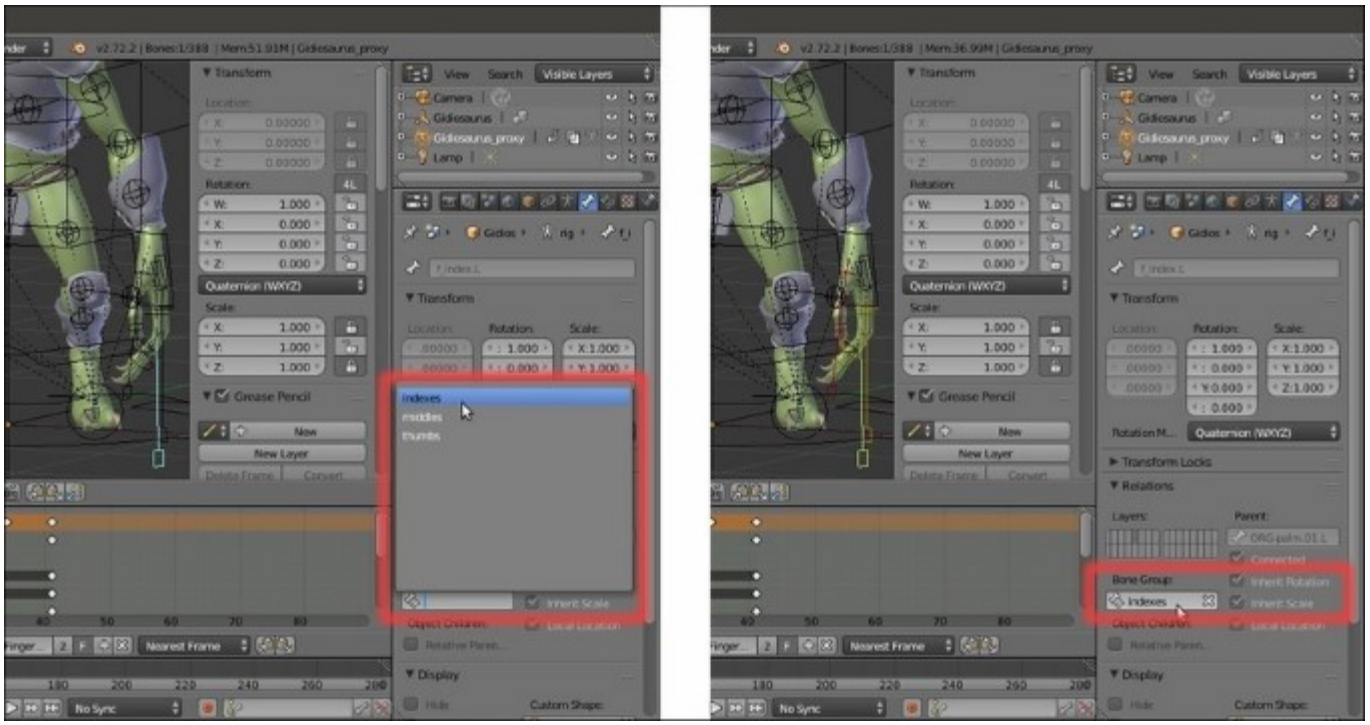
Choosing a Theme Color Set for the Bone Group

5. Repeat the steps from 2 to 4 for the other two fingers, thus creating the **indexes** and **middles** bone groups and selecting a different **Theme Color Set** option for each group:



Three different Bone Groups

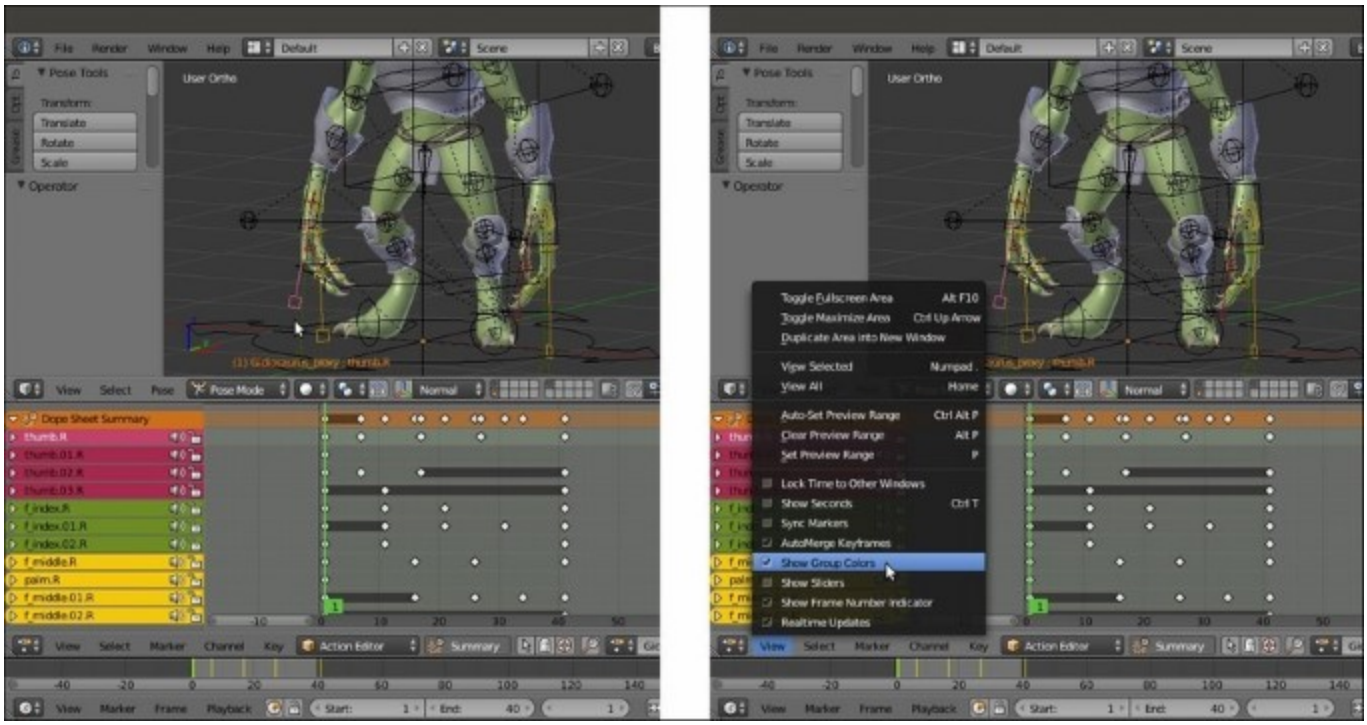
6. In the **Outliner**, hide the **rig** item again and save the file.
7. Re-open the `Gidiosaurus_walkcycle.blend` file; the colored bones don't show in the **proxified rig**, and this is because we had already proxified it and only later assigned the **bone groups** to the library file.
8. The solution to fix this is simply to select the affected bones one at a time and by going to the **Relations** subpanel under the **Bone** window, click on the **Bone Group** empty field to select the name of the appropriate group:



Reassigning the Theme Color Set to the proxified bones

By the way, it is always better to do the **Bone Groups** before the **proxy**, if possible.

The colors of the **Bone Groups** also show as *background color* for the bone channels inside the **Action Editor** window, making it a lot easier to select all the bones of a group; just be sure to have the **Show Group Colors** item enabled in the **View** menu on the **Action Editor** toolbar:



The Group Colors enabled for the bones

You can find the library with the colored fingers' control bones under the alternative file named `Gidiosaurus_library_colors.blend`.

See also

The walk cycle and the other actions we built in this recipe are, from an *animation point of view*, very simple and basic, not meant to teach you *how to animate* but only to show enough of Blender's tools for you to easily start animating a rigged character.

If you want to go deeper into the animation process, in Blender or not, here are some links to visit:

- http://www.fjasmin.net/walk_cycle_tutorial/index.html
- http://cgcookie.com/blender/2010/01/24/learning-basic-animation-and-a-walk-cycle/
- http://wiki.blender.org/index.php/Doc:2.6/Manual/Animation
- http://wiki.blender.org/index.php/Doc:2.6/Manual/Animation/Techs

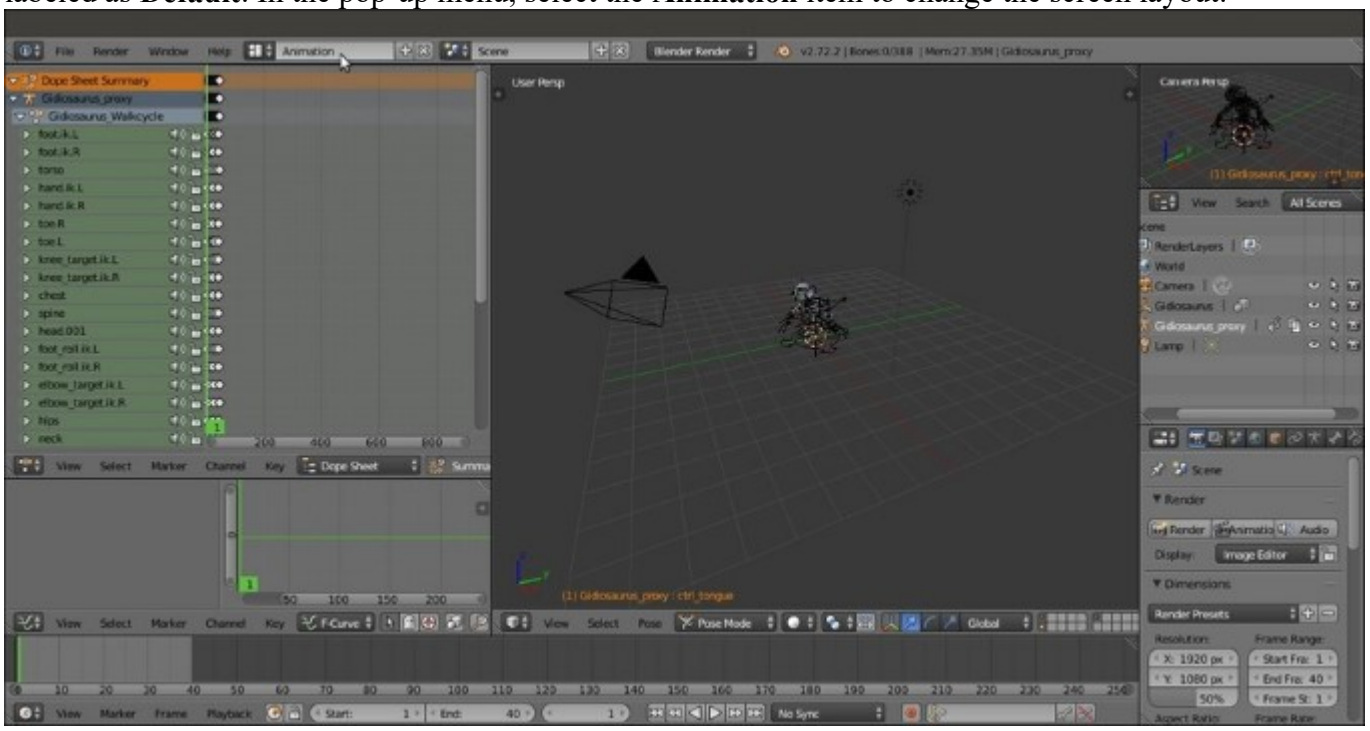
Tweaking the actions in Graph Editor

In the previous recipe, we built **Actions** by setting position, rotation, and/or scaling keys, which Blender **interpolates through F-Curves** to create the character's animation. In this recipe, we are going to see the **Graph Editor** window, a tool to modify these **F-Curves** to fix errors or fine-tune the movements of the animated character.

Getting ready

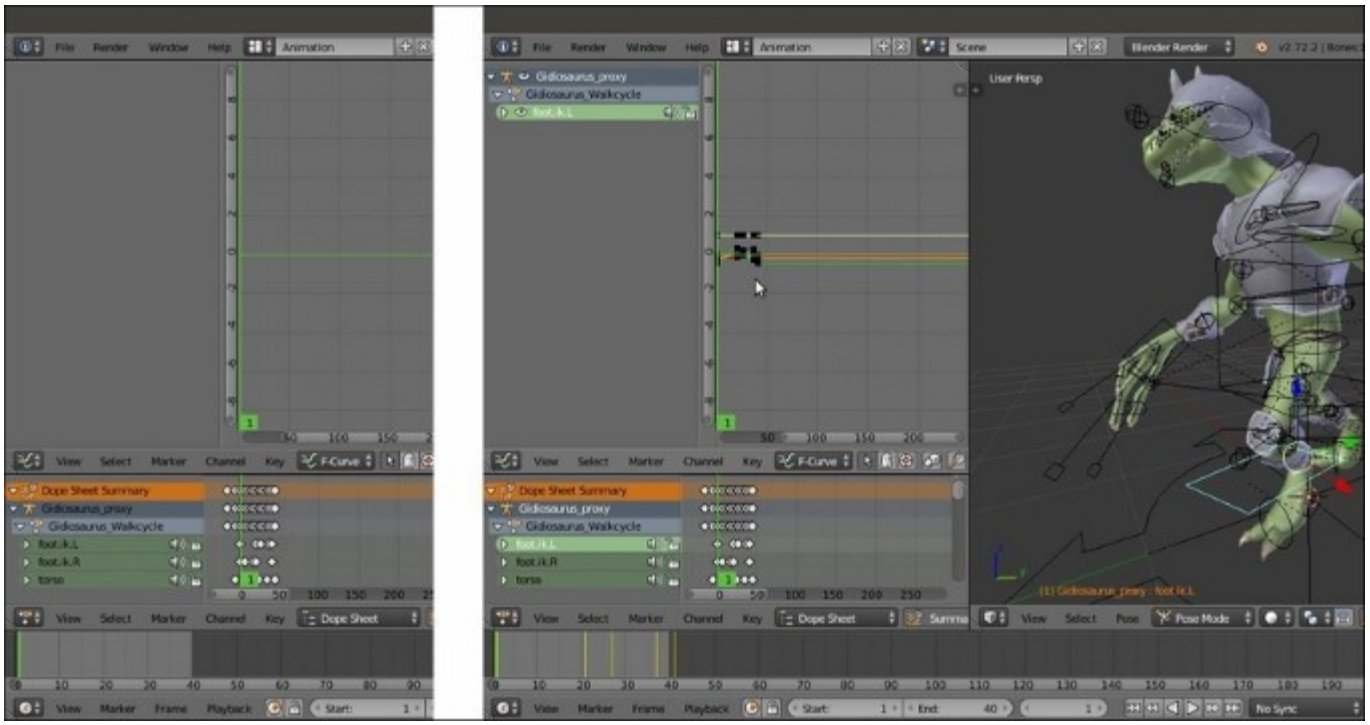
Open the `Gidiosaurus_walkcycle.blend` file.

1. If it's not already loaded, in the **Action Editor** window, load the **Gidiosaurus_walkcycle** action.
2. Go to the top main window header and click on the two little arrows to the left side of the button labeled as **Default**. In the pop-up menu, select the **Animation** item to change the screen layout:



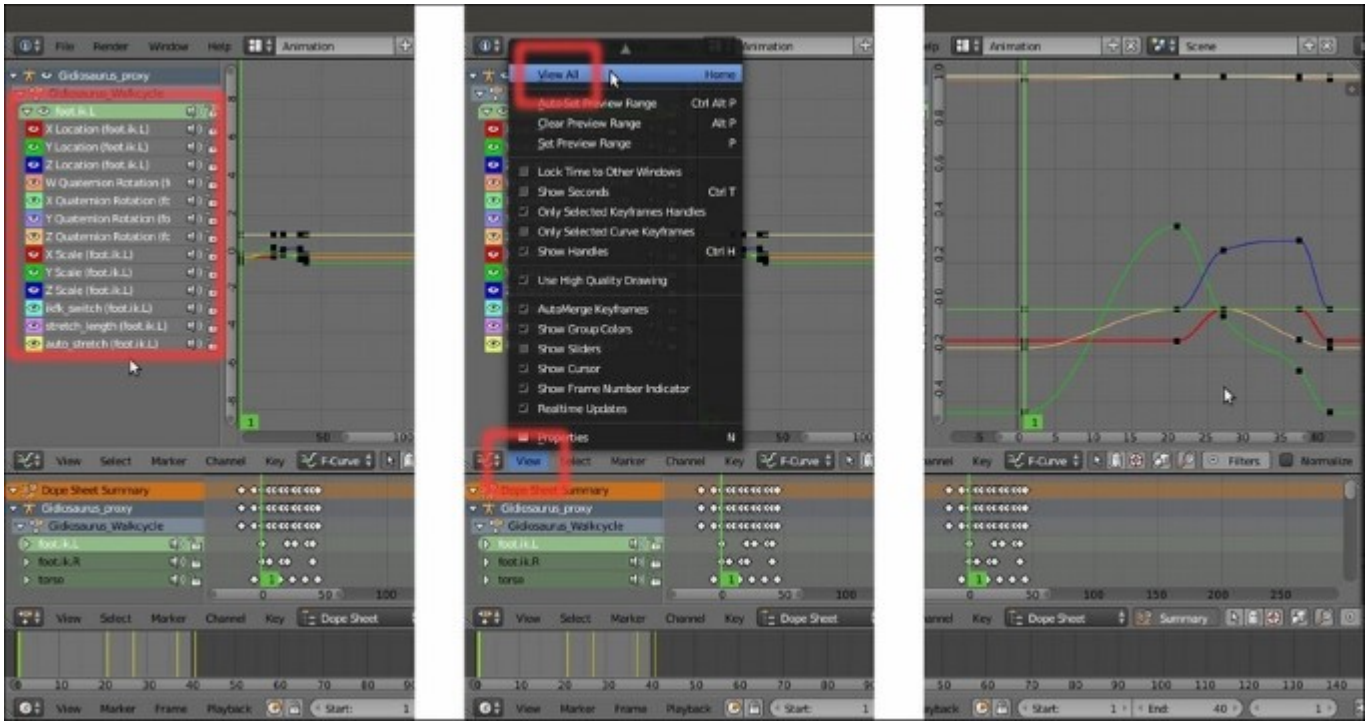
The premade Animation screen layout

3. Press the *Ctrl* key and left-click on the **top right corner** of the **Dope Sheet** window, then drag the mouse towards the **Graph Editor** window below to switch the two windows.
4. Go to the 3D viewport and zoom to the character to select the **foot_ik.L** bone; the **F-Curves** for the selected bone appear in the **Graph Editor** window:



The F-Curve of the animation keys of the selected bone in the Graph Editor window

5. Expand the **foot_ik.L** item in the **Graph Editor** list-tree by clicking on the little arrow to the side of the item itself, then click on the **View** item in the toolbar and select the **View All** item to better visualize the curves inside the **Curve Editor** area:



The list of the available F-Curves for the selected bone and the automatic zoom through the View All item

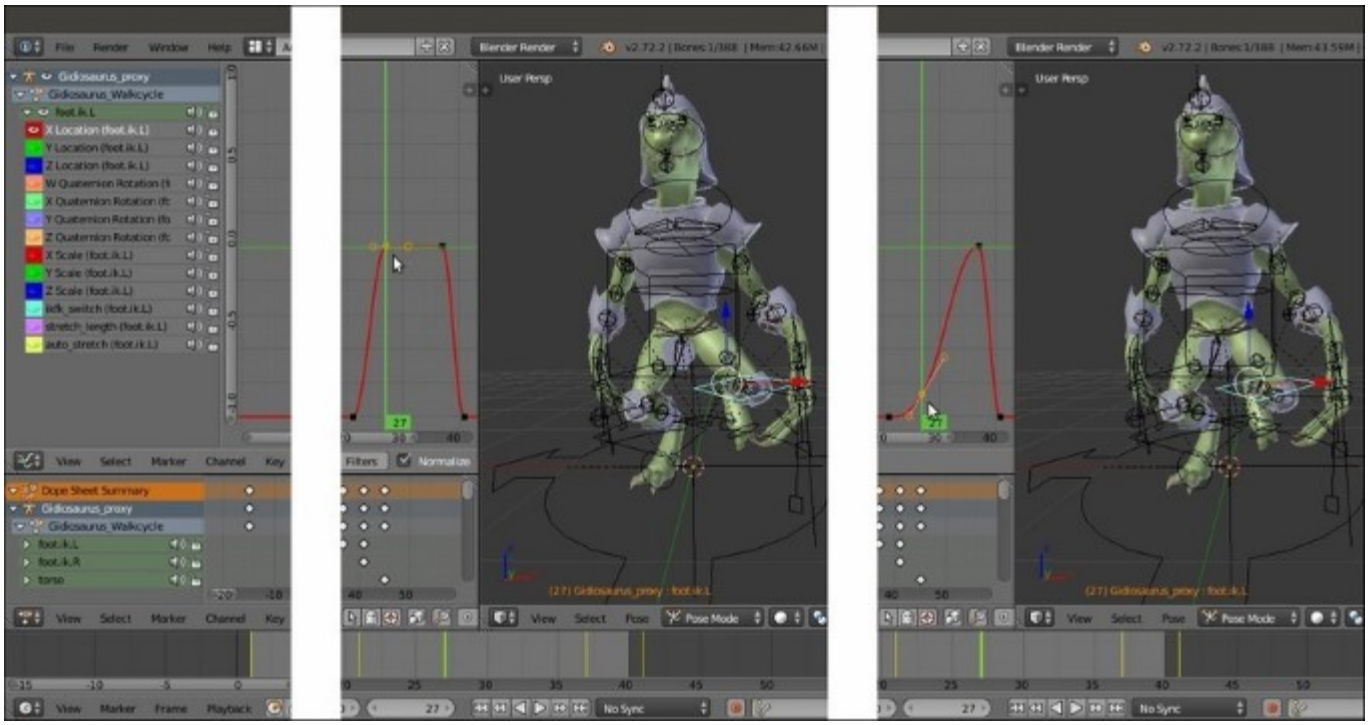
6. Hide (by clicking on the eye icon) and/or delete (select using left-click and the *X* key) the unnecessary curve items such as (at least in this case) **X Scale**, **Y Scale**, **Z Scale** or **ikfk_switch (foot_ik.L)**, and so on. Join the unnecessary windows together and adjust the size of the **Edit Area** (the part with the keyframes) in the **Dope Sheet** to make them more easily readable. Optionally, enable the **Normalize** item in the **Graph Editor** toolbar to show all the **F-Curves** in a normalized **-1** to **1** range.
7. Save the file as `Gidiosaurus_F-Curves.blend`:

The Animation screen with a bit of customization

How to do it…

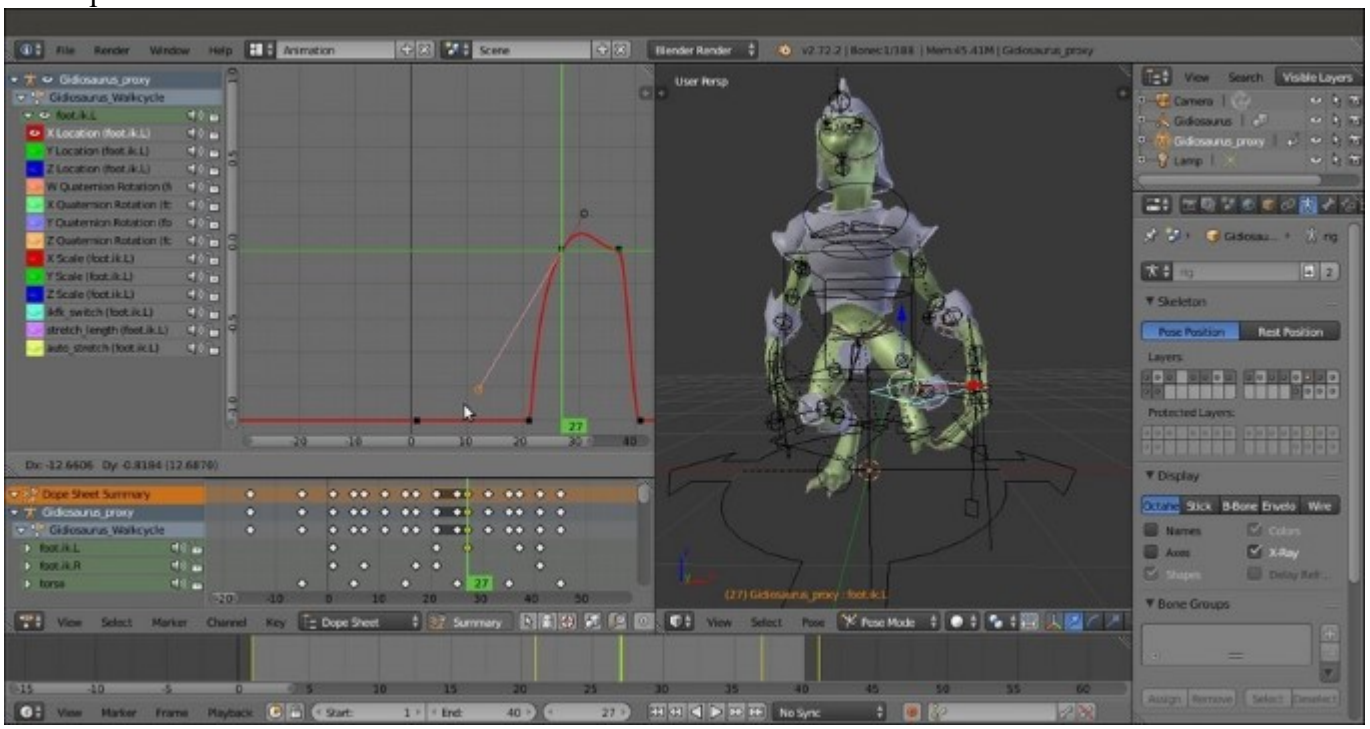
By selecting a curve name item and/or hiding the others in the **Graph Editor** list-tree, we can concentrate on one curve at a time. For example, what if we want to change the position of the **right foot** at frame **27** on the global *x* axis, when it's high off the ground?

1. Just left-click on the **X Location (foot_ik.L)** item in the list to highlight it and/or simply hide the others. Right-click on the curve keyframe/control point at frame **27** to reveal the **handles**, then press *G* | *Y* to move the keyframe handles on the vertical axis and see the foot move accordingly in the viewport on the global *x* axis:



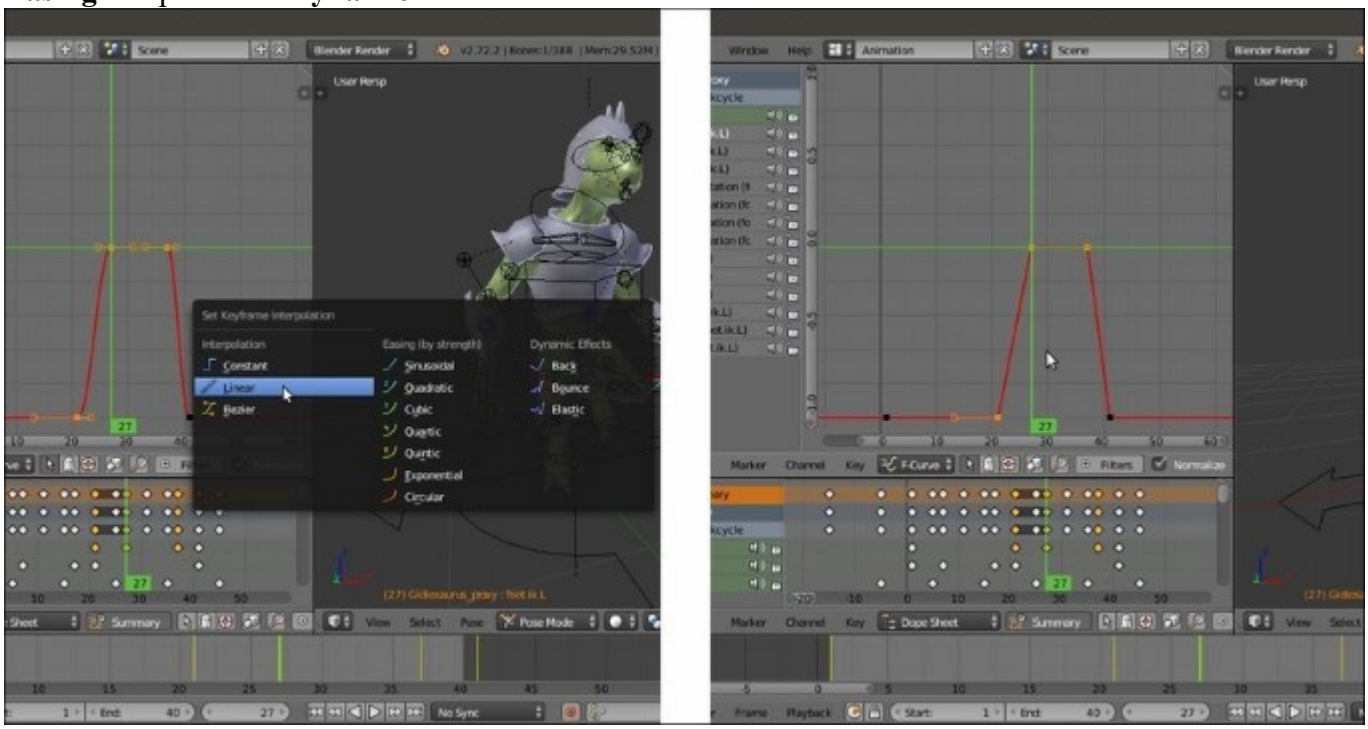
Editing the points of the F-Curve to tweak the bone's position

2. Or else, right-click only on one of the handles of the keyframe to move it and change the curve's envelope:



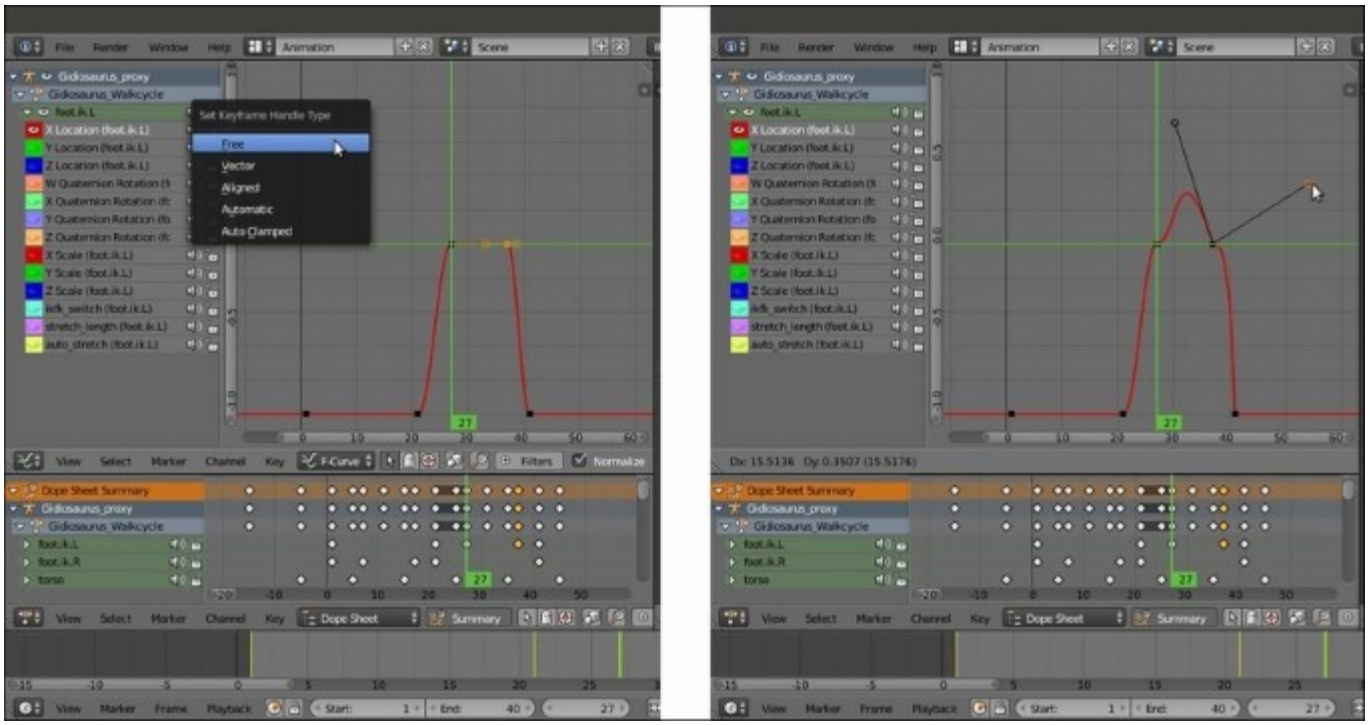
Changing the envelope of the F-Curve by modifying one of the point's handles

3. By *Shift*-selecting two or more keyframes of an **F-Curve** and pressing the *T* key, it is possible to set the **interpolation** type through the **Set Keyframe Interpolation** pop-up menu; by default the **F-Curves** are **Bezier**, but they can be switched to **Linear** or **Constant**. There are also **Easing** and pre-made **Dynamic** effects:



Changing the F-Curve Interpolation mode

4. Finally, the handles' type can also be set through the **Set Keyframe Handle Type** menu by pressing the *V* key; by default the handles are **Aligned**, but they can be set as **Free**, **Vector**, **Automatic**, and **Auto Clamped** too:



Changing the handle's type to further tweak the curve's envelope

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Animation/Editors/Graph

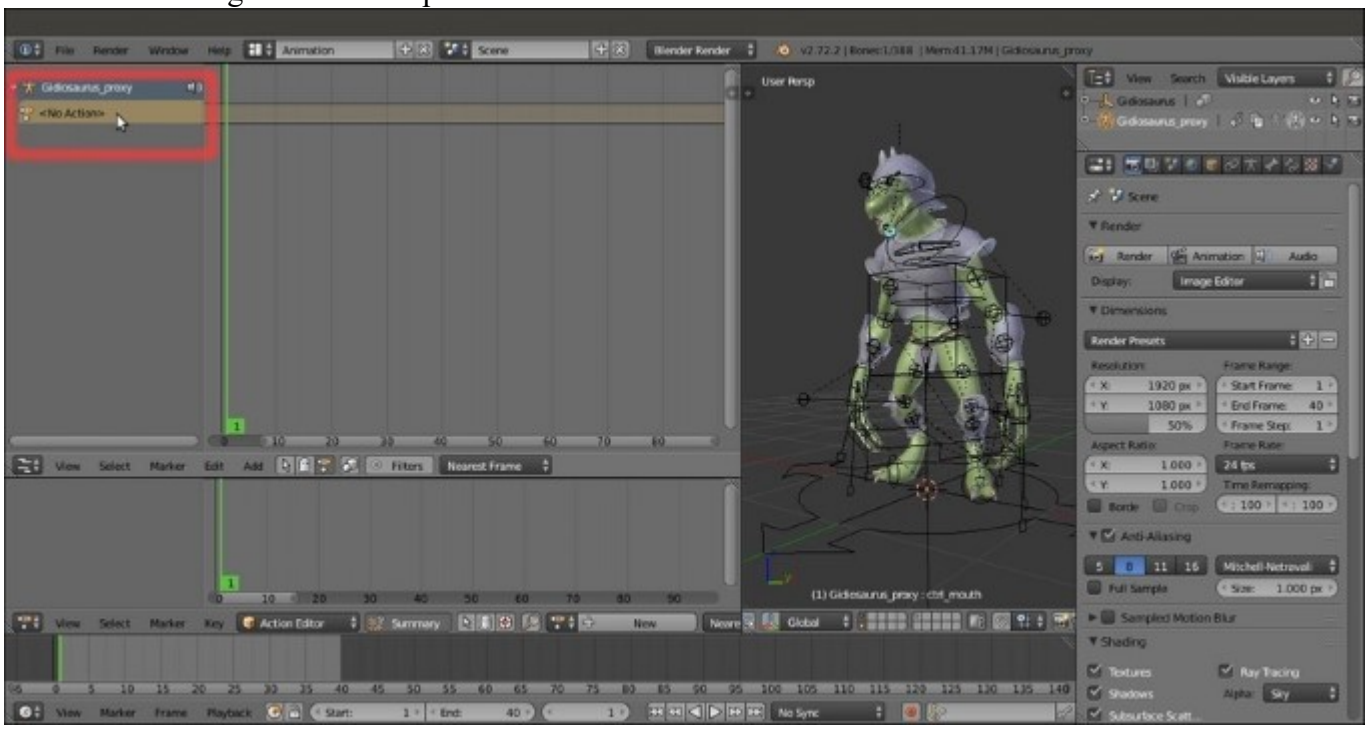
Using the Non Linear Action Editor to mix different actions

It's finally time to use the **NLA Editor** to compose a longer animation using the actions we built in the previous recipes.

Getting ready

As usual, first let's prepare the screen:

1. Start Blender and press *Ctrl* + *Alt* + *U* to call the **User Preferences** panel; in the **Editing** tab, enable the **Allow Negative Frames** item.
2. Click on the **Save User Settings** button and close the panel.
3. Load the `Gidiosaurus_F-Curves.blend` file and switch the **Graph Editor** to the **NLA Editor** window, and the **Dope Sheet** below it with the **Action Editor** window.
4. If necessary, click on the **X** icon button in the **Action Editor** window toolbar to unlink any action from the rig and clear the pose:



The Animation screen with the (still empty) NLA editor window

How to do it…

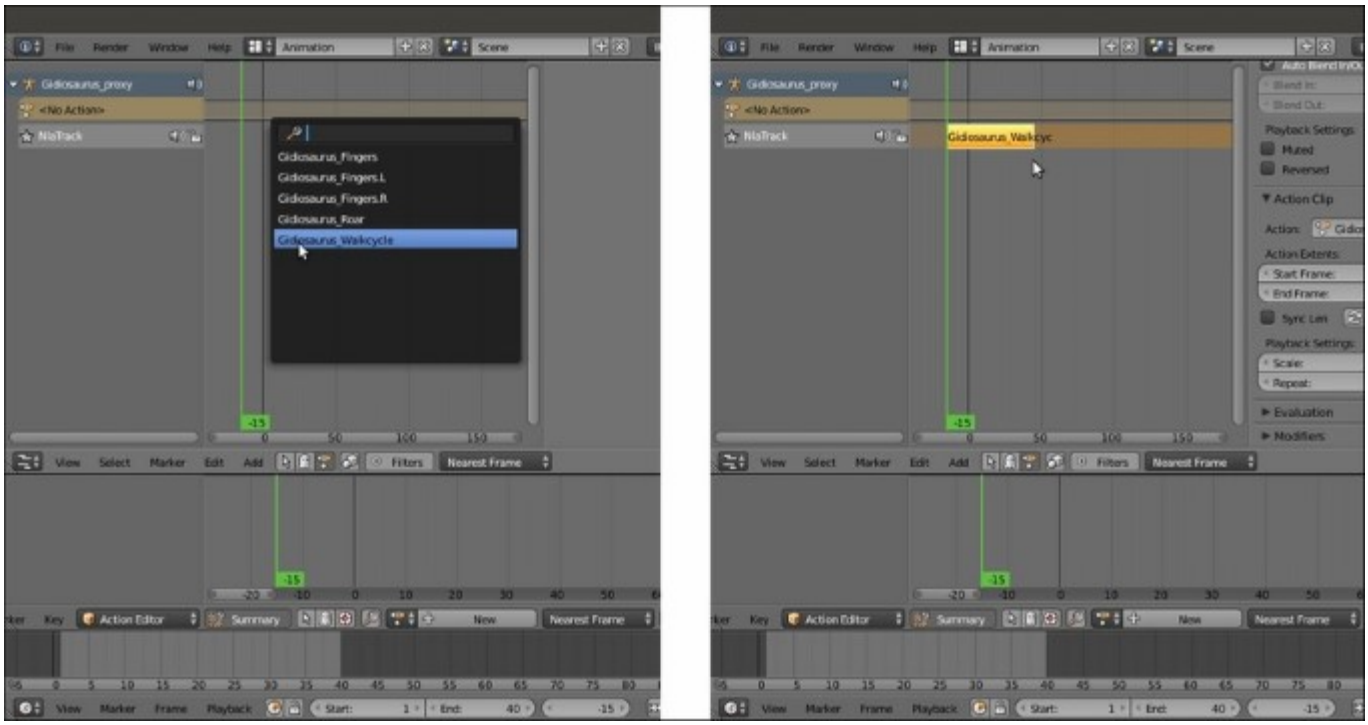
We are going to add **Action strips** to the **Gidiosaurus_proxy** rig, so it's mandatory to have at least one bone selected (any one, but in this case, it's the **ctrl_mouth** bone):

1. Put the mouse cursor in the **Track Region** (**NLA-stack**) of the **NLA Editor** window, right under where it shows **Gidiosaurus_proxy** | **<No Action>** items, and press *Shift* + *A* to add **a NlaTrack** channel:

Adding a first track to the NLA Editor window

Now, take a moment and load the **Gidiosaurus_walkcycle** action in the bottom **Action Editor** window to see the action extension; it starts at frame **-15** and ends at frame **45**.

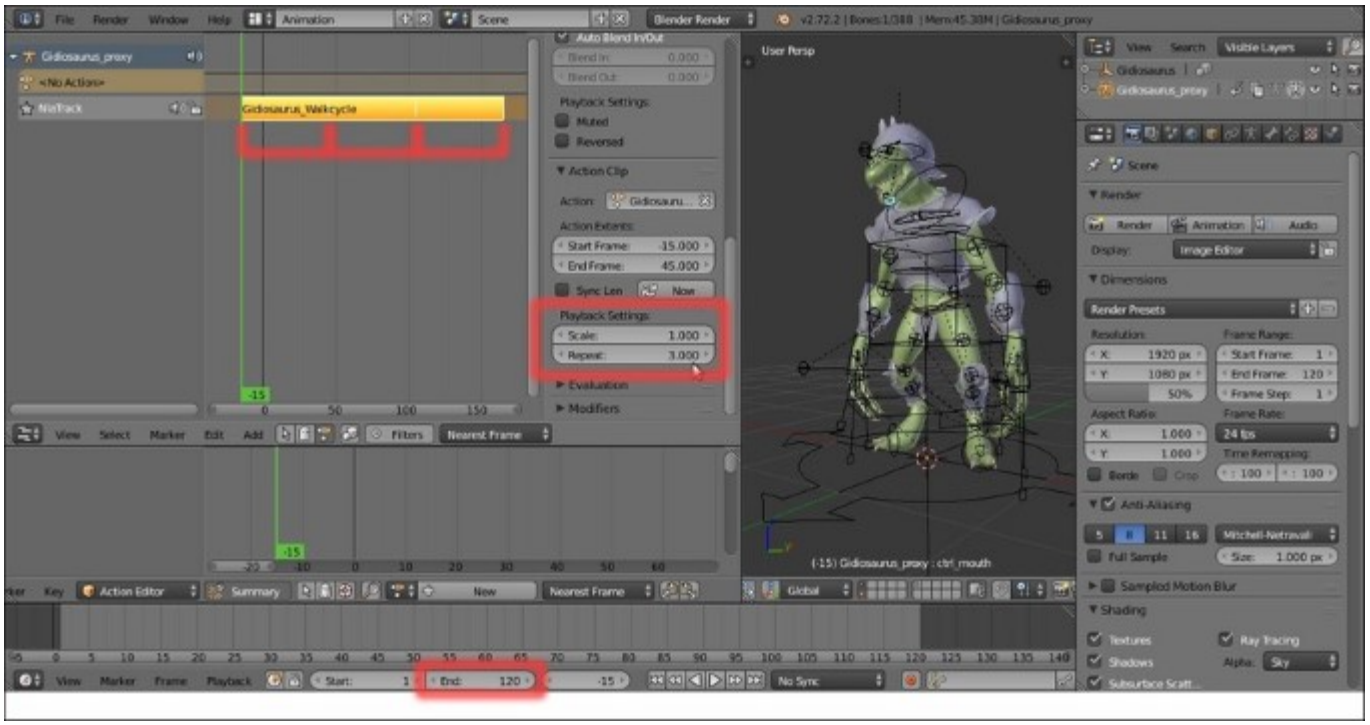
2. Unlink the action in the **Action Editor** and move the **Time Cursor** to negative frame **-15**; put the mouse cursor in the **Strip Edit** area to the right side of the **NlaTrack** item and again press *Shift* + *A*. From the pop-up menu, select the **Gidiosaurus_Walkcycle** item:



Loading the Gidiosaurus_walkcycle action into the track

A yellow action strip, with the **Gidiosaurus_Walkcycle** name superimposed, is added to the track at the **Time Cursor** location (the vertical green bar showing the frame number. If you now press the **Play** button in the **Player Control** on the **Timeline** toolbar, the animation starts at frame **1** and because the animation is only **40** frames long, it loops correctly, exactly as if the action was loaded in the **Action Editor** window.

3. If not already present, press the *N* key to call the **Properties** sidepanel of the **NLA Editor** window, right-click on the action strip to select it, and then go to the **Action Clip** subpanel. Under **Playback Settings**, set the **Repeat** value to **3.000**.
4. Click on the **End** button in the **Timeline** toolbar and change the frame value from **40** to **120** (**40** frames x **3**):

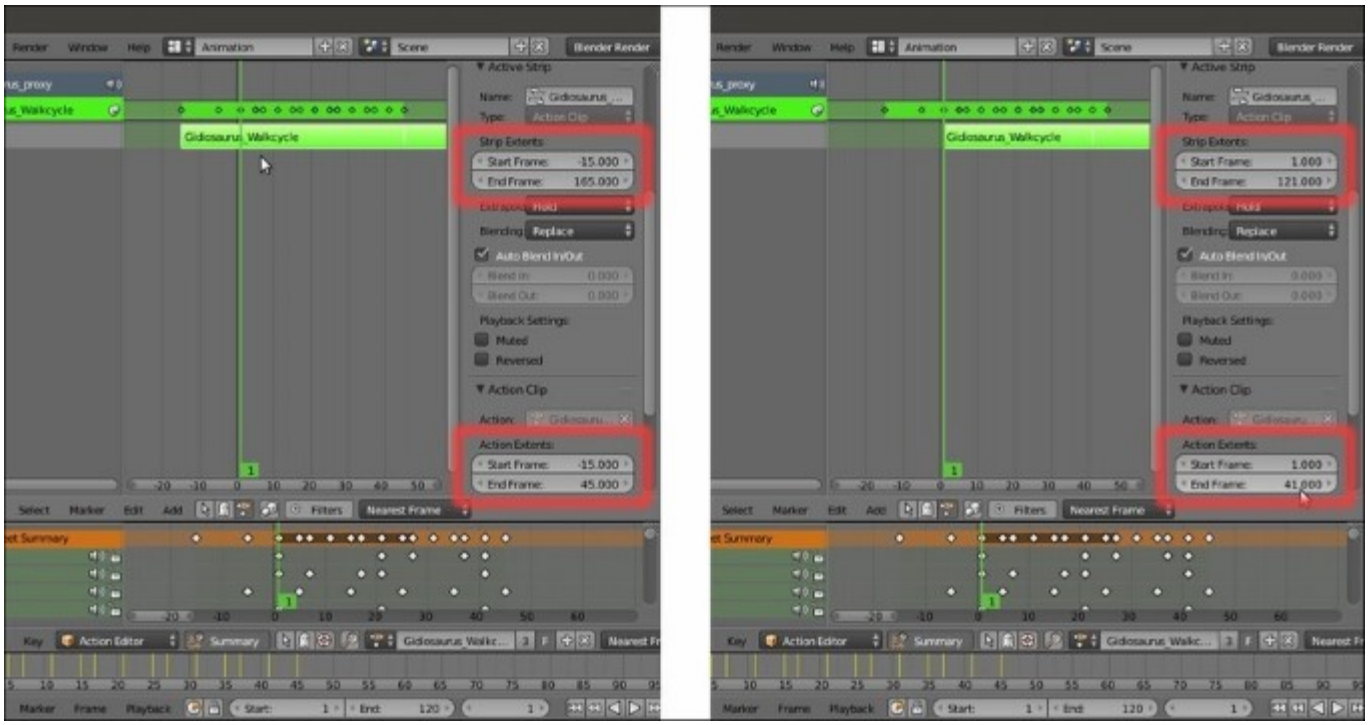


The Gidiosaurus_walkcycle action set to be repeated three times

If you press the **Play** button now, the animation is repeated **3** times *but* it doesn't loop correctly anymore because the **negative frames** keys are also included, in both the second and third repetitions. This is because we loaded the action at frame **-15**, so this is the **Start Frame** value for **Action Extents** (**Start Frame** = **-15**, **End Frame** = **45**).

Hence, some adjustment must be done to the action strip:

5. First, move the **Time Cursor** to frame **1**; with the strip selected, press the *Tab* key to go into **Edit Mode** and make the inner keys of the strip visible, both above the strip in the **NLA Editor** window, and as an **Action** in the **Action Editor** window. This way it's simpler to understand what keys are at what frame, and so on.
6. Second, go to the **Active Strip** subpanel and under the **Strip Extents** item, set the **Start Frame** value to **1.000**.
7. Go to the **Action Clip** subpanel and under the **Action Extents** item, set **Start Frame** to **1.000** as well and the **End Frame** value to **41.000**:

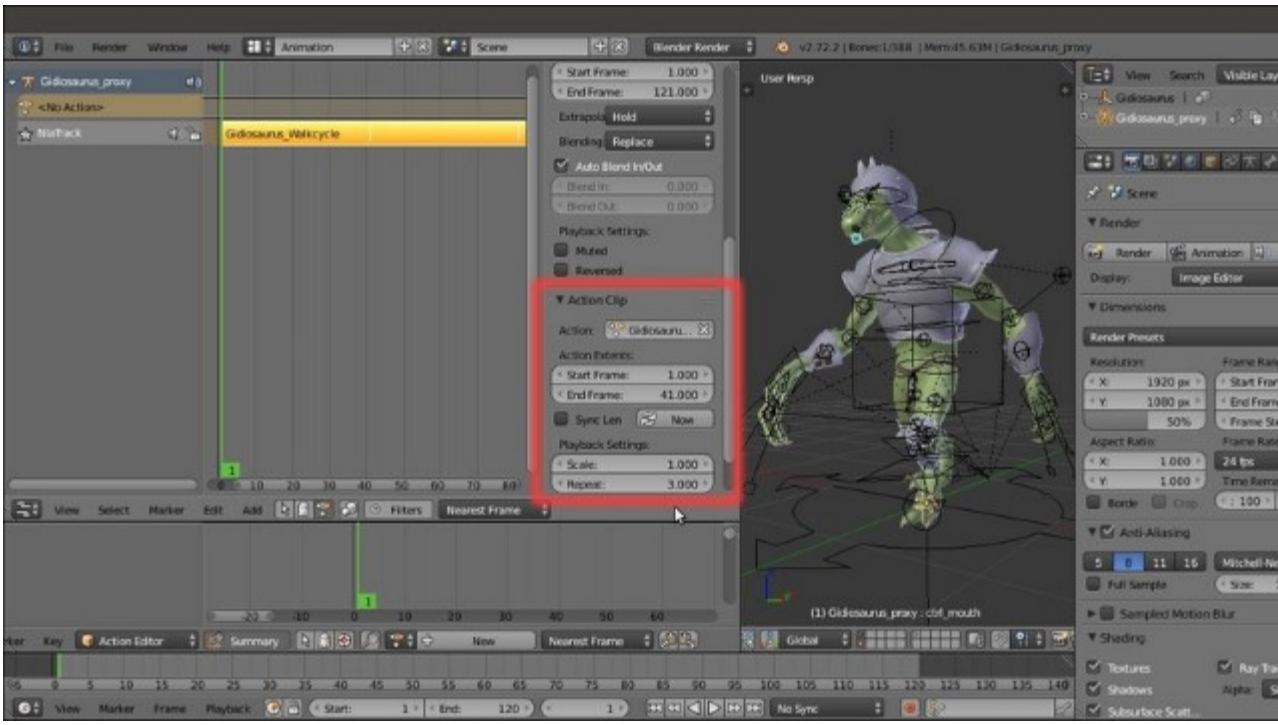


The action in Edit Mode and the Strip Extents and Action Extents values in the Properties subpanel of the NLA window

8. Press *Tab* to go out of **Edit Mode**.

   Now, the walk cycle animation loops correctly for all the **120** frames, and obviously it is also possible to loop it even more by raising the **Repeat** value.

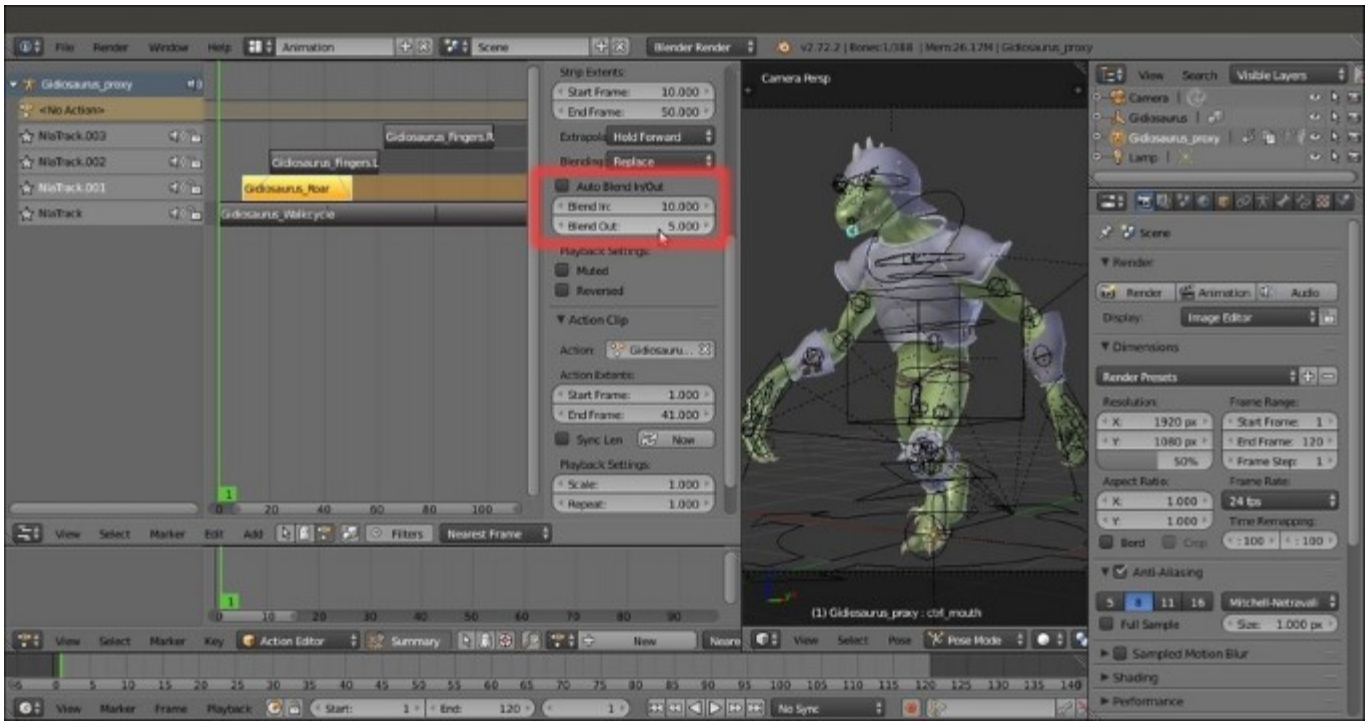
   So, the correct and fastest procedure would have been, from the start:
   1. At frame **1**, load the action strip in the **NLA Editor** window.
   2. In the **Properties** sidepanel, under the **Action Extents** item in the **Action Clip** subpanel, set the **Start Frame** value to **1.000** and the **End Frame** value to **41.000**.
   3. Under **Playback Settings**, set the **Repeat** value to **3.000** and the total length of the animation to **120** frames in the **End** button of the **Timeline** toolbar:



Recapitulating the action extents values to be set

Now, let's see how to add the other actions:

1. Put the mouse cursor under the **NlaTrack** item and press *Shift + A* to add a new track (**NlaTrack.001**); load the **Gidiosaurus_Roar** action strip and move it (*G* key) to start at frame **10**.
2. Select the **Gidiosaurus_Walkcycle** strip and in the **Active Strip** subpanel, disable the **Auto Blend In/Out** item but leave the values as **0.000**. Select the **Gidiosaurus_Roar** strip and disable the **Auto Blend In/Out** item as well, then set the **Blend In** value to **10.000** and the **Blend Out** value to **5.000**.
3. Add **two** more tracks, select the **NlaTrack.002** track and load the action strip **Gidiosaurus_fingers.L**, then select the **NlaTrack.003** track and load the action strip **Gidiosaurus_fingers.R**.
4. Select the **Gidiosaurus_fingers.L** strip and in the **Active Strip** subpanel, disable the **Auto Blend In/Out** item, and leave the values as **0.000**; repeat for the **Gidiosaurus_fingers.R** strip.
5. Move the two strips separately in different positions inside the **120** frames animation range.



Setting the Blend In and Blend Out values to mix the other actions

6. Press the **Play** button in the **Player Control** on the **Timeline** toolbar to watch the composited animation and save the file as `Gidiosaurus_NLA.blend`.

At this point it could be possible to start to render at least some **OpenGL** preview to see the result, but there are still several steps missing in our workflow before we reach the final goal, from the **texturing** to the **shaders**, **lighting**, and finally **beauty-rendering** and **compositing**; all stuff that we'll see in the next few chapters.

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Animation/Editors/NLA_Editor

Chapter 10. Creating the Textures

In this chapter, we will cover the following recipes:

- Making a tileable scales image in Blender Internal
- Preparing the model to use the UDIM UV tiles
- Baking the tileable scales texture into the UV tiles
- Painting to fix the seams and to modify the baked scales image maps
- Painting the color maps in Blender Internal
- Painting the color maps in Cycles

Introduction

In this chapter, we are finally going to create the textures for the **Gidiosaurus** character, meaning all the image textures that we'll need later, to build the **shaders** for the body and for the **armor**. Basically, the essential images we need are:

- A grayscale reptilian scales image to be used as a bump map and to color the skin
- Painted image textures for the skin diffuse coloration
- A tileable image for the worn armor metallic surface
- A bump image for the armor decoration patterns

In this chapter, we'll focus on the skin of the **Gidiosaurus**, and the last two textures for the **armor** will be treated in the next chapter.

The most difficult and tedious part is, no doubt, rendering the **scales** on the **Gidiosaurus** skin; I mean, if we had to paint the scales one by one. Instead, we'll try to obtain the complex scales pattern with the minimum effort possible, using a couple of techniques to speed up the work.

Trying to keep things clear, from now on we'll use *two different texture folders*: the usual `textures` one and a `textures_making` folder, where the latter is used to contain the images we need during the process to produce the final image textures.

Making a tileable scales image in Blender Internal

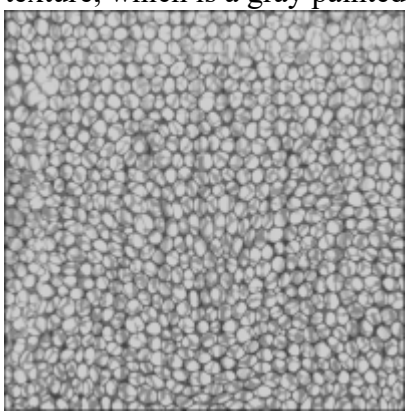
So, the first thing to do is to obtain a **tileable** grayscale **reptilian scales** image; we'll start from an already existing image obtained from an old and larger texture I had painted in **Gimp** for a dinosaur model some years ago… but that's a different story.

In any case, if you prefer, you can paint a new reptilian scales image from scratch by using painting software such as **Gimp** or **Photoshop** or open source applications such as **MyPaint** (http://mypaint.intilinux.com/) or **Krita** (https://krita.org/).

Getting ready

We are taking for granted that in your **Blender User Preferences** window, you still have the **Import Images as Planes** addon enabled; if not, start Blender and just enable it as already explained in Chapter 1, *Modeling the Character's Base Mesh*. Then, follow these steps:

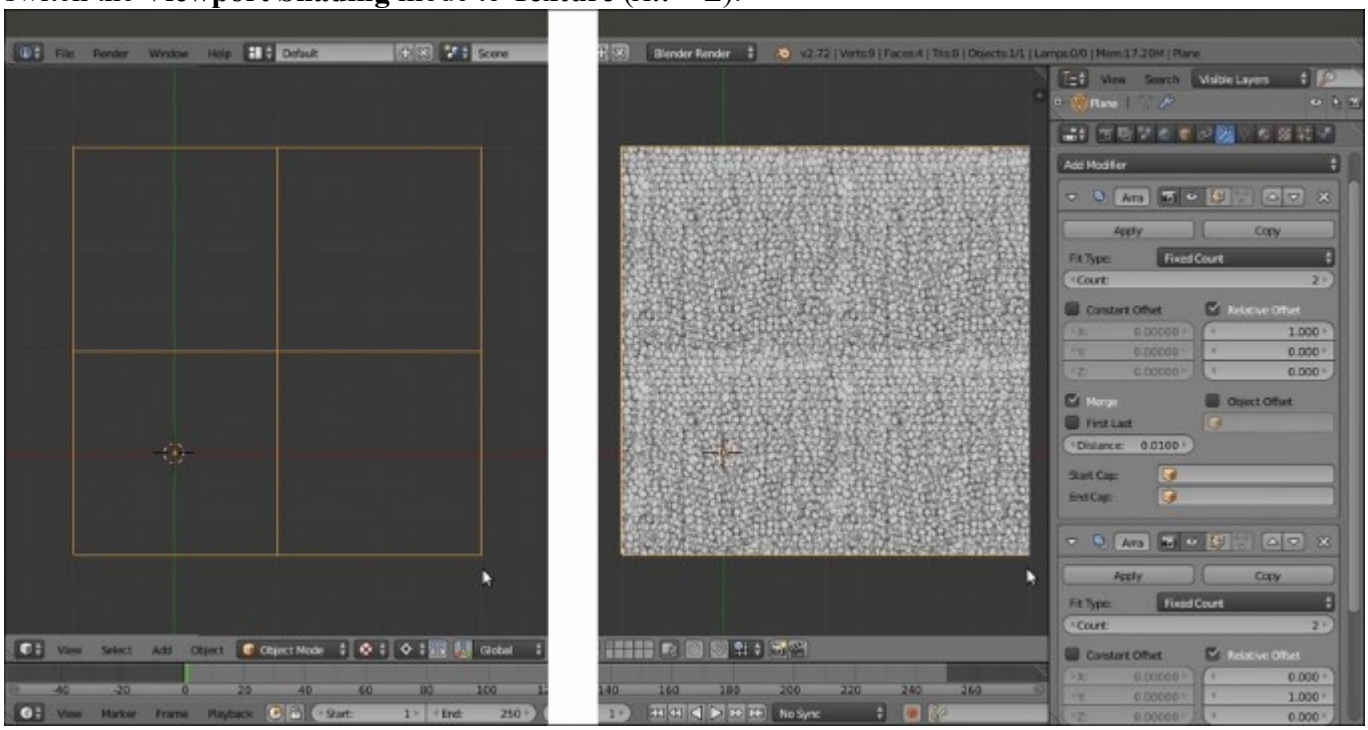
1. Select and delete the default **Cube** primitive in the scene. Select the **Camera** and the **Lamp** and move them to the **6th** scene layer.
2. Click on the main **File** menu and then on the **Import** item; select the **Images as Plane** item.
3. Browse to the `textures_making` folder and select the provided `scales.png` image texture, which is a gray painted scales image:



The "scales.png" image provided with this cookbook

4. Press the period (.) key on the numpad to center the view on the selected **Plane** and then the *7* key on the numpad to switch to the **Top Ortho** view.
5. Go to the **Object Modifiers** window and assign an **Array** modifier to the **Plane**; check the **Merge** item and leave all the other settings as they are.
6. Click on the **Copy** button to assign a new identical **Array** modifier, and in the new **Array** modifier, under the **Relative Offset** item, change **X** to **0.000** and **Y** to **1.000**.
7. Save the file as `4886OS_10_scales_tiles_01.blend`.

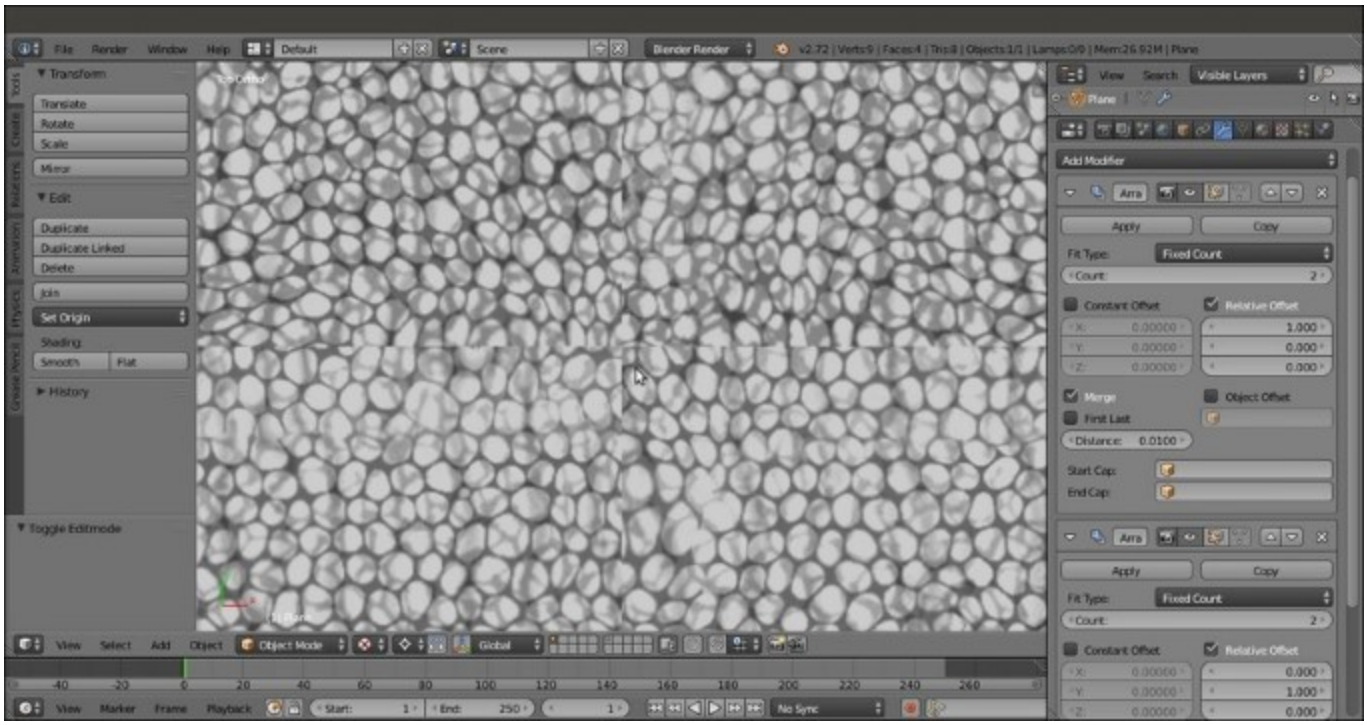
8. Press the period (.) key again on the numpad to center the view on the *enlarged* **Plane**, then switch the **Viewport Shading** mode to **Texture** (*Alt* + *Z*):



The UV mapped Plane with the Array modifiers assigned

As you can see in the preceding screenshot, the mapped **Plane** is now repeated **4** times.

By zooming towards the middle seams, it's clear that the mapped scales image is not tileable yet:



The visible seams at the borders of the Plane instances

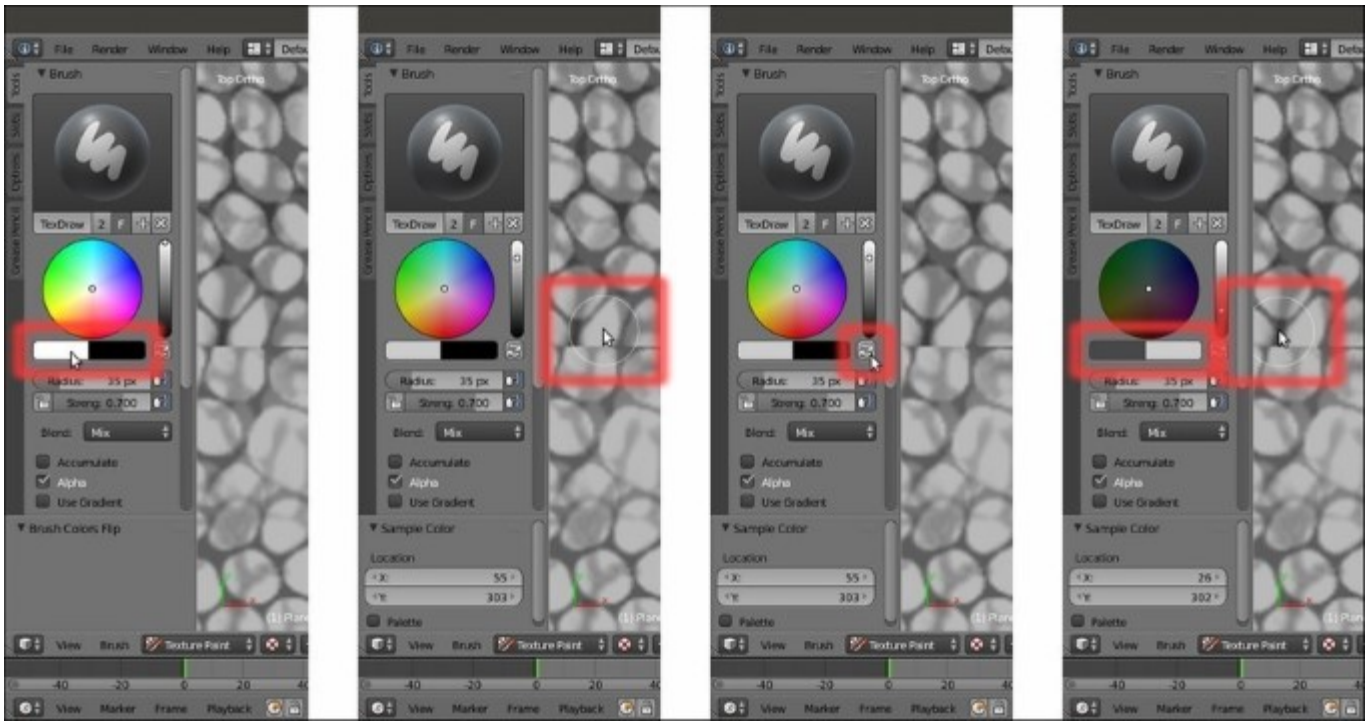
Tip

Just in case the previous screenshot is not readable enough, open the provided `4886OS_10_scales_tiles_01.blend` file to have a better look.

How to do it…

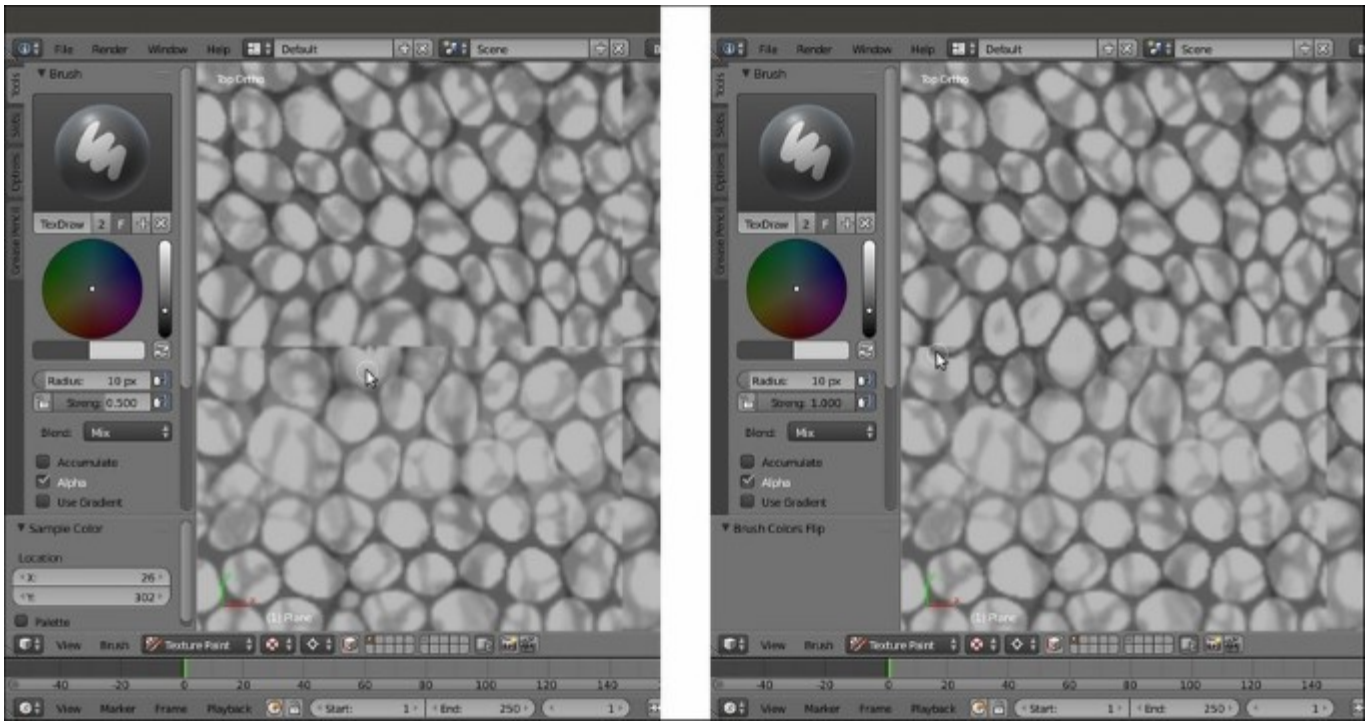
At this point, the file is ready and we can start to paint on the image to make it tileable:

1. Click on the mode button (*Sets the object interaction mode*) on the viewport toolbar to select the **Texture Paint** mode item.
2. Put the **mouse cursor** on a bright value in the scales image and press the *S* key to sample it, then go near the color selector in the **Brush** subpanel under the **Tool Shelf** to the left and click on the *Toggle foreground and background brush colors* button (the one with the two opposing arrows) to switch the active color. Otherwise, simply press the *X* key, put the mouse cursor on a dark area, and press *S* again to sample it as the opposite color.



Sampling the light and dark colors of the image

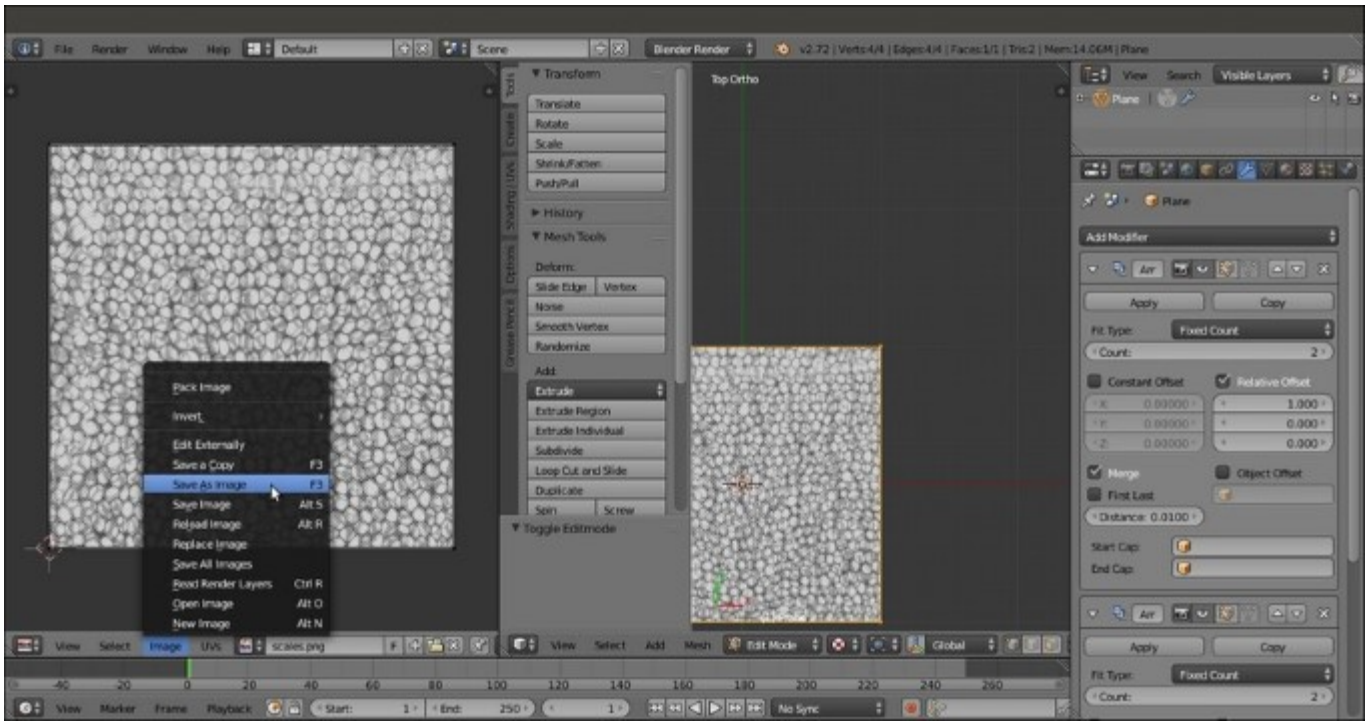
3. Set the brush's **Radius** and **Strength** values, and if you are using a **graphic tablet**, be sure to have the **two** *tablet pressure sensitivity* buttons at the sides of the previous items enabled.
4. Start to paint by fixing the scales on the image at the seams areas. Because we can also paint on the **Planes** duplicated by the **Array** modifier, and because we are always painting on the same instanced image, it's quite simple to visually join the scales at the four sides, actually making the image tileable:



Painting on the image at the borders to make the scales seamless

It's enough to fix the areas along the middle horizontal and vertical axes of the **Planes** to cover all the four edges (in fact, fixing **two** edges is automatically fixing **four**).

5. When you are done, go back into **Object Mode** and open a new window with **UV/Image Editor**; press *Tab* to go into **Edit Mode**, make the revised image appear, and save it (by clicking on **Image** | **Save as Image** in the toolbar or simply press the *F3* key) in the `textures_making` folder as `scales_tiles.png`:



Saving the tileable scales image with a different name

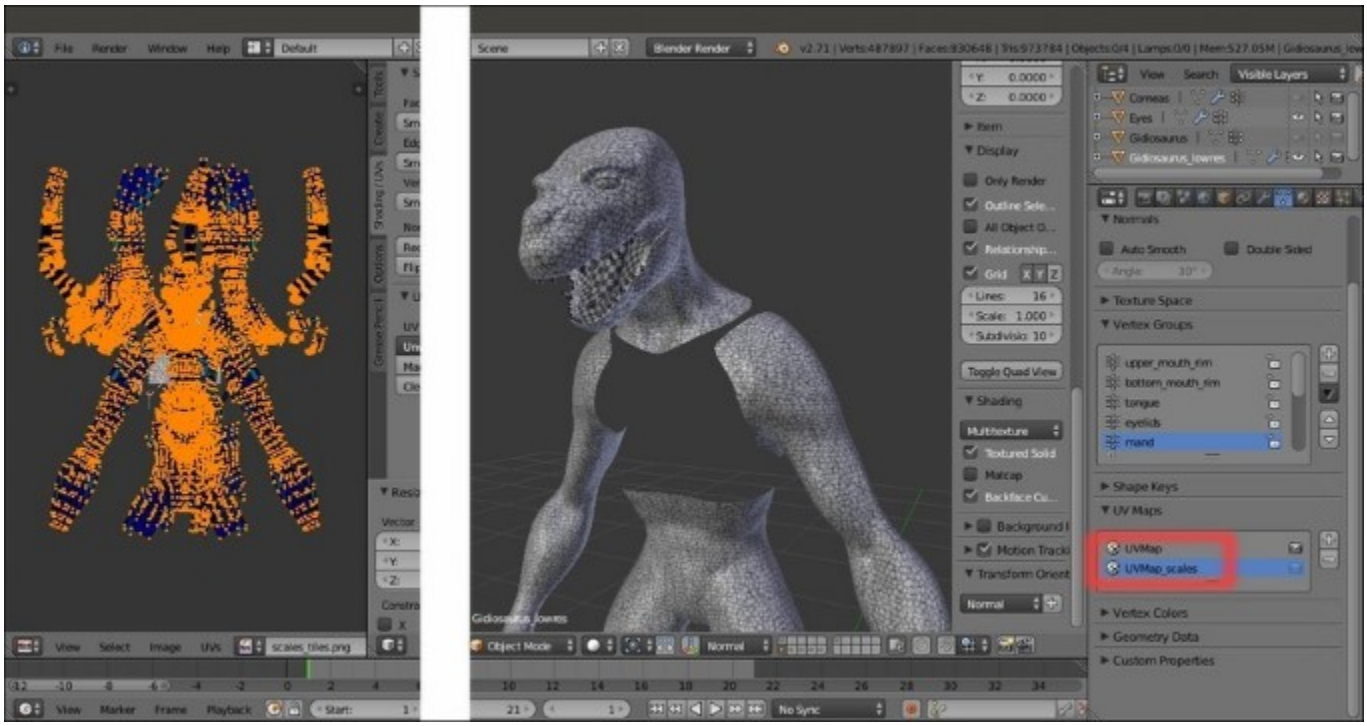
6. Save the file as `4886OS_10_scales_tiles_02.blend`.

How it works…

You might wonder why we didn't use the **Make Seamless** filter of **Gimp** (**Filters** | **Map** | **Make Seamless**) to obtain a tileable image in one click; well, the answer is simple: the **Make Seamless** plugin actually offsets and blends together whole areas of the image, and this can work in several cases, but not for a complex pattern made by scales, where a simple fading is not good enough. In this case, I prefer to paint the joining line between them by hand.

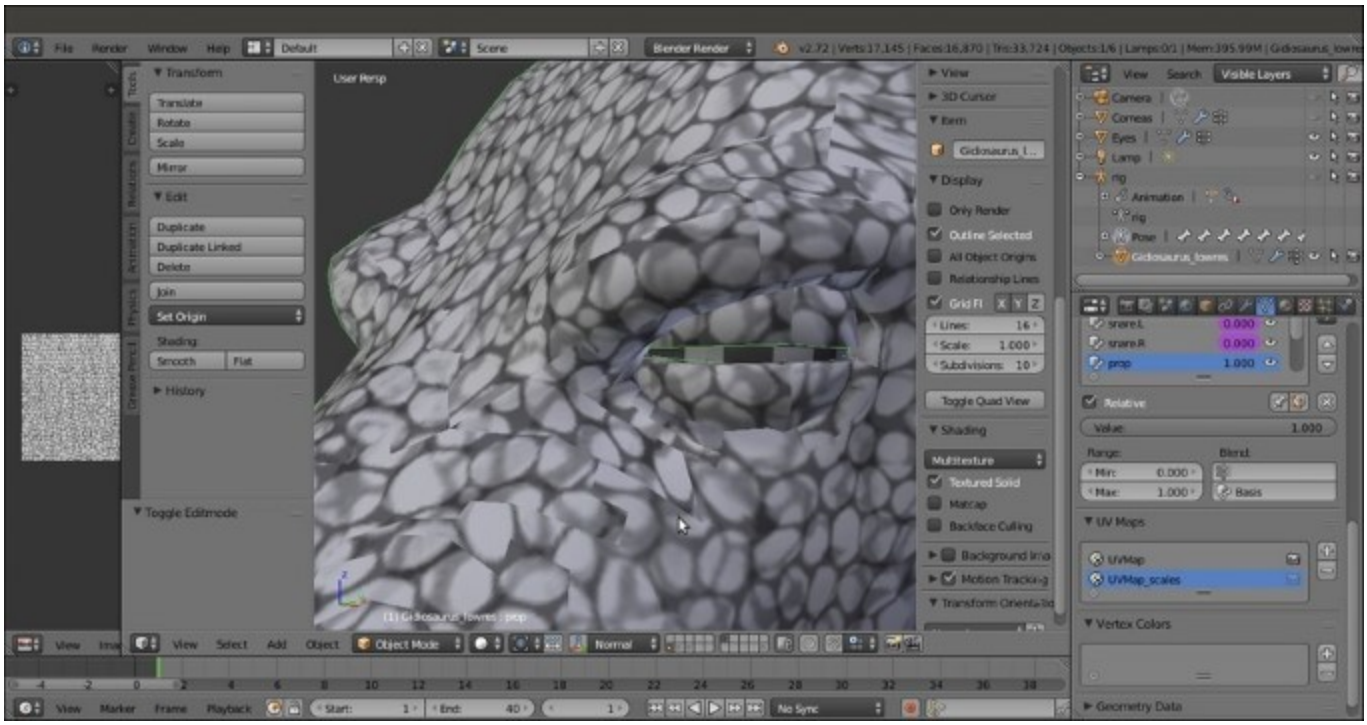
Preparing the model to use the UDIM UV tiles

In the previous recipe, we made the scales image texture seamless, ready to be seamlessly mapped on our model. If you go back to Chapter 5, *Unwrapping the Low Resolution Mesh*, you'll remember that we assigned **two** different sets of **UV coordinate** layers to it: the **UVMap** layer, divided into **5 different tiles** (this is called **UDIM UV Mapping**; it's a popular standard in the industry and means **U-Dimension**), and the **UVMap_scales** layer, set up to repeat the `scales_tiles.png` image pattern at the right size on the model:



The two UV coordinates layers in the UV Maps subpanel

By zooming in and looking carefully at the result of the tiling (in **Textured Solid** mode, which is enabled under the **Shading** subpanel of the *N* viewport sidepanel), you can see that although we used a tileable scales image, we still have seams in some areas. This is obviously due to the fact that the **UVMap_scales** layer (as you can see in **UV/Image Editor** in the **Edit Mode** after selecting all the mesh's vertices) is made up of separated and overlapping islands to obtain a randomly distributed mapping of the scales on the **Gidiosaurus** skin:



The seams on the Gidiosaurus scales skin

A simple solution to fix these seams is to bake the random scales pattern on the **5** tiles of the **UVMap** layer and then use the **Paint Tool** to adjust the gaps. This a step that we have to do in any case to allow further texture modifications such as the painting of befitting facial scales around the **eyebrows**, the **eyes**, the **nostrils**, and so on, but let's go in order.

To be able to bake and then paint on the different tiles in real time through the 3D viewport, we must prepare the file a bit.

Getting ready

Start Blender and open the `Gidiosaurus_library.blend` file; save it as `Gidiosaurus_baking_scales_01.blend`:

1. *Shift*-click on the **13th** scene layer to disable it and hide the **Armor** object, then enable the **6th** scene layer to show **Camera** and **Lamp**. Split the 3D view into two windows and change the left one into a **UV/Image Editor** window (if it shows the **Render Result** image datablock, just click on the **X** icon button to unlink it).
2. Put the mouse in the 3D viewport and press the *T* key to hide the **Tool Shelf** panel, and then maximize the **UV/Image Editor** window as much as possible. Go to the **Outliner** and click on the eye icon to the side of the **Camera** item to disable its visibility in the viewport.
3. Go to the **UV Maps** subpanel under the **Object Data** window and be sure to have the **UVMap** layer selected (the one with the **5** different space tiles); if necessary, click on the camera icon to the right to enable it as the active UV coordinates layer.

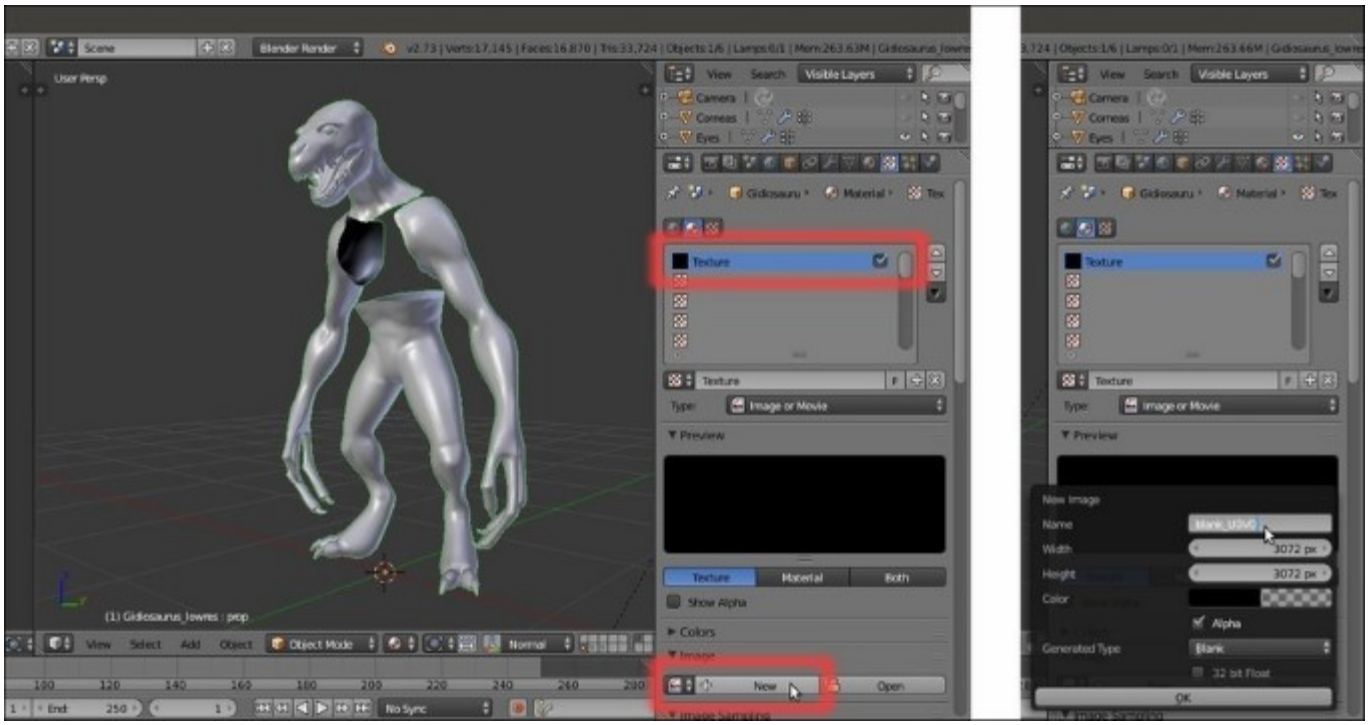
How to do it…

Now, let's prepare the materials; remember that, at the moment, we are under the **Blender Render** engine and not under **Cycles**:

1. Go to the **Material** window and out of **Edit Mode**, click on the **–** icon button to the right of the materials datablock window (*Remove the selected material slot*) to unlink both the `Enamel` and the `Body` material datablocks.
2. Now click on the **+** icon button to add **5** material slots, and then add **5** materials by selecting each slot and clicking on the **New** button.
3. Starting from the top one, rename the **5** materials as `Material_U0V0`, `Material_U1V0`, `Material_U2V0`, `Material_U0V1`, and `Material_U1V1`.

Adding the 5 materials to the Gidiosaurus object

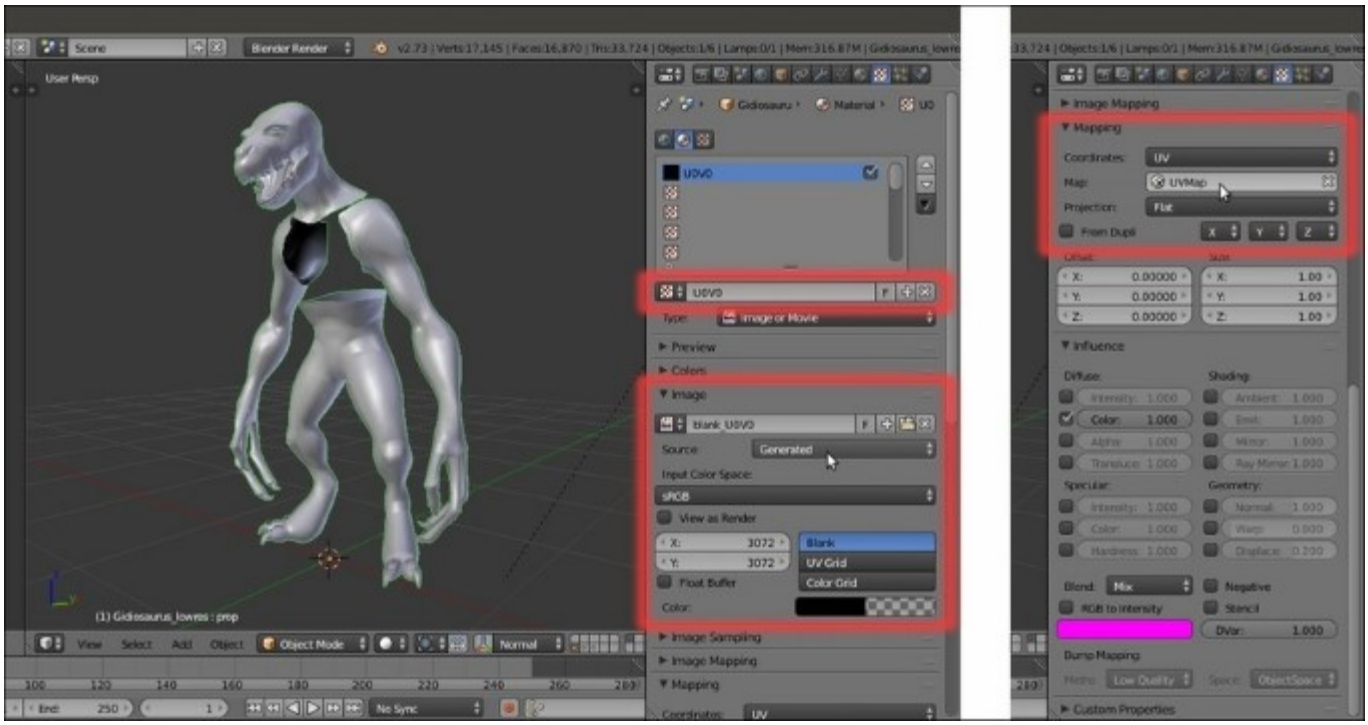
4. Select the `Material U0V0` slot and go to the **Textures** window to click on the **New** button and add a texture.
5. Scroll down the vertical panel by rotating the middle mouse wheel and click on the **New** button under the **Image** subpanel; in the **New Image** pop-up panel, click on the **Color** slot to set the **Alpha** (**A**) value to **0.000**, then *Ctrl* + click on the **Width** slot and right after the default value of **1024**, type `*3`, then press *Enter* (in Blender, you can do a math calculation for any parameter like this anywhere). Copy and paste (*Ctrl* + *C* and *Ctrl* + *V*) the result of the multiplication, **3072**, into the **Height** slot; click on the **Name** slot to write the texture name as `blank_U0V0`, then press the **OK** button at the bottom of the panel:



Adding a blank image texture to the first material

This adds a blank (*alpha background*) **3072** x **3072** pixels image as a texture on the material.

6. *Ctrl* + left-click on the *Unique datablock ID name* slot right above the **Type (Image or Movie)** slot, and rename the default **Texture** name as `U0V0`. Go down to the **Mapping** subpanel and click on the **Map** slot to select the **UVMap** item:



The "Unique datablock ID name" slot, the Image subpanel, and the Mapping subpanel

7. Go to the **UV/Image Editor** to the left side of the screen and click on the double arrows to the side of the **New** button in the toolbar; from the pop-up menu select the **blank_U0V0** item. Slide the toolbar to the right and click on the **Image** item. In the pop-up menu, select the **Save as Image** item (or press the *F3* key) and save the image in the `texture_making` folder as `blank_U0V0.png`, then click on the pin icon button to the right to activate it (*Display current image regardless of object selection*):

The assigned blank image loaded in the UV/Image Editor window and pinned to be displayed regardless of the object selection

As the image is saved under the **Image** subpanel, the **Source** slot caption changes from **Generated** to **Single Image**.

8. Repeat the procedure for all the remaining four materials, assigning and saving a blank image texture for each material. So inside the `texture_making` folder, you have saved the images: `blank_U0V0`, `blank_U1V0`, `blank_U2V0`, `blank_U0V1`, and `blank_U1V1`.
9. Start to split the **UV/Image Editor** window until you have **5 UV/Image Editor** windows. Press the *Tab* key to go into **Edit Mode** with the mesh; put the mouse in the 3D viewport and press the *A* key to select all the mesh's vertices and therefore show the UV islands in all the **UV/Image Editor** windows.
10. Enlarge one **UV/Image Editor** window as much as possible and enable the *Keep UV and edit mode mesh selection in sync* button on the toolbar.
11. If it's the case, deselect everything, then box-select the islands (*B* key then left-click and drag the mouse) in the **U1V0** tile space. In the **Material** window, select the `Material_U1V0` slot and click on the **Assign** button. Go to the top right **UV/Image Editor** window and click on the **X** icon button on the toolbar to unlink the current image datablock (which is still **blank_U0V0**). Then click on the **Image** item on the toolbar and from the drop-down list, select the `blank_U1V0` image.
12. Press the *A* key to deselect everything and box-select the islands in the **U2V0** tile; select the `Material_U2V0` slot and again click on the **Assign** button. Go to the following image editor and unlink the current image datablock to load the `blank_U2V0` image.

13. Repeat for the other two missing tiles and material slots (note that this is not necessary for the **U0V0** ones, which are, by default, first assigned to the whole mesh and the first created material and so still remain associated to `Material_U0V0`). Then go out of **Edit Mode**.

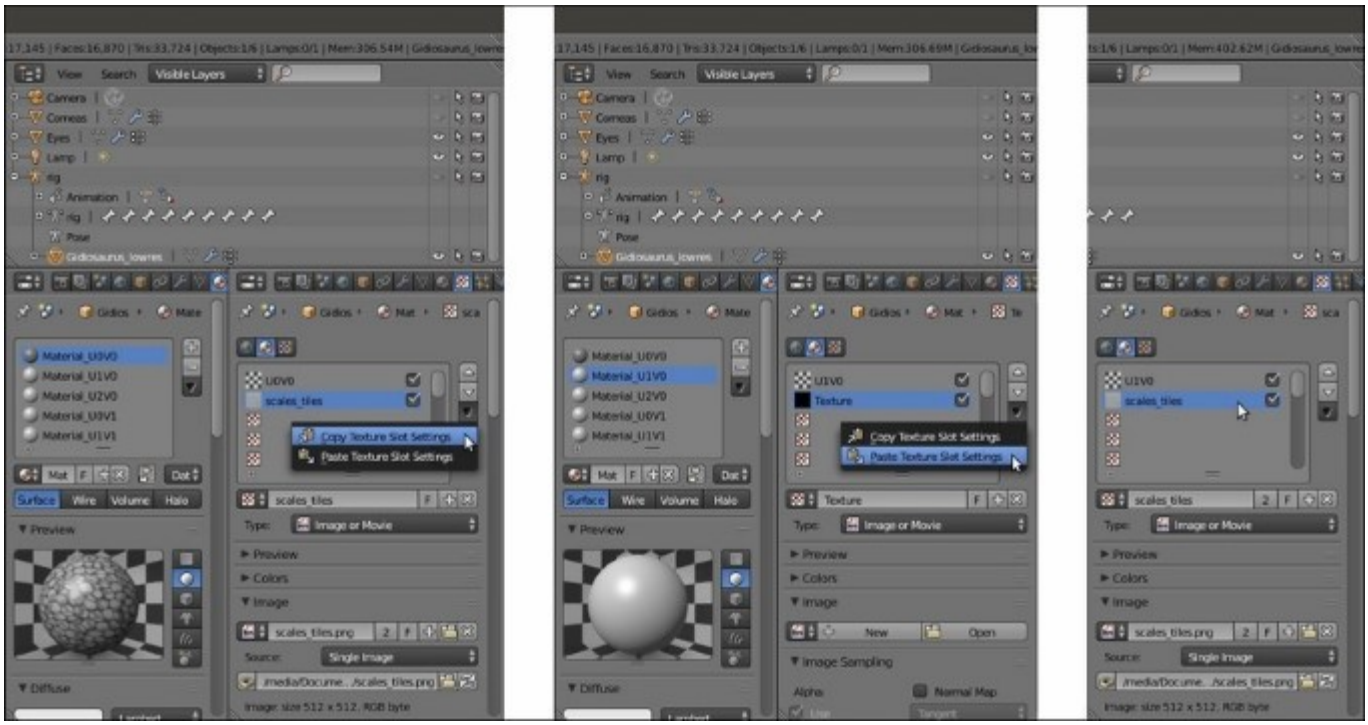
The work-space prepared with the 5 UV/Image Editor windows with their respective blank images

As you can see, by selecting the vertices of the UV islands in **UV/Image Editor**, the corresponding vertices on the mesh are also selected. Moreover, this makes all the UV islands visible in the image editor, even though, we haven't selected a single vertex on the mesh yet (normally, you see only the islands of the selected vertices in the image editor). This way, it's simple to associate a certain UV island with a certain material and a certain group of vertices on the mesh.

14. Go to the **Material** window and select the `Material_U0V0` slot. Go to the **Texture** window and click on the second texture slot right under the **U0V0** one. Click on the **New** button, scroll down to the **Image** subpanel, and click on the **Open** button to browse to the `texture_making` folder and load the `scales_tiles.png` image.
15. Go to the **Mapping** subpanel and in the **Map** slot, select the **UVMap_scales** UV coordinates layer. Rename the `Unique datablock ID name` slot as `scales_tiles`. Click on the checkbox to the side of the **U0V0** texture slot to disable it (this is just temporary but mandatory for the baking, otherwise it would create a dependency loop, that is, the *Circular reference in texture stack* message in the top main header and in the **Terminal** panel as well):

Disabling the blank texture image and loading the "scales_tiles.png" image in the first material

16. Click on the button with a black arrow pointing downward, right after the + and –icon buttons, and from the pop-up menu, select the **Copy Texture Slot Setting** item. Select the `Material_U1V0` slot and then click on the second texture slot right under the **U1V0** one and click on the **New** button. Click again on the black arrow button and this time, select **Paste Texture Slot Setting**:



Copying and pasting the "scales_tiles" texture slot to the other materials

17. Repeat this copy and paste for the other three materials, and also remember to disable the first texture slot for all the materials.
18. Press *Tab* to go out of the **Edit Mode** and save the file.

How it works…

Thanks to the pin icon button that is enabled for each loaded image, it's possible to keep the different images visible at the same time. At this moment, the **5** different PNG images are blank, so this isn't particularly evident; it will be a lot more clear when we start to actually paint on the model through the 3D viewport.

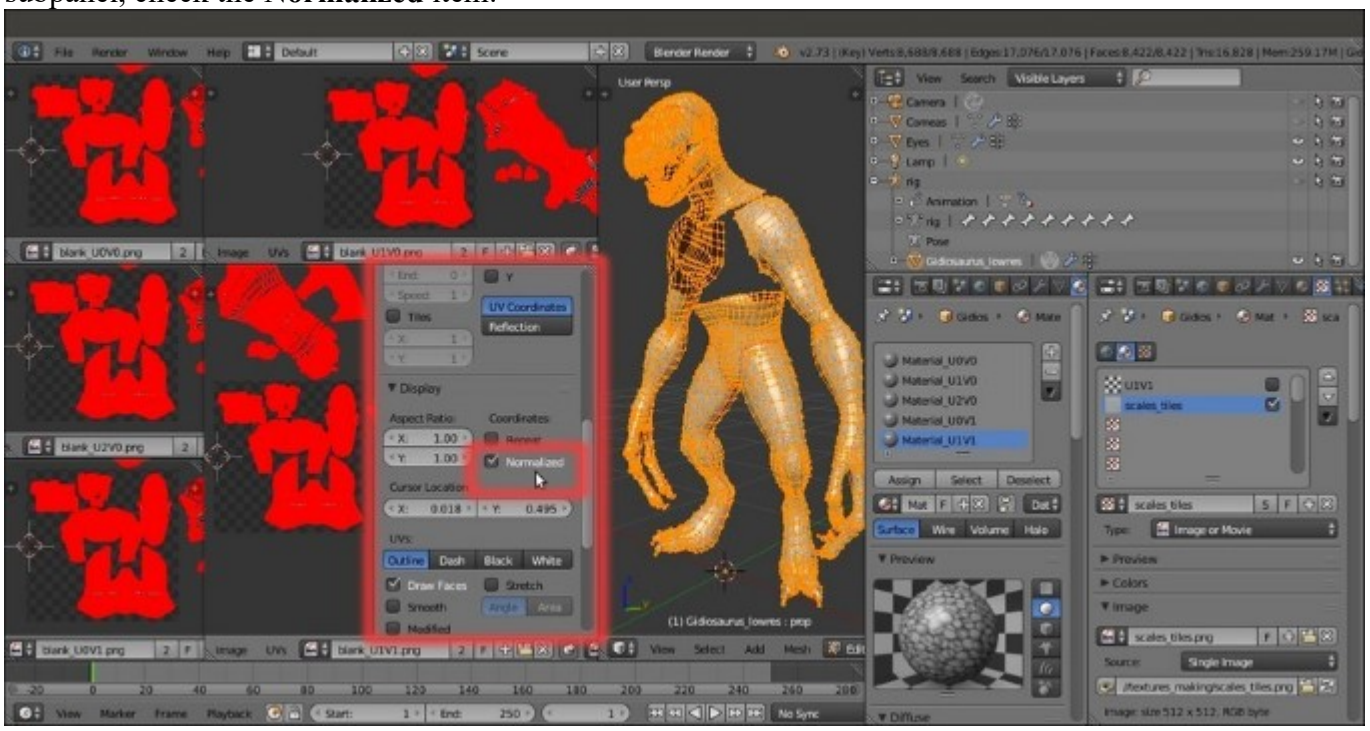
Baking the tileable scales texture into the UV tiles

What we have to do now is to bake the `scales_tiles.png` image map (used in all the materials and mapped on the **UVMap_scales** coordinates layer) on the **5** tiles of the **UVMap** coordinates layer.

Getting ready

At this moment, Blender is not able to bake automatically outside of the default **U0V0** tile space yet, so a bit of additional work is needed; nothing particularly difficult by the way. The steps are as follows:

1. Press *Tab* to go into **Edit Mode** again and then put the mouse in the **blank_U0V0 UV/Image Editor** window; press the *N* key to call the **Properties** sidepanel and under the **Display** subpanel, check the **Normalized** item:



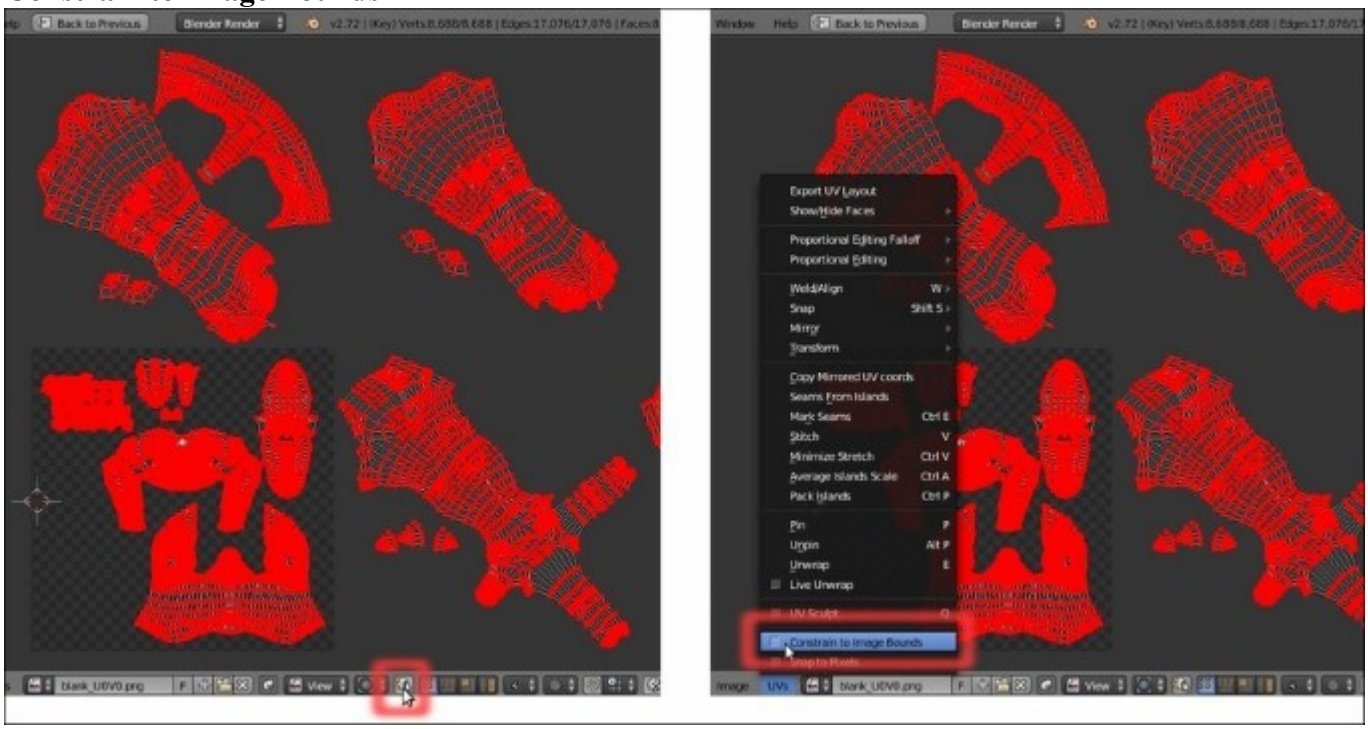
The Normalized item in the Display subpanel under the N Properties sidepanel of the UV/Image Editor window

2. Press *N* again to hide the **Properties** sidepanel. Go to the **UV Maps** subpanel under the **Object Data** window and click on the + icon button to the right to add a new UV coordinates layer (**UVMap.001**), then rename it **UVMap_temp** (or whatever you prefer).

How to do it…

We are now going to create a new UV coordinates layer for the baking by moving all the islands in the outside tiles to the space of the default one; but before we go on, we must be sure about two things:

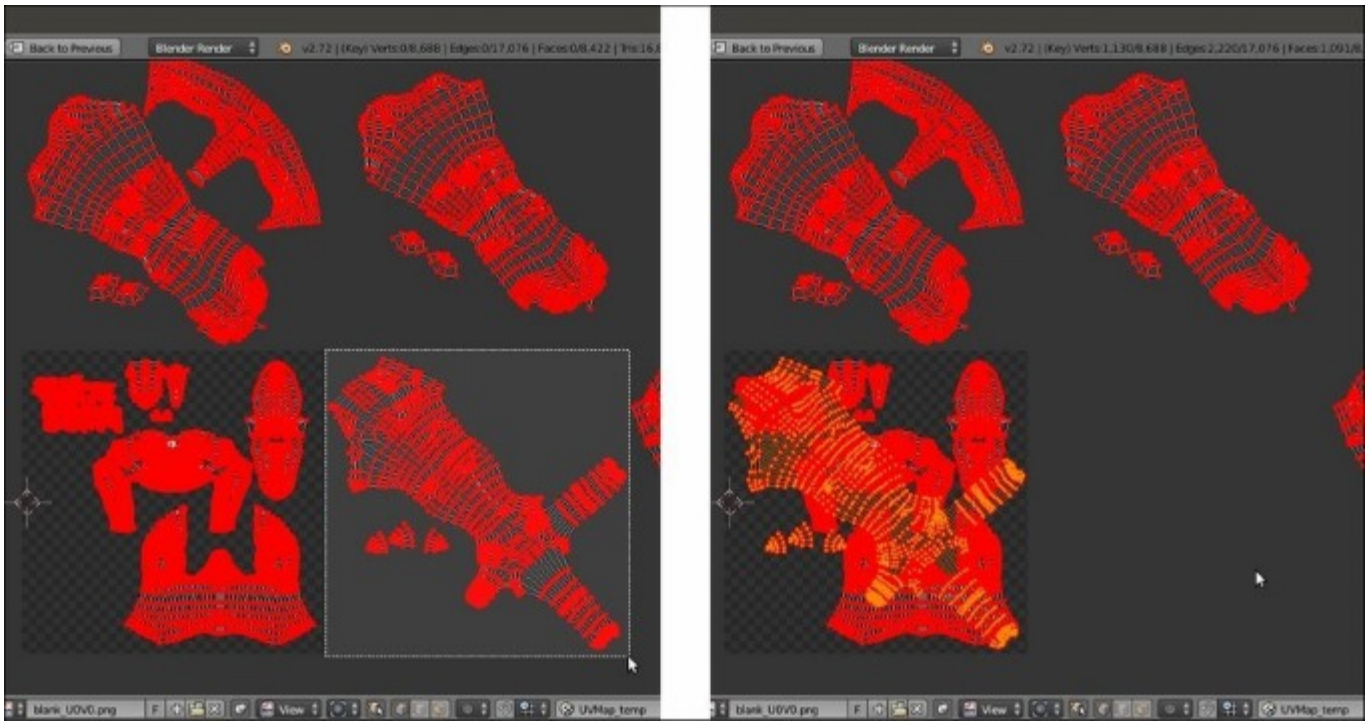
- In the toolbar of the **blank_U0V0** image editor window, the *Keep UV and edit mode mesh selection in sync* button must now be disabled
- In the pop-up menu, accessible by clicking on the **UVs** item in the image editor toolbar, the **Constrain to Image Bounds** item must be deselected:



The "Keep UV and edit mode mesh selection in sync" button and the Constrain to Image Bounds item

Go to the **blank_U0V0** image editor window; if you prefer, maximize it (mouse cursor into the window and press *Ctrl* + Up Arrow). If necessary, press *A* to deselect all the islands.

1. Now, box-select the islands on the **U1V0** tile, and move them to the default **U0V0** tile space (*G* | *X* | **-1** | *Enter*):



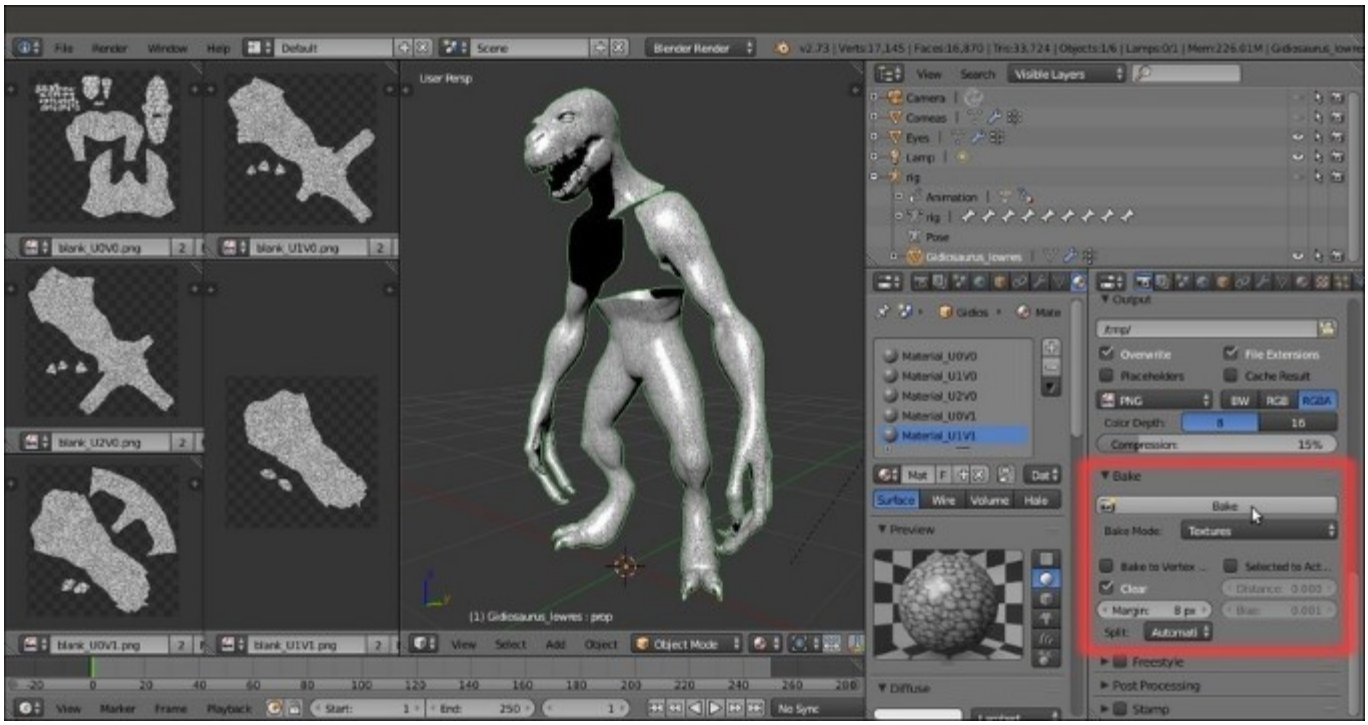
The UV islands of the U1V0 tile space, box-selected and moved to the default U0V0 tile space

2. Deselect everything and box-select the islands at **U2V0**, then move them to the default space, which is the same as the previous one (*G* | *X* | **-2** | *Enter*).
3. Repeat for the last **2** islands tiles (*G* | *Y* | **-1** | *Enter*) and (*G* | *X* | **-1** | *Enter* and then *G* | *Y* | **-1** | *Enter*), then rearrange the image editor windows.
4. Go to the **Object Data** window and in the **UV Maps** subpanel, be sure to have the **UVMap_temp** layer, the **last one**, enabled as the active one, that is, the **camera icon** to the right side of the **UVMap_temp** item must be the one enabled and visible (*Set the map active for rendering*):

The new UVMap_temp coordinates layer

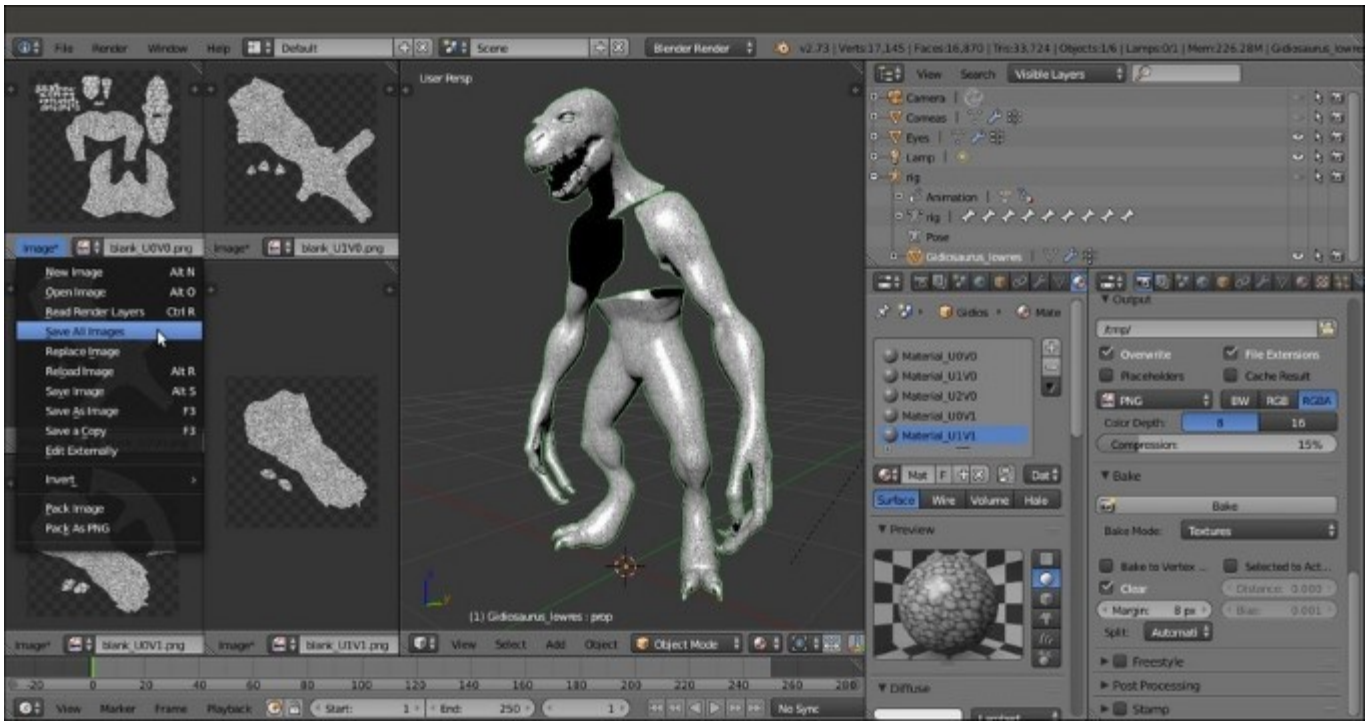
5. Out of **Edit Mode**, go to the **Render** window and then go to the **Bake** subpanel (usually at the bottom of the panel). If necessary, click on the **Bake Mode** slot to select **Textures**, then set the **Margin** value to **8** or higher; and check the **Clear** item flag. Be sure to have the **Gidiosaurus** object still selected and press the **Bake** button.

   After a while, the baked scales textures appear on the **5** PNG images, baked according to the **UV** islands of the **5** tiles of the **UVMap** layer:



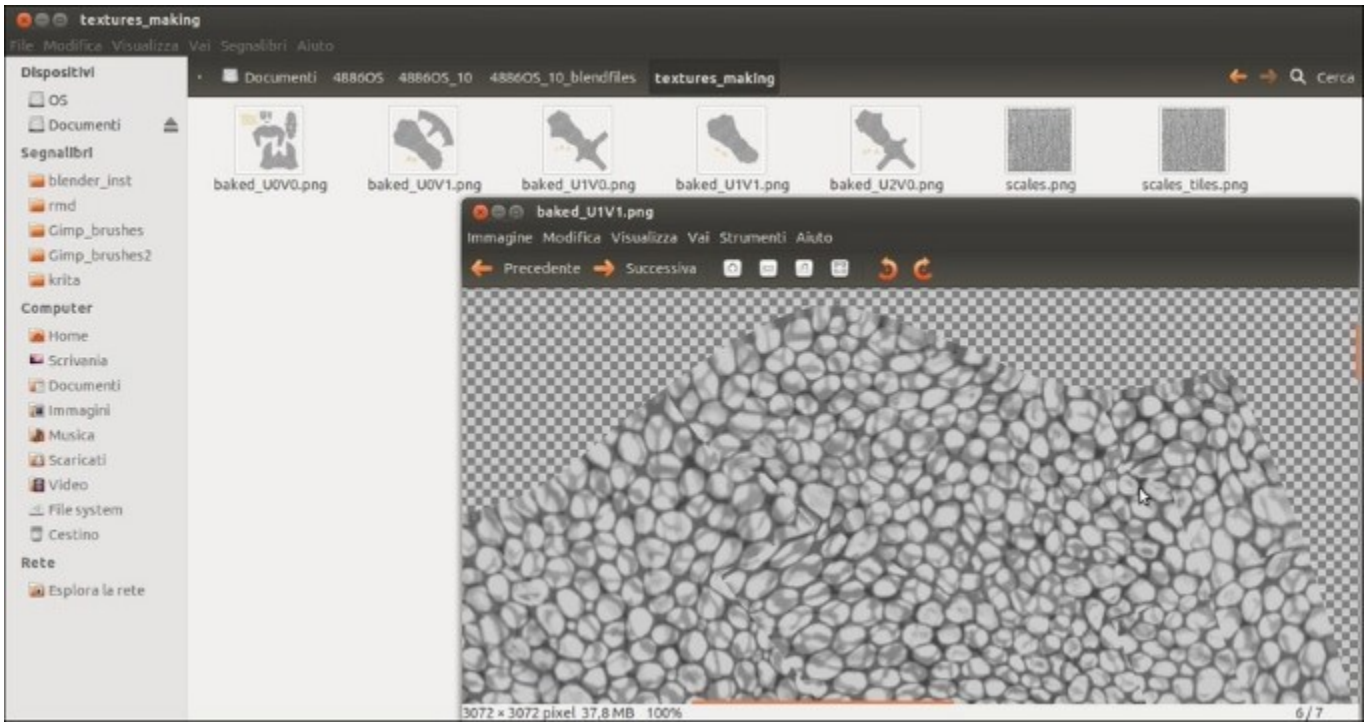
The 5 baked images and the Bake subpanel under the Render window

6. Click on the **Image** item on the **UV/Image Editor** toolbar and from the pop-up menu, select the **Save All Images** item or if you want to preserve your blank images (we are going to use them again later), just save each image at a time (**Save As Image** item or *F3* key) with the names `baked_U0V0.png`, `baked_U1V0.png`, and so on:



Saving the baked image maps

Opening the `texture_making` folder on your desktop, you will now find the baked textures:



The baked textures saved inside the "texture_making" folder

As you can see in the information bar at the bottom of the **GNOME** image viewer (I'm working in **Linux Ubuntu**), each image saved from Blender is **37.8** megabytes.

The large size of the images can of course be reduced (a lot) by opening them in **Gimp** (or any other 2D application) and re-saving.

How it works…

All the UV islands have been moved to the default **U0V0** tile space, which is the only one where the baking happens, but because each image is associated with a different part of the mesh, each image is correctly baked with the right islands and textures.

In fact, inside Blender, and in our case, the location of each tile in the UV space doesn't actually matter; we made a new UV coordinates layer and kept the old one just in case the model should be exported to a different 3D application.

To move the islands exactly by the correct number of pixels, we enabled the **Normalized** item in the **Display** subpanel of the image editor *N* sidepanel to display the UV coordinates from **0.0** to **1.0**, rather than in pixels. Anyway, without the **Normalized** item enabled, it would have been enough to move the islands by **3072** pixels, that is, the width (or/and height) in pixels of the assigned blank image.

There's more…

As with any other software, Blender is not free from bugs; particularly, the baking section seems to have an annoying bug, which is very difficult to fix because it happens very rarely and randomly, so that it cannot easily be reproduced and consequently submitted to the Blender bug tracker (https://developer.blender.org/maniphest/project/2/type/Bug/).

It's difficult to understand the reason for this, but sometimes the software refuses to do the baking, claiming that *No objects or images (are) found to bake to* (the message appears on the top right main header and in the **Terminal** panel as well); in our case, this seems to happen when you switch the *active for rendering* UV coordinates layers.

If this happens, one thing you can do is check that all the images assigned in the **UV/Image Editor** windows to the different materials under one UV layer, also appear correctly assigned under the other UV layer (it shouldn't make a difference, but who knows), eventually re-assigning them one at a time.

If the baking still fails, there is a simple workaround; switch to the **UVMap** coordinates layer instead, rather than the **UVMap_temp** one, and just move the islands to the default **U0V0** space and bake them *one at a time*. To do this, first bake the islands of the **U0V0** tile space and save the image, then move the islands of the **U1V0** tile space to the **U0V0** tile space, bake and save as a different image, and so on with the islands of all the tiles.

Painting to fix the seams and to modify the baked scales image maps

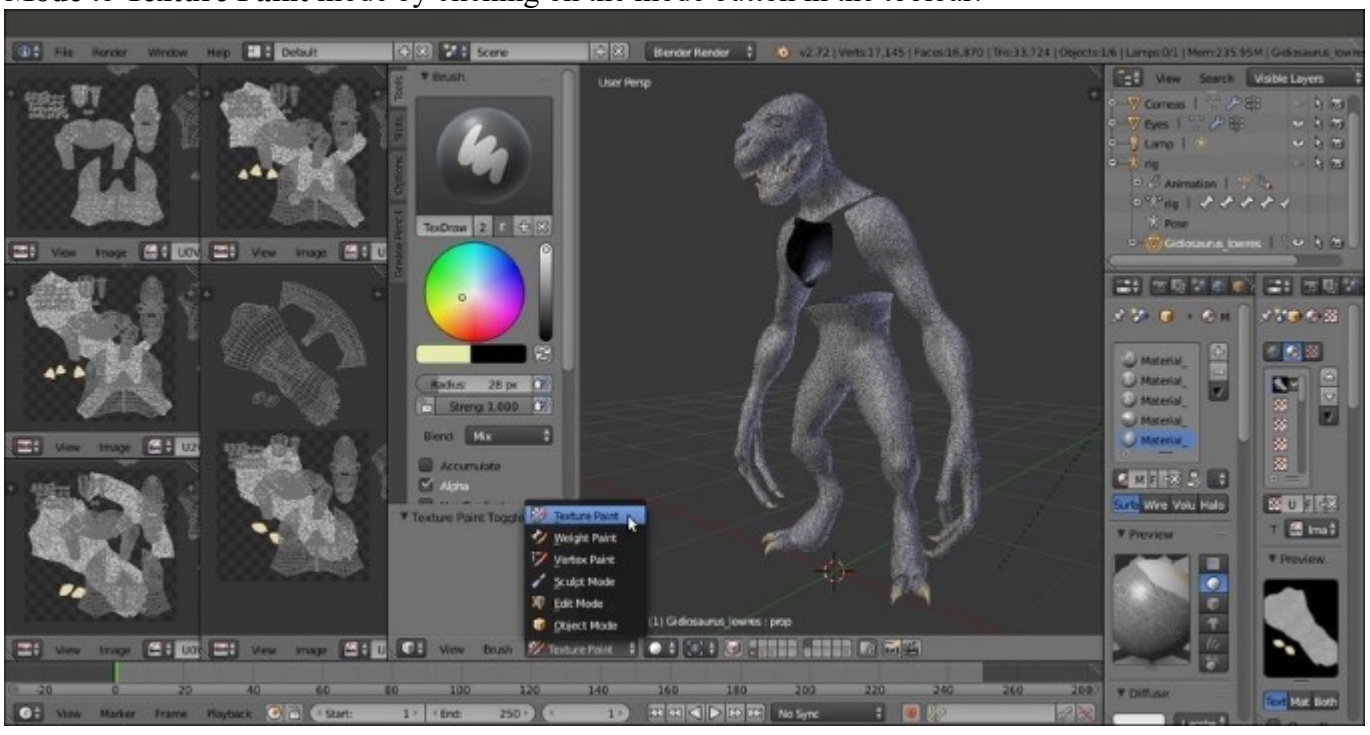
In the previous recipe, we baked the *randomly* tiled scales image map on the **5** tiles of the **UVMap** coordinates layer. This was necessary for the next step to be able to fix seams and modify certain areas of the baked scales images through the **Paint Tool**.

In order to paint in real time on both the model and on all the images assigned to the **5** different UV tiles, and at the same time, once again we need to first prepare the file. To be more precise, we must assign **5** different materials to the mesh, one for each tile and each one with the appropriate image texture.

Getting ready

Start Blender and re-open the `Gidiosaurus_baking_scales_01.blend` file; save the file as `Gidiosaurus_baking_scales_02.blend`.

1. Minimize the image editor windows on the left as much as possible, then also minimize the **Outliner**, the **Material**, and the **Texture** windows on the right to make room for the 3D viewport.
2. Click on the **Viewport** Shading button in the 3D viewport toolbar and switch the shading mode from **Material** to **Solid**, then press the *T* key to call the **Tool Shelf**. Then switch from **Object Mode** to **Texture Paint** mode by clicking on the mode button in the toolbar:



Switching to Texture Paint mode

3. Click on the **Options** tab inside the **Tool Shelf** and under the **Project Paint** subpanel, enable the **Occlude**, **Null**, and **Normal** items:

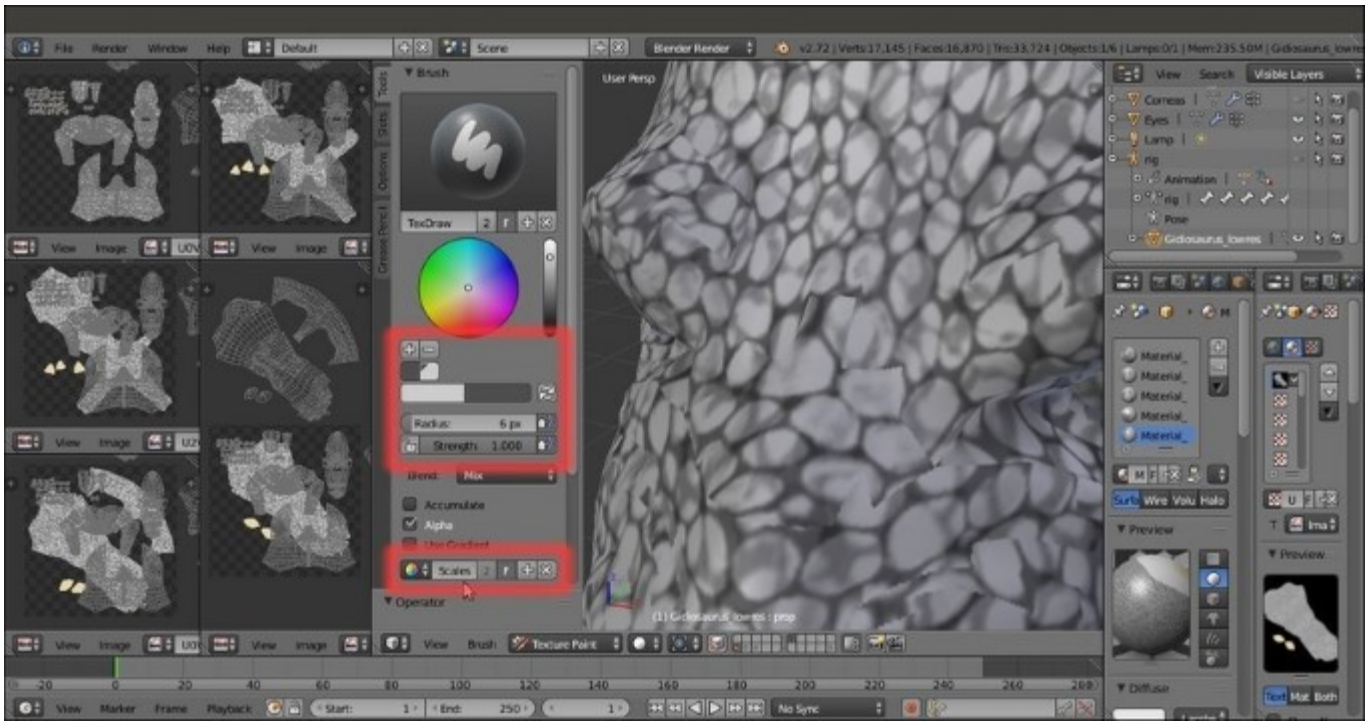
Items to be enabled under the Options tab

4. Click on the **Tools** tab inside the **Tool Shelf** to go back to the **Brush** subpanel options.

How to do it…

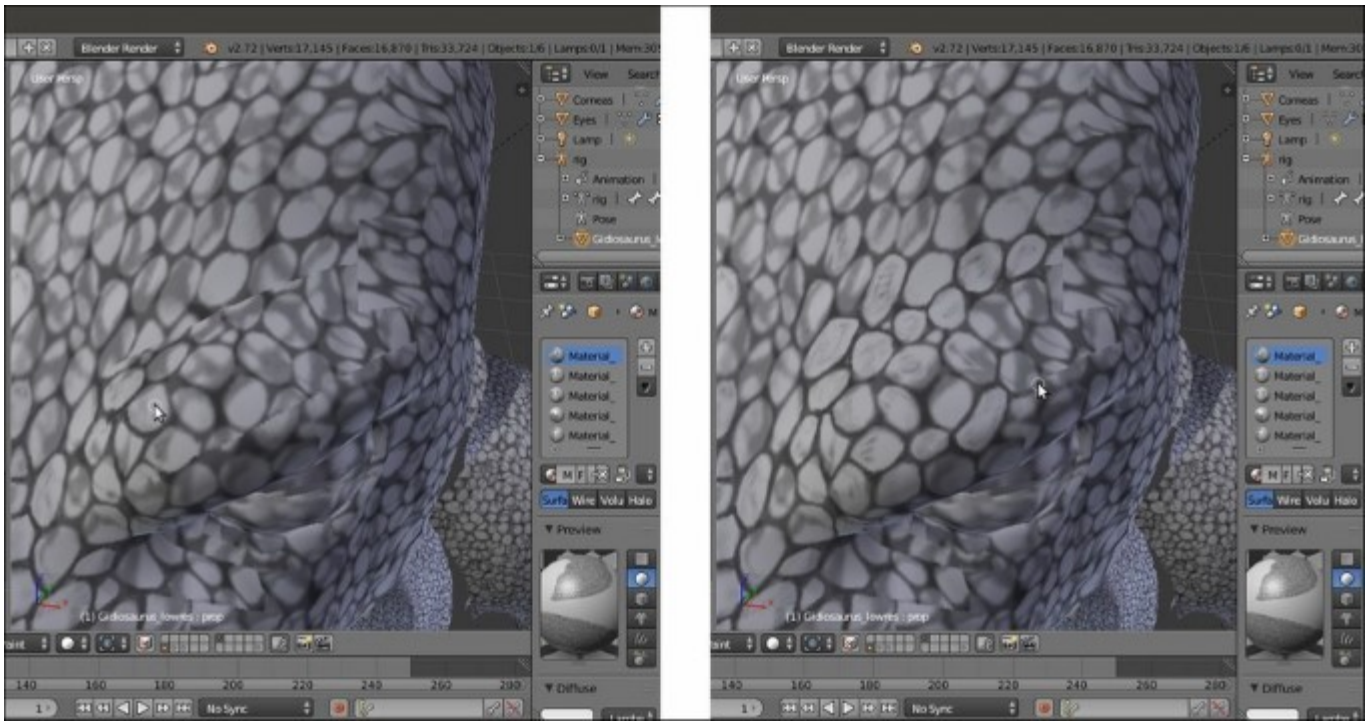
At this point, we are ready to start to paint both directly on the model in the 3D viewport or also in the **UV/Image Editor** windows (just for all eventualities, I suggest you make a copy of the baked scales images before starting to paint):

1. Zoom in on a part of the **Gidiosaurus** object in the 3D viewport, for example, the **head**.
2. Put the **mouse cursor** on a bright value of the scales image on the model and press the *S* key to sample it, then go near the color selector in the **Brush** subpanel under the **Tool Shelf** to the left and click on the *Toggle foreground and background brush colors* button (the one with the two opposing arrows) to switch the active color. Otherwise, simply press the *X* key, put the mouse cursor on a dark area, and press *S* again to sample it as the opposite color.
3. Scroll down and click on the **New** button (*Add new palette*) at the bottom of the **Brush** subpanel; + and – icon buttons will have appeared above the color switcher. Click on the + icon button to add the active color to the palette, then switch the colors and click on the + button again to add a new color to the palette.
4. Set the brush's **Radius** value to **6** and **Strength** value to **1.000**, and if you are using a **graphic tablet**, be sure to have the **2** *tablet pressure sensitivity* buttons at the sides of the previous items enabled. Change the default **Palette** name in **Scales**.



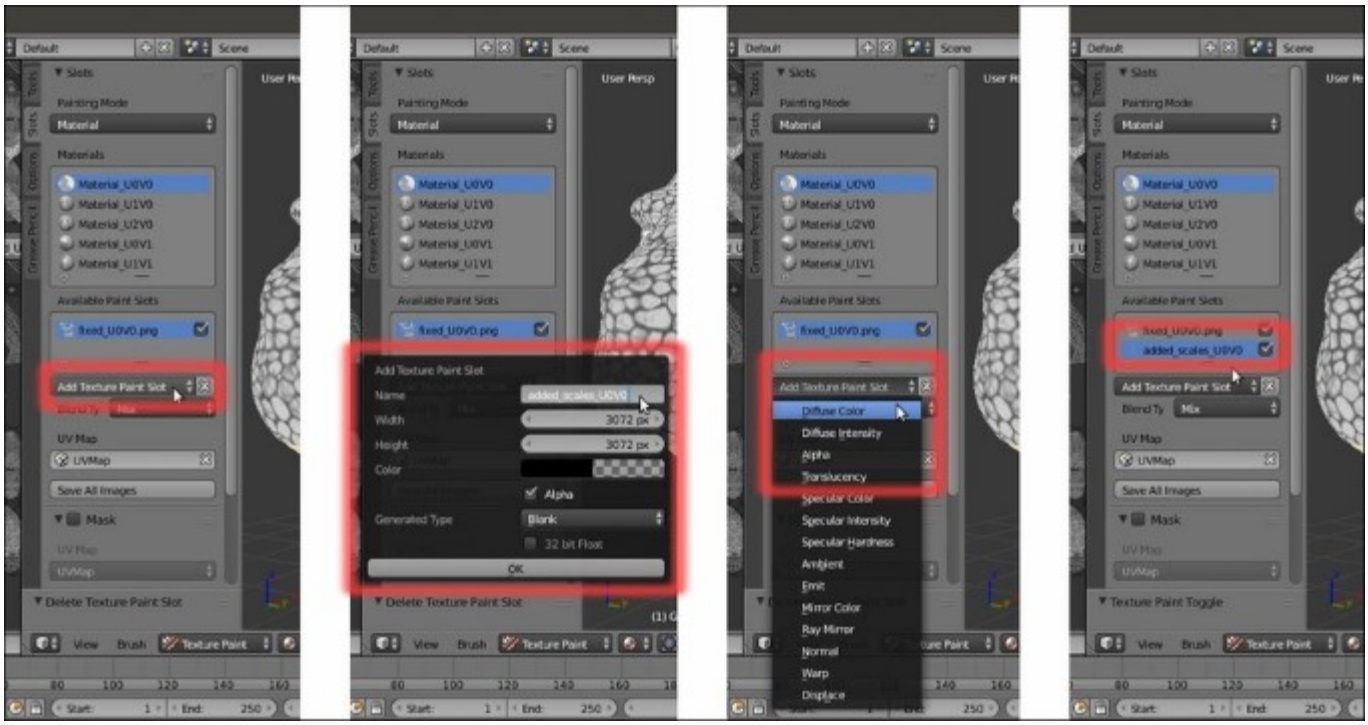
Setting a palette and the brush strength and radius

5. Simply start to paint on the model, re-drawing the scales where there are seams by flipping the color as you need to, by pressing the *X* key and painting the dark folds and the light scales. The two colors we sampled, used with the pressure sensitivity enabled, should be enough, but feel free to sample new ones and add it to the palette as you go on:



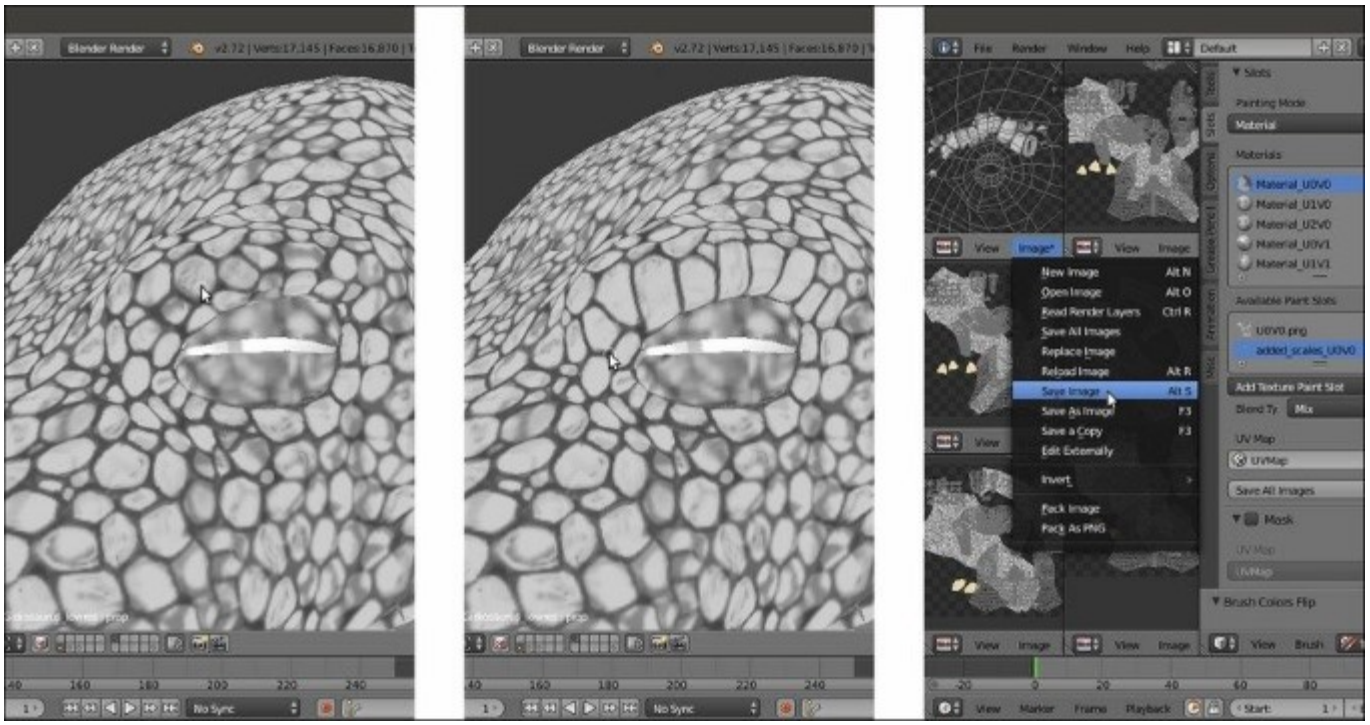
Painting on the model to fix the image texture seams

6. From time to time, click on the **Slots** tab in the **Tool Shelf** and click on the **Save All Images** button.
7. Do most of the fixing you can, across the entire **Gidiosaurus** body, keeping in mind that it's quite useless to spend time fixing seams in areas that will later be covered by the **Armor** (for example, the top of the head).
8. When you are done, *be sure to have saved all the edited images* as explained in step 6 (but you can also do it one image at a time through the **Image** | **Save Image** item in each editor window toolbar or by pressing the *Alt* + *S* shortcut).
9. Now, be sure to have the `Material_U0V0` slot selected as active in the **Material** window and go to the **Texture** window; left-click on the empty slot right under the **U0V0** one and then click on the **New** button to add a new image texture.
10. Scroll down to the **Image** subpanel and click on the **New** button. In the **New Image** pop-up panel, write `added_scales_U0V0` in the **Name** slot, then set the **Width** and **Height** values to **3072** and the **Alpha** (**A**) value to **0.000** (basically add a new blank and background transparent image as shown in step 5 of the *How to do it…* section of the *Preparing the model to use the UDIM UV tiles* recipe):



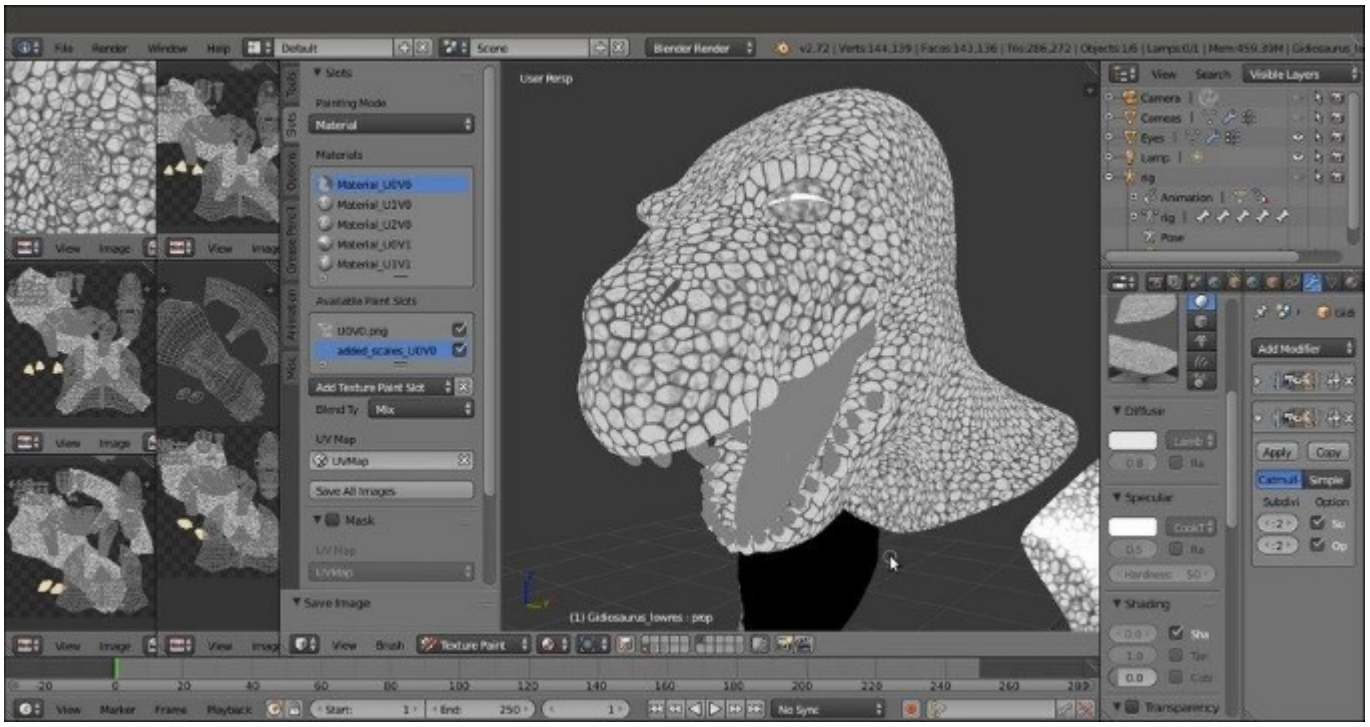
Adding a new texture paint slot layer

11. Go to the **Shading** subpanel of the **Material** window and enable the **Shadeless** item for the `Material_U0V0` slot. Then go to the 3D viewport toolbar and change **Viewport Shading** from **Solid** to **Material**.
12. If not selected already, click on the **Slots** tab in the **Tool Shelf** panel and select the `added_scales_U0V0` item that appears under the `U0V0.png` once inside the **Available Paint Slots** window.
13. Directly in the 3D view, start to paint new scales on the **eyebrows** to replace the randomly distributed ones; use the light color of the palette to conceal the old scales on the first layer, and the dark color to draw the new ones. Try to build a consistent pattern, also using photos of real reptiles as references. When you are done, save the image in the `texture_making` folder:



Painting new scales on the eyebrow

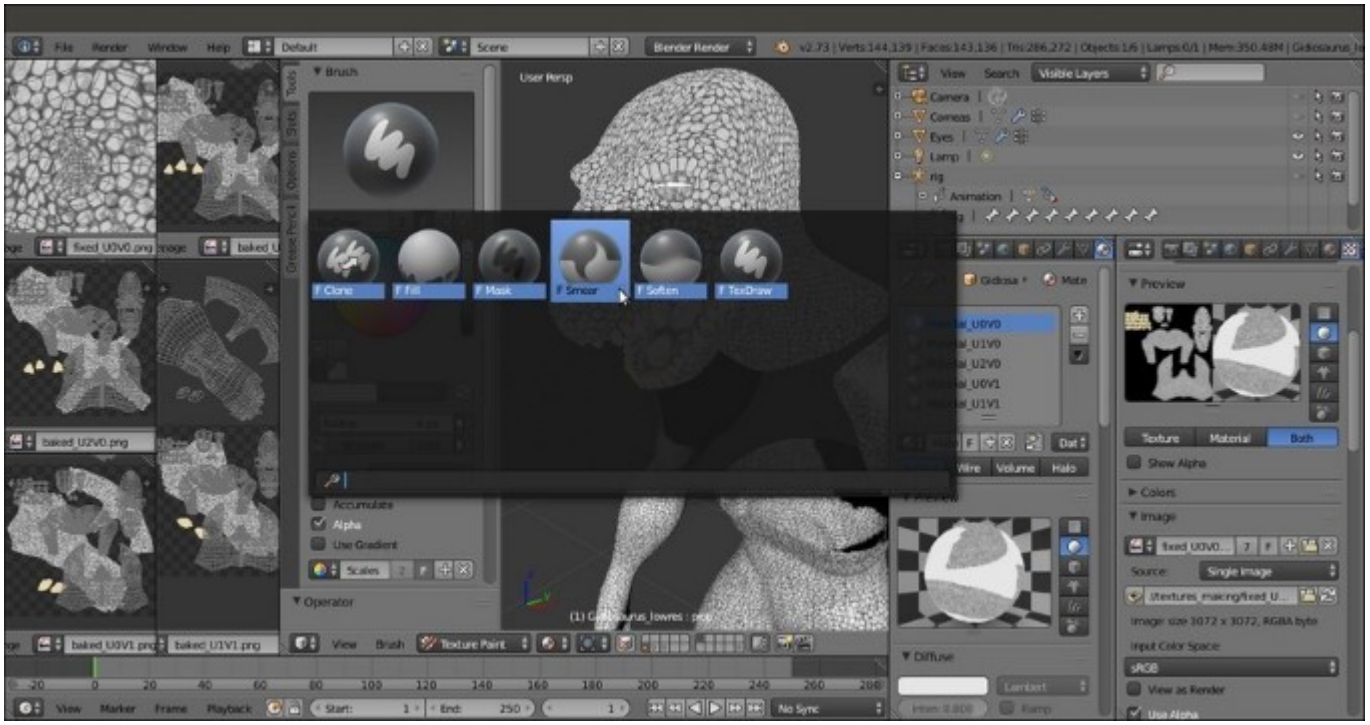
14. Draw new scales around the **nostrils** and the **rim** of the **mouth**:



Painting new scales also around the nostrils and at the rim of the mouth

The Blender **Paint Tool** also has other handy brushes; a particularly useful one is the **Smear** brush, which smudges the borders or any blotch in the scales.

To access the brushes, just click on the big window in the **Brush** subpanel under the **Tool Shelf** and click on the chosen one to select it:



The brushes selection pop-up menu

Remember to always save the painted images before closing Blender, otherwise you'll lose them.

Also remember that if you have more than one texture layer to save, it's necessary to load each one of them into an **UV/Image Editor** window. This is actually very quick and easy, just select each layer in the **Available Paint Slots** window (in the **Slots** subpanel under the **Slots** tab) to make it appear in the image editor window and save it through the **Image** | **Save As Image** menu in the editor toolbar.

Once saved the first time, it's possible to re-save all of them in one single click, through the **Save All Images** items, both in the tab, as well as in the toolbar menu.

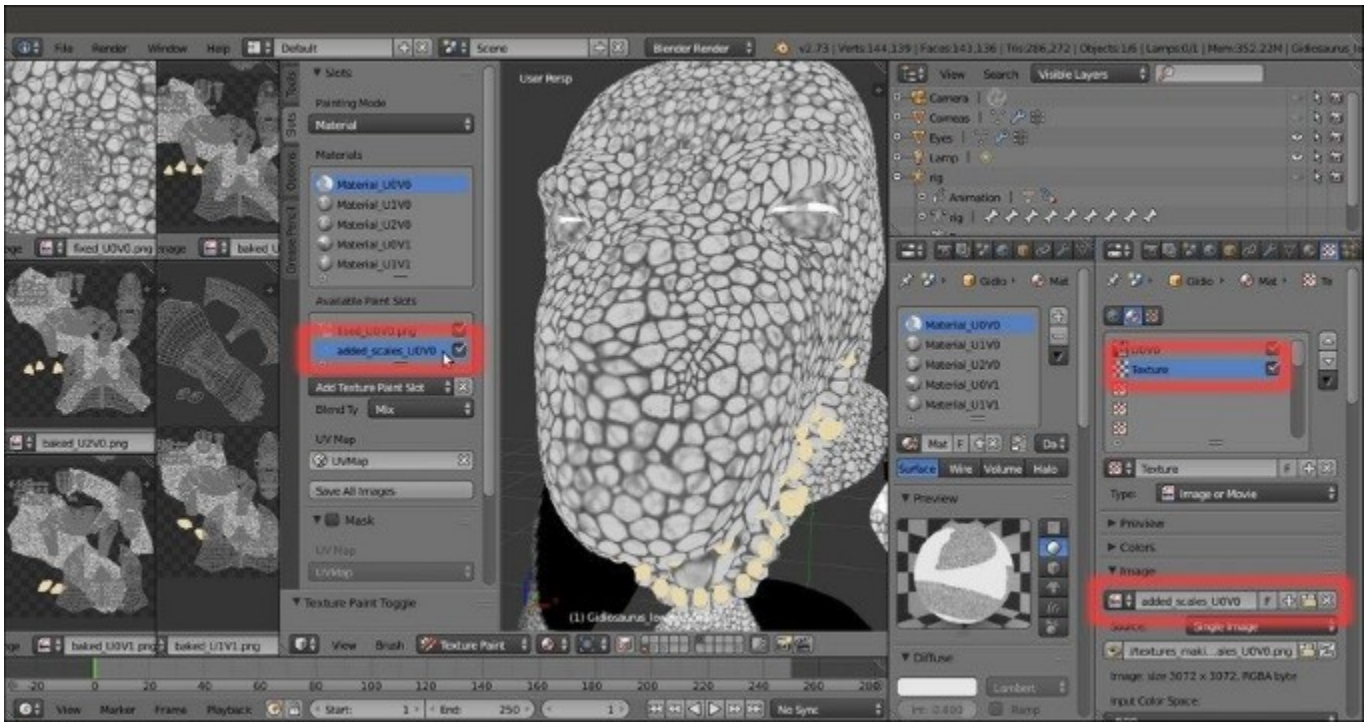
15. Save the file as `Gidiosaurus_painting_scales_fix.blend`.

Note

In the textures and blend files provided with this cookbook, you'll find textures fixed only in the **head** area; I leave the task of finishing the fixing and drawing of new scales on the rest of the body (for example, bigger scales can be added to the upper side of the hand **fingers**, **feet**, **shoulders**, and so on) to you.

How it works…

The new Blender **2.73** texture paint layering feature works simply by adding a new texture slot to the material, and automatically setting it as required, by the type of texture you selected in the **Add Texture Paint Slot**; in fact, by going to the **Texture** window, it is possible to see the added new texture slot and also, if necessary, to change the settings:



The added texture paint slot also appearing as a texture slot in the Texture window

Nonetheless, it is a great addition to Blender that can simplify the texture painting workflow a lot.

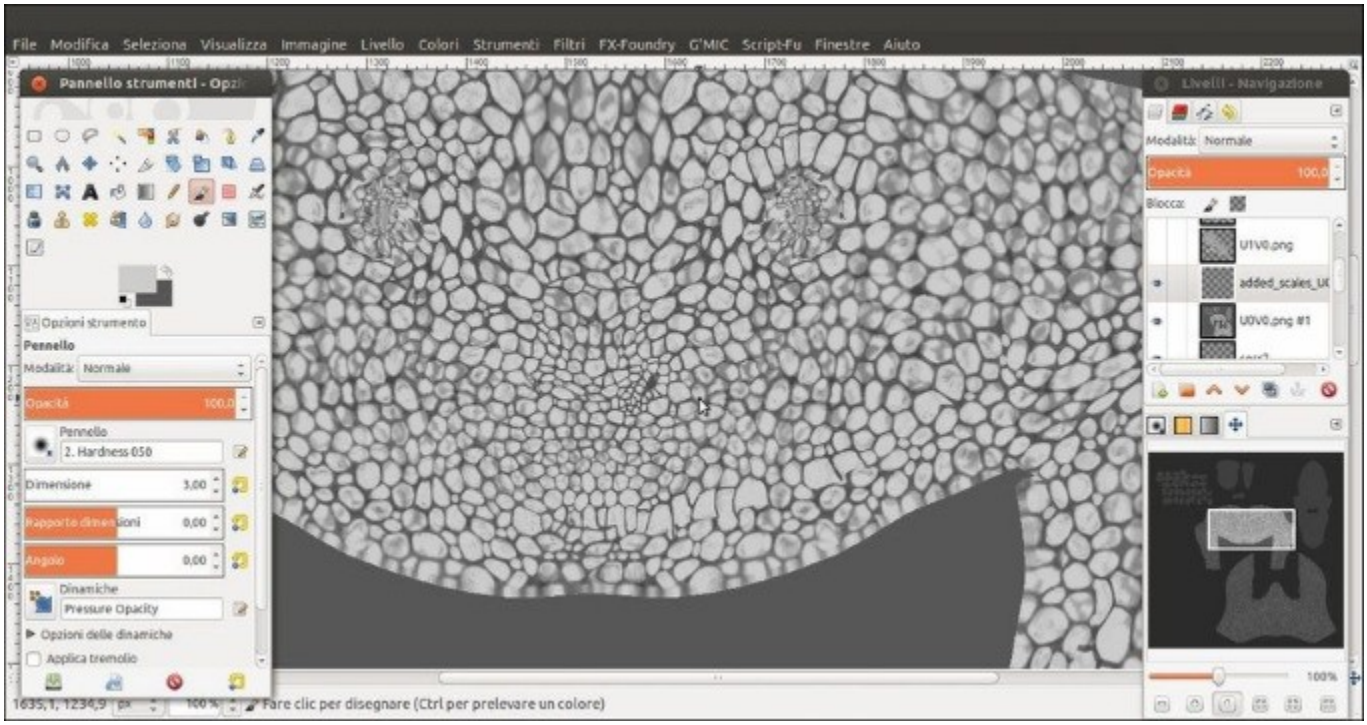
There's more…

To bake the added scales as a single image with the background scales images, perform the following steps:

1. Enter **Edit Mode** and click on the **Select** button for `Material_U0V0` to select the vertices assigned to that tile.

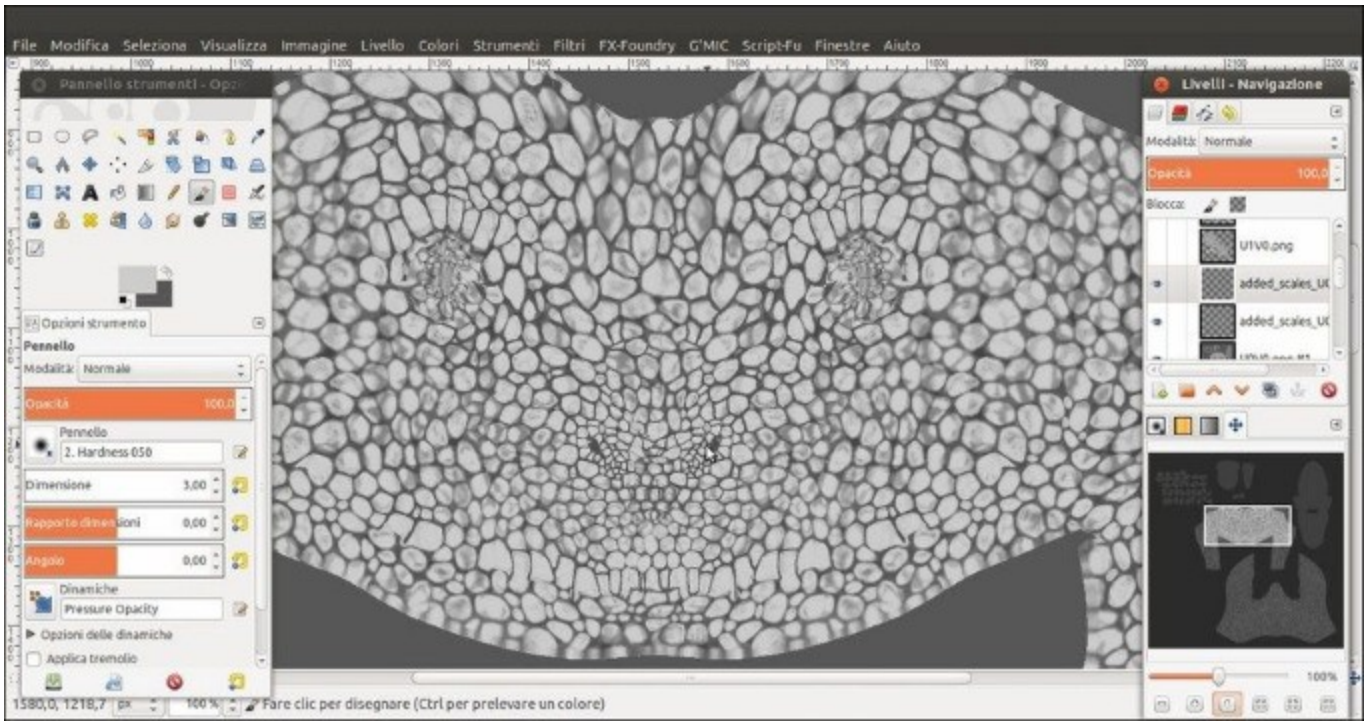
2. Go to the top left **UV/Image Editor** window and press *Alt* + *N* to call the **New Image** pop-up menu; add a new blank image of **3072** x **3072** pixels named `baked_scales_U0V0`, and save it inside the `textures_making` folder.
3. Go out of **Edit Mode** and if it's the case, click on the double arrows icon to the left side of the image name datablock to re-assign the just-created `Untitled` image.
4. Repeat for the other four materials, naming the new images according to the tile and saving them inside the `texture_making` folder as well.
5. Go to the **UV Maps** subpanel under the **Object Data** window to make the **UVMap_temp** coordinates layer active.
6. Go to the **Render** window, and be sure that the **Bake Mode** under the **Bake** subpanel is set to **Textures**, then click on the **Bake** button.
7. After the baking is done, click on the **Image** item in the toolbar of one of the image editors and select the **Save All Images** item.

Not necessarily everything has to be fixed by painting in Blender; for example, it would be enough to fix the scales on only the half of the **head**, export the painted image texture, and open it in **Gimp** (or any other 2D image editing software):



The scales "U0V0.png" image map and the "added_scales_U0V0.png" layer in Gimp

Then, by duplicating the layer and mirroring it, plus a little bit of painting to adjust the seams, it's really simple to obtain the missing half of the new scales texture:



The duplicated and mirrored "added_scales_U0V0.png" layer in Gimp

Of course, if you want to fix every side and part by hand-painting on the model in Blender to obtain a more natural looking result, no one is going to stop you!

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Textures/Painting
- http://wiki.blender.org/index.php/Dev:Ref/Release_Notes/2.72/Painting
- http://docs.gimp.org/en/gimp-tutorial-quickie-flip.html

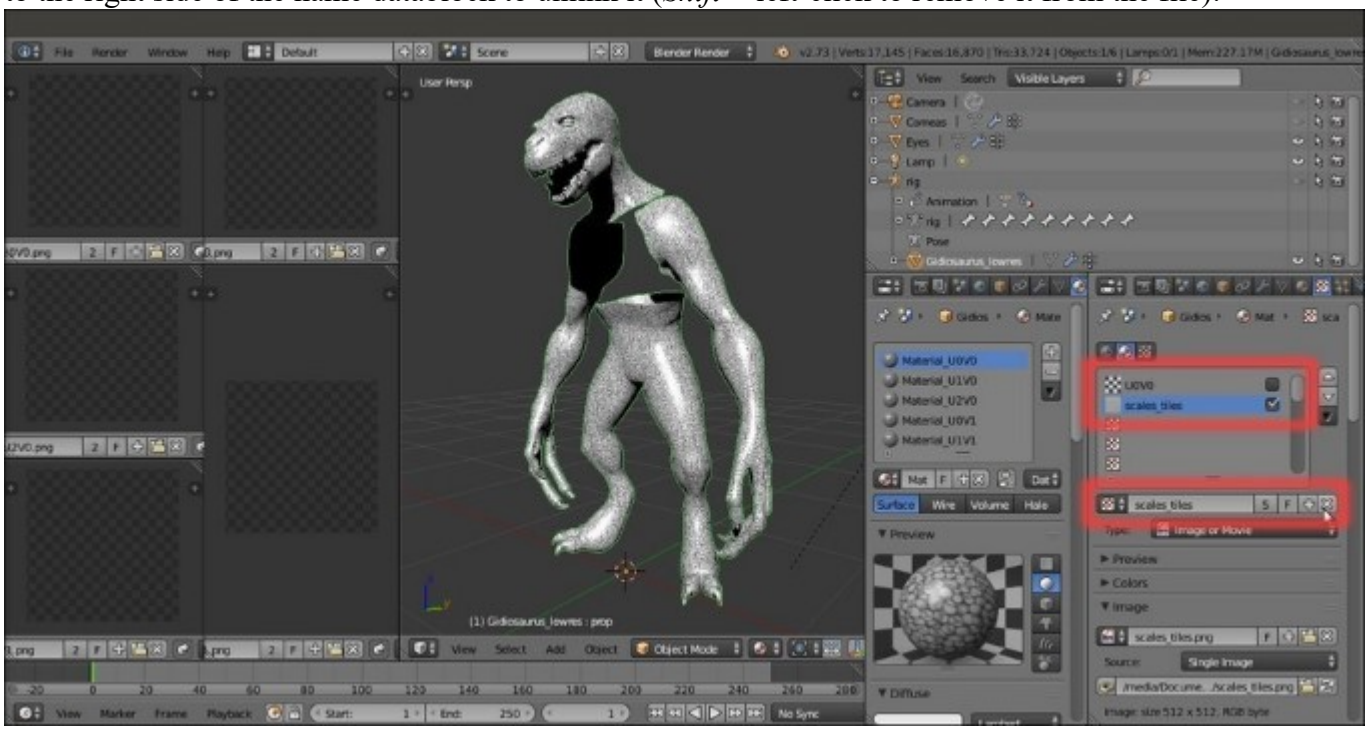
Painting the color maps in Blender Internal

After having obtained the scales textures, we must now paint the diffuse color of the **Gidiosaurus** character.

Getting ready

Start Blender and open the `Gidiosaurus_baking_scales.blend` file:

1. Go to the main **Properties** panel and be sure to have the **UVMap** coordinates layer selected and active, in the **UV Maps** subpanel under the **Object Data** window.
2. Go to the **Material** window and select the `Material_U0V0` slot, then go to the **Texture** window and be sure to have the **scales_tiles** texture slot selected; left-click on the **X** icon button to the right side of the name datablock to unlink it (*Shift* + left-click to remove it from the file):



Unlinking the scales_tiles texture slot datablock

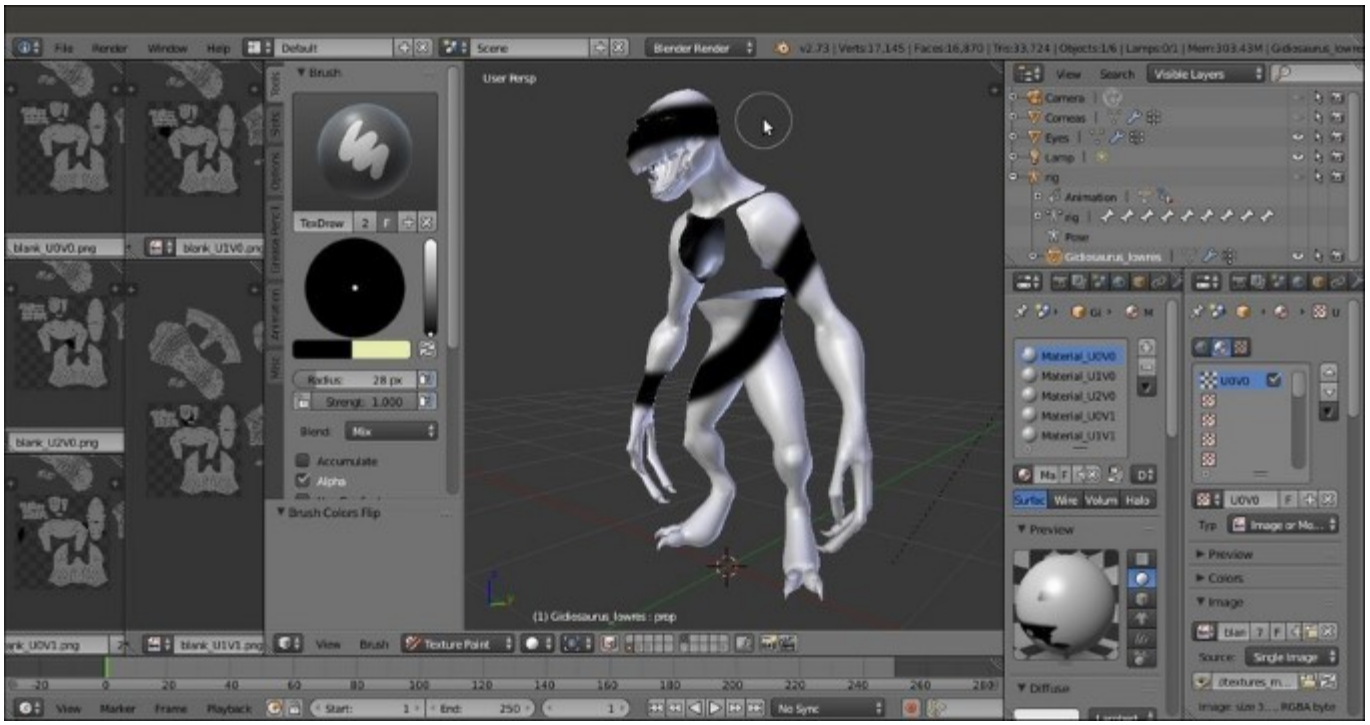
3. Select and enable (by clicking on the checkbox to the right) the **U0V0** texture slot. Repeat the procedure at steps 2 and 3 for all **5** materials.

   I'll take for granted that you have preserved your blank images and that they are the ones loaded into the current file; otherwise, substitute them with new blank images (you have to do this both in the **Texture** window as well as in the **UV/Image Editor** windows) by following steps 5 and 7 of the *Preparing the model to use the UDIM UV tiles* recipe in this chapter.
4. Minimize the image editor windows to the left and the **Material** and **Texture** panels to the right as much as possible, then click on the mode button (*Sets the object interaction mode*) on the

toolbar to go into **Texture Paint** mode. Press *T* with the mouse pointer over the 3D view to call the **Tool Shelf** and click on the **Viewport Shading** button on the toolbar to switch to **Solid** mode:

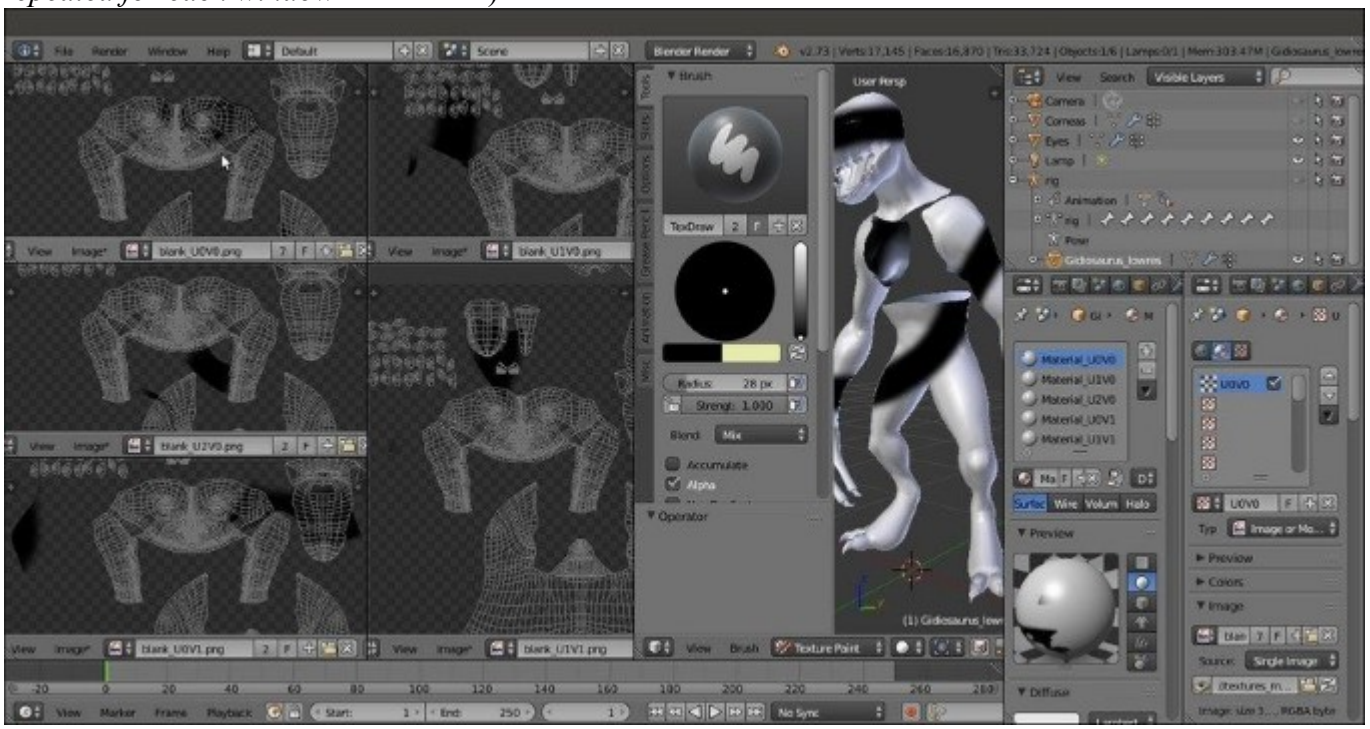
Switching to Solid viewport shading mode

5. Just to verify that everything works correctly, select a black color (or any other one) in the color wheel under the **Brush** subpanel and trace a continuous stroke in the 3D viewport that envelopes all the **Gidiosaurus** body parts:



Testing that everything works correctly with a single stroke on the mesh

By enlarging the **UV/Image Editor** windows, you will see that after the stroke, each image has been updated with the corresponding painting (pay no attention to the over-imposed and *repeated for each window* UV islands):



The test stroke correctly visible inside each one of the UV/Image Editor windows

6. Rearrange the image editor windows, then press *Ctrl* + *Z* to undo the stroke and save the file as `Gidiosaurus_painting_BI.blend`.

How to do it…

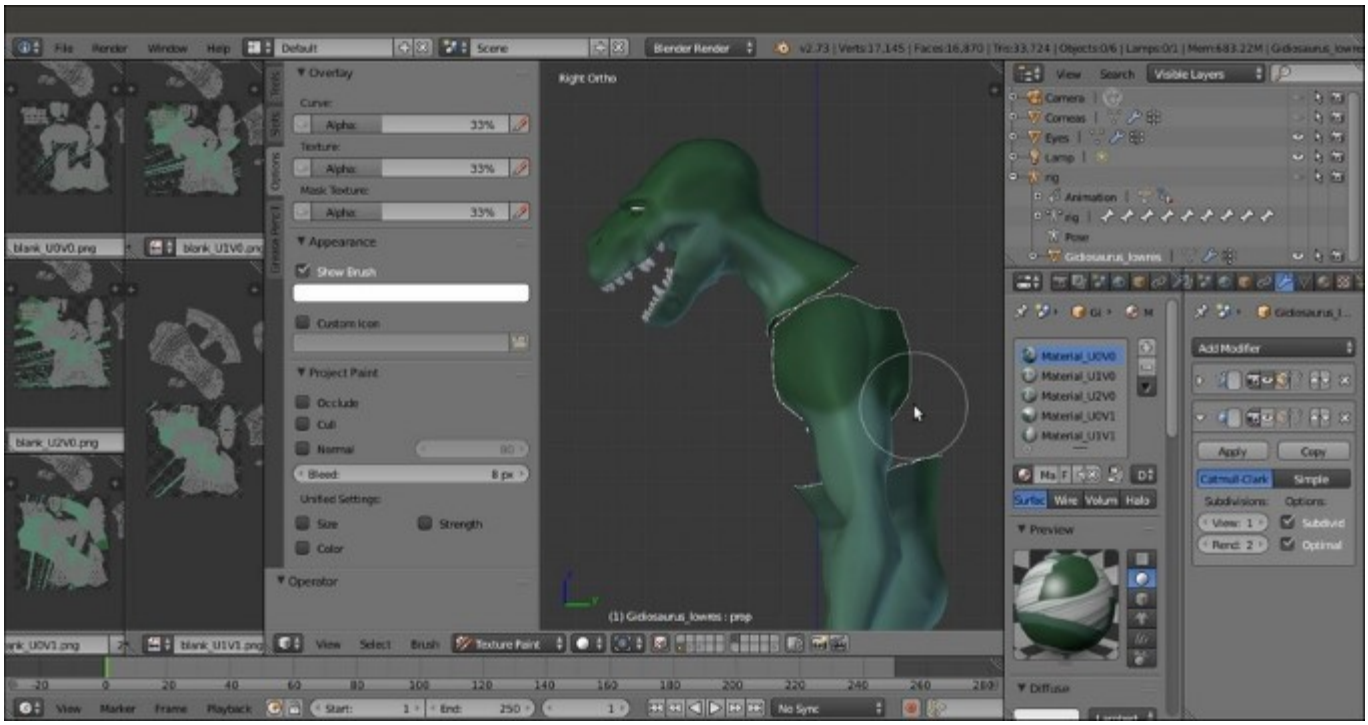
We are now ready to paint the basic color for the **Gidiosaurus** character. But first, one more little thing:

1. Select the **Gidiosaurus** object and enter **Edit Mode**; select the vertices of all the **teeth** and all the **talons**, then assign a new vertex group renamed **enamel**; press *Ctrl* + *I* to invert the selection and go out of **Edit Mode**.
2. Now, start by selecting a medium dark greenish color (**R 0.349**, **G 0.510**, **B 0.435**) in the color wheel under the **Brush** subpanel. Scroll down and go to the bottom of the subpanel and click on the **New** button to create a new palette, then click on the **+** icon button (**Add Swatch**) above the **Foreground Color** slot (the left one) to add the color to the palette. Rename the default **Palette** name as `Gidiosaurus_colors`.
3. In the 3D viewport toolbar, click on the *Face selection masking for painting* button to enable the masking tool; now it's possible to paint only on the part of the mesh that has selected vertices in **Edit Mode**, so in this case we want to paint only on the skin, leaving the teeth and the talons blank.
4. Click on the **Brush** window and select the **Fill** brush; if necessary, click on the greenish color box added to the palette (called **Swatch**) to load it as the **foreground color** (note that with the **Fill** brush, the background color swatch disappears and the foreground color swatch becomes the only one available). Set the **Strength** value to **1.000** and click on the **Gidiosaurus** object in the 3D viewport.

   After a while, all the *paintable* parts of the mesh are filled with the active color (and therefore also the textures in the **UV/Image Editor** windows; there are weird straight lines, probably a bug, but not a problem in this case because they don't show on the mesh and we can fill in the texture's backgrounds later anyway). If any tiny part is left out, just click on one of the parts again to fill it:

The Fill brush and the Mask button in the 3D view toolbar

5. Now select the **TexDraw** brush and a darker and more saturated green color (**R 0.129**, **G 0.275**, **B 0.125**) as the foreground color, and add it to the palette.
6. Under the **Tool Shelf**, go to the **Options** tab and be sure to have the **Occlude**, **Cull** and **Normal** items disabled still; then go into the **Ortho Side** view.
7. Now it's time to use a tablet, if you have one; enable both the tablet pressure sensitivity buttons to the side of the **Radius** and **Strength** items and start to shade the **Gidiosaurus** body on the **head**, **shoulders**, **arms**, and **legs**:

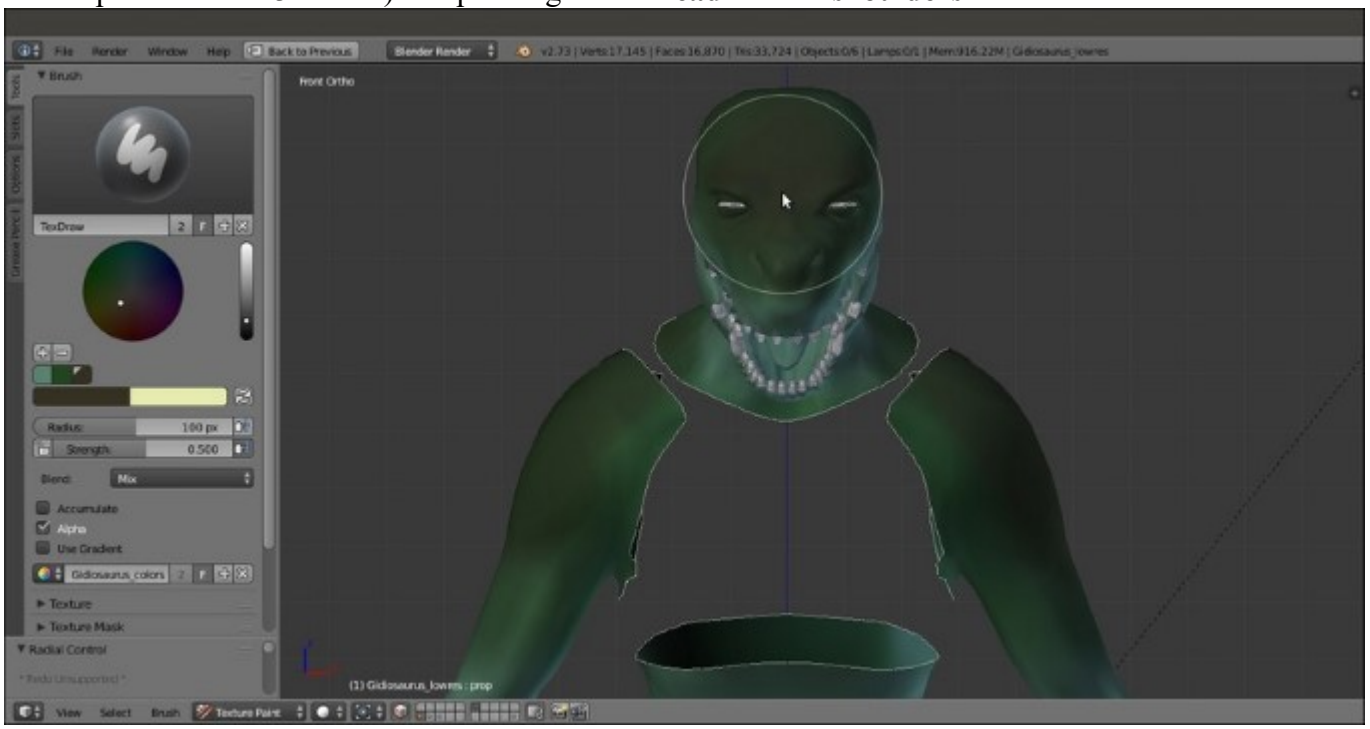


Painting colors on the model

8. Select a brownish color (**R 0.204**, **G 0.188**, **B 0.133**) and add it to the palette; disable the tablet pressure sensitivity for **Radius** and lower the **Strength** to **0.500**. Go into the **Front** view, maximize the 3D viewport (mouse pointer in the window and press *Ctrl* + Up Arrow), and keep on adding shades to the **hands**, **feet**, and **legs**:

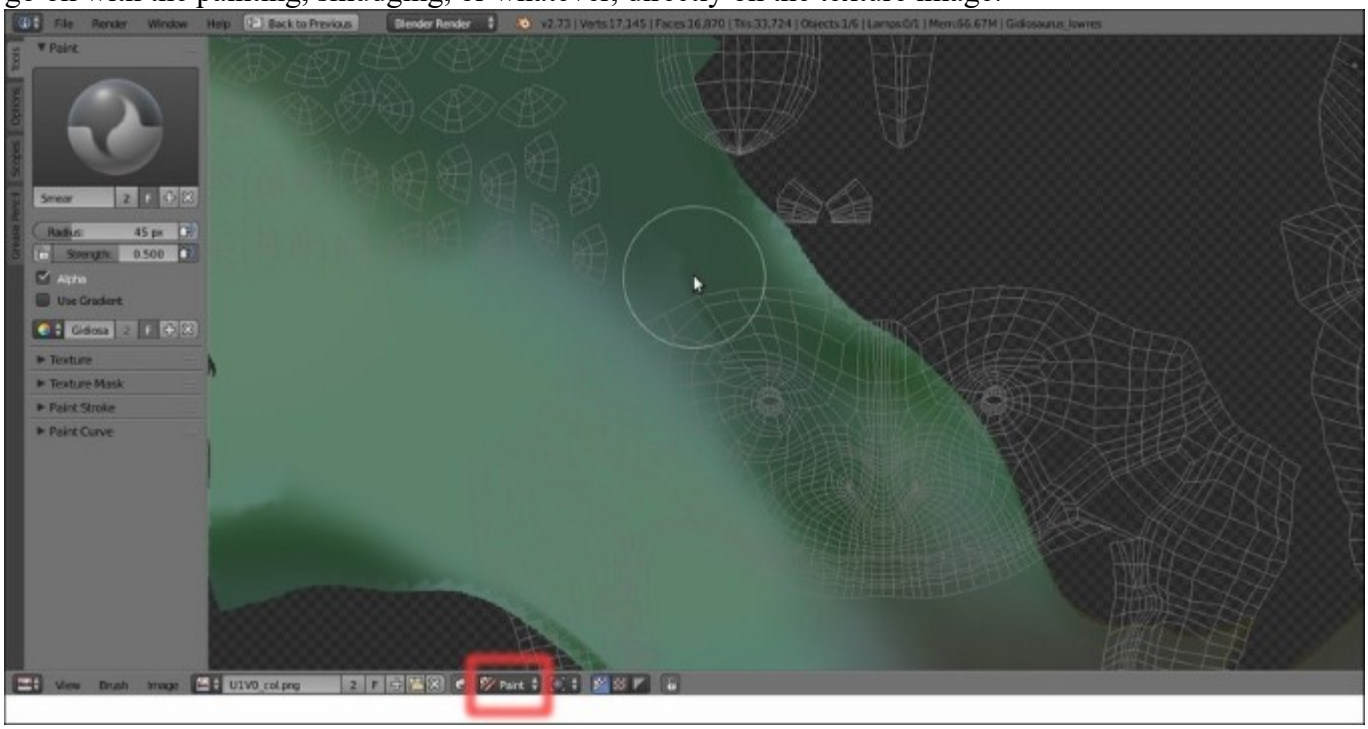
Shading the character's limbs with darker hues

9. Increase the **Radius** value to **100** (using the slider or by pressing the *F* key and moving the mouse pointer in the 3D view) and painting on the **head** and the **shoulders**:



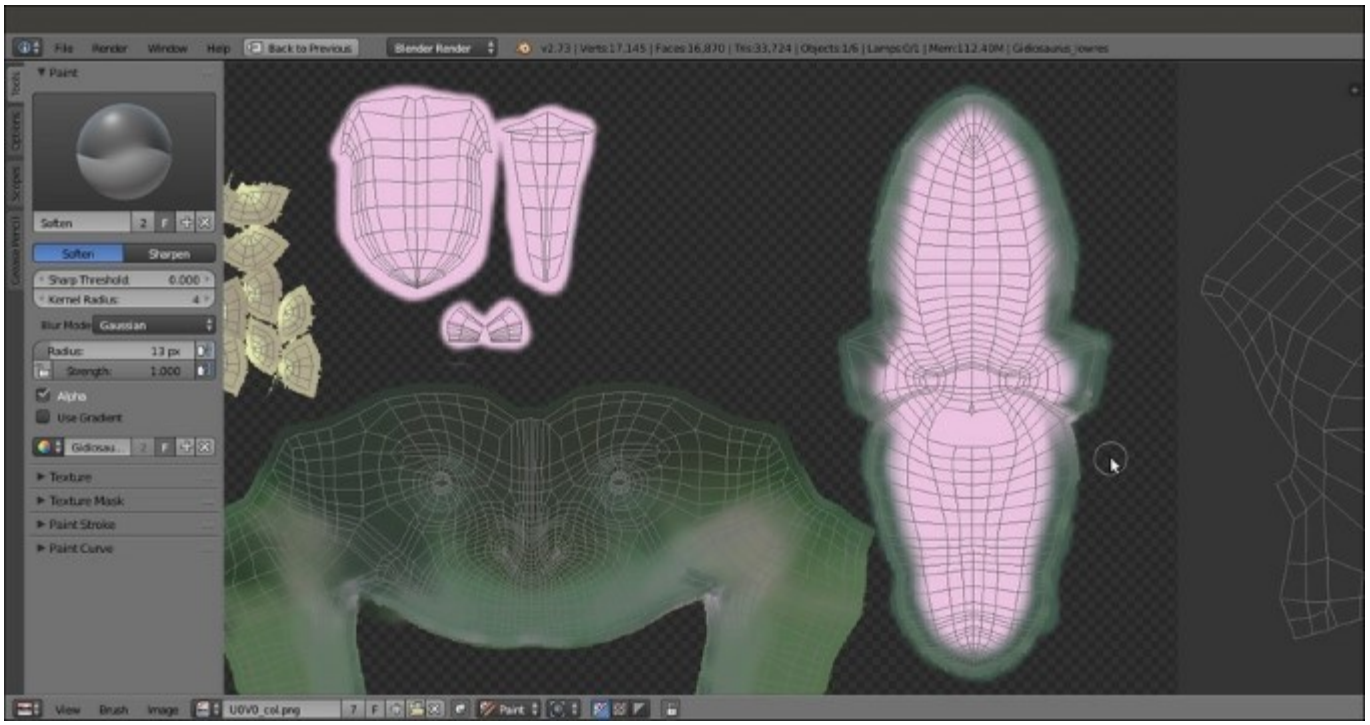
Shading the head and shoulders

10. If, for any reason, it becomes difficult to paint directly on the model through the 3D viewport, you can maximize the involved **UV/Image Editor** window (mouse pointer in the window and press *Ctrl* + Up Arrow), click on the **Mode** button (*Editing context being displayed*) in the toolbar (which by default shows **View**), and switch it to **Paint**. Press *T* to call the **Tool Shelf** and go on with the painting, smudging, or whatever, directly on the texture image:



Painting directly on the image map in the UV/Image Editor window

For example, this is the way I painted the inside of the **mouth** and the **tongue**, then went back to the 3D viewport to smudge and soften the joining line of the pink tissue with the green skin at the borders:



Working on the inside of the mouth in the UV/Image Editor window

I'm not going to show you every step in this process, but basically this is the procedure I used to paint the diffuse coloration for the character. I also added lighter and warmer colors for the face's areas close to the **mouth** and more bluish and colder hues to de-saturate the brownish **hands** and **feet**, and then inverting the **enamel** vertex group to paint in **Edit Mode**, through the use of the **Mask** tool, the **teeth** and **talons** as well:

The completed Gidiosaurus diffuse color texturing

To have a look at the final **Gidiosaurus_colors** palette, open the `Gidiosaurus_painting_BI_02.blend` file provided.

11. When you are done, go to the top left **UV/Image Editor** window, **blank_U0V0**, and click on the **Image** item in the toolbar. Save the image texture in the `textures_making` folder as `U0V0_col.png`, and do the same with the other **4** image textures.
12. To keep the palette, save the file.

How it works…

There is not that much to explain about this recipe, except I just want to highlight the fact that we disabled the **Occlude**, **Cull**, and **Normal** items in the **Options** tab under the **Tool Shelf**. This is so we were able to paint (from the **Side** view) on both sides of the model at the same time; in fact, with these settings disabled, the mesh is *not occluding itself*. It seems that all three items must be disabled for this to work.

Instead, to smear and/or soften the texture on some parts, for example, the inside of the **mouth**, we had to re-enable them, in order to prevent our mouth-painting from accidentally overwriting our skin-painting.

Remember, the **Occlude**, **Cull**, and **Normal** items should always be enabled if you want to paint only on the model's surface right under your brush. You can disable them to paint on the front/outer and the back/inside of the mesh at the same time.

See also

- http://www.cgmasters.net/free-tutorials/layered-painting-in-blender-2-72/
- http://blender.stackexchange.com/tags/texture-painting

Painting the color maps in Cycles

There are no differences in painting in **Blender Internal** or in **Cycles**, because the **Paint Tool** is exactly the same; the only difference is in the preparation of the materials.

In this recipe, we are not going to repeat the procedure already explained in the previous one; we'll just set up the file for the painting and test whether it's possible to paint in real time on all **5** image textures at the same time, as it is in **Blender Internal** (*spoiler*: it is).

Getting ready

Let's start with the `Gidiosaurus_painting_BI.blend` file; in that file, we already have the **UV/Image Editor** windows set and the **5** materials assigned to the **5** different **UDIM** tiles and parts of the mesh.

In case you want to start with a brand new file, here you need to repeat the steps of the *Preparing the model to use the UDIM UV tiles* recipe in this chapter. Then, continue with the following:

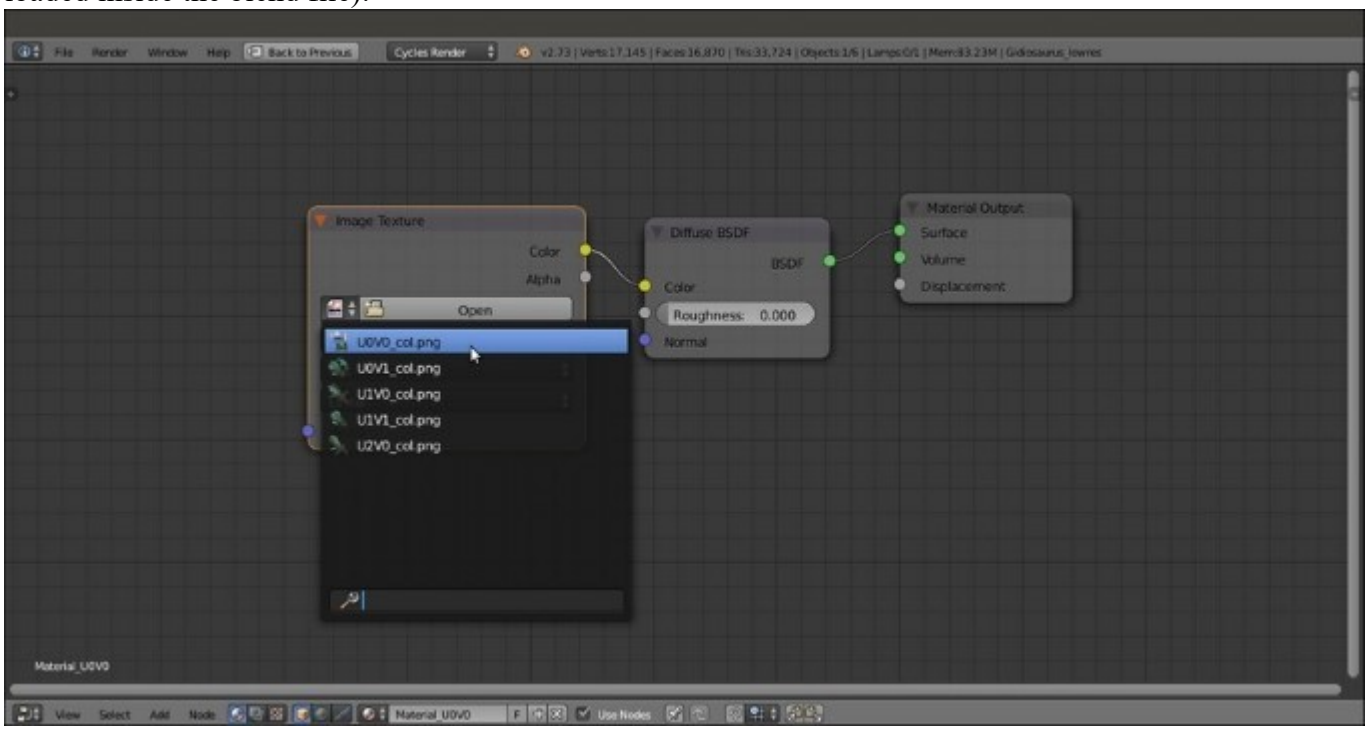
1. Be sure you're in **Object Mode**.
2. Go to the main top header and click on the *Engine to use for rendering* button; switch from **Blender Render** to **Cycles Render**.
3. Split the 3D view into two horizontal rows and change the top one into a **Node Editor** window; press the *N* key to get rid of the **Properties** sidepanel.
4. In the **Material** window, select the **Material_U0V0** slot; click on the **Use Nodes** button or select the **Use Nodes** checkbox in the **Node Editor** toolbar:

Enabling the nodes for the materials under Cycles

5. Put the mouse pointer inside the **Node Editor** window and add an **Image Texture** node (press the *Shift* + *A* keys and in the pop-up menu, go to the **Texture** item to select **Image Texture**). Connect its **Color** output to the **Color** input socket of the **Diffuse BSDF** node.

   At this point, if we haven't already painted the color textures in **Blender Internal**, we should load the `blank_U0V0.png` image in the **Image Texture** node and then do the same for the other **4** materials.

   Instead, because we already have the color textures, let's load them in the **Cycles** materials. To see whether everything works as it should, we'll paint on them through the 3D viewport.
6. Click on the double arrows to the side of the **Open** button in the **Image Texture** node and select the **U0V0_col.png** item from the pop-up menu (remember that the **5** color textures are already loaded inside the blend file):



Selecting one of the already loaded images in the Image Texture node for the materials under Cycles

7. Repeat step 4 to step 6 for the other **4** materials.

How to do it…

Now, the steps are really simple:

1. Go to the **Brush** subpanel and switch the foreground color with the background black color.
2. Trace in the 3D viewport, a continuous stroke enveloping all the **Gidiosaurus** body parts:

The single stroke test under Cycles

This is the proof that it works exactly as in **Blender Internal**.

3. Press *Ctrl* + *Z* to undo the stroke and save the file as `Gidiosaurus_painting_Cycles.blend`.

Chapter 11. Refining the Textures

In this chapter, we will cover the following recipes:

- Sculpting more details on the high resolution mesh
- Baking the normals of the sculpted mesh on the low resolution one
- The Armor textures
- Adding a dirty Vertex Colors layer and baking it to an image texture
- The Quick Edit tool

Introduction

In Chapter 10, *Creating the Textures*, we have prepared the **color** and **bump** texture images for the **Gidiosaurus** skin. In this chapter, we'll see the process for creating some additional (but equally important, nonetheless) textures, both for the character and the iron **Armor**.

Sculpting more details on the high resolution mesh

In Chapter 2, *Sculpting the Character's Base Mesh*, we sculpted the **Gidiosaurus** character's features, obtaining a high resolution mesh that we re-topologized in the following Chapter 4, *Re-topology of the High Resolution Sculpted Character's Mesh*, to have a low resolution mesh for easy rigging and texturing.

Because in the following recipe (*Baking the normals of the sculpted mesh on the low resolution one*) we are going to bake the normals of the sculpted mesh on the low resolution one, we should now add as much detailing and finishing to the sculpted model.

I'm not going to explain every step in detail, here, because the procedure is the same as already seen in the Chapter 2, *Sculpting the Character's Base Mesh*, so just a quick tour to show what I've done should be fine.

Getting ready

Let's start by preparing the file:

1. Start Blender and load the `Gidiosaurus_painting_BI.blend` file; if necessary, go out of **Texture Paint** mode back to **Object Mode** and save the file as `Gidiosaurus_details_sculpt.blend`.
2. Collapse all the **UV/Image Editor** windows on the left of the screen and then join them with the 3D viewport (put the mouse pointer on the edge of one of the two windows; as it changes into a two opposite arrows pointer, right-click and in the **Area Options** pop-up menu, left-select the **Join Areas** item; then, move the mouse pointer towards the window to be eliminated and left-click to join them).
3. Join the **Material** and **Texture** windows in the main **Properties** panel, switch to the **Object Data** window, and enlarge the 3D viewport as much as possible.
4. Click on the **File** item in the main top header and then select the **Append** item (or else, directly press the *Shift* + *F1* keys); navigate to the `Gidiosaurus_retopology_02.blend` file, click on it, and then click on the **Object** item (folder) to select the **Gidiosaurus** item.
5. Click on the **Append from Library** button on the top-right of the screen and then go to the **Outliner** window to click on the **eye** and the **arrow** icon buttons (*Restrict view-port visibility* and *Restrict view-port selection*) and enable both the object visibility and selection in the 3D viewport.
6. Move the appended high resolution **Gidiosaurus** mesh to the **14th** scene layer (*M* key):

The appended, sculpted Gidiosaurus mesh

7. Press *N* to call the **Properties** sidepanel and in the **Display** subpanel, enable the **Only Render** item; go down to the **Shading** subpanel, enable the **Matcap** item, and then select your favorite matcap type (mine is always the brick red colored **Zbrush**-like).
8. Enable the **12th** scene layer to show the **Eyes**; however, in the **Outliner**, just to be sure, disable the selection arrow icon button.
9. Press *N* again to hide the **Properties** sidepanel and then switch to **Sculpt Mode** and save the file.

The Gidiosaurus object ready for the new sculpting session

How to do it…

We are now ready to sculpt again on the **Gidiosaurus** mesh; first, let's do some more settings pertinent to the sculpt tools:

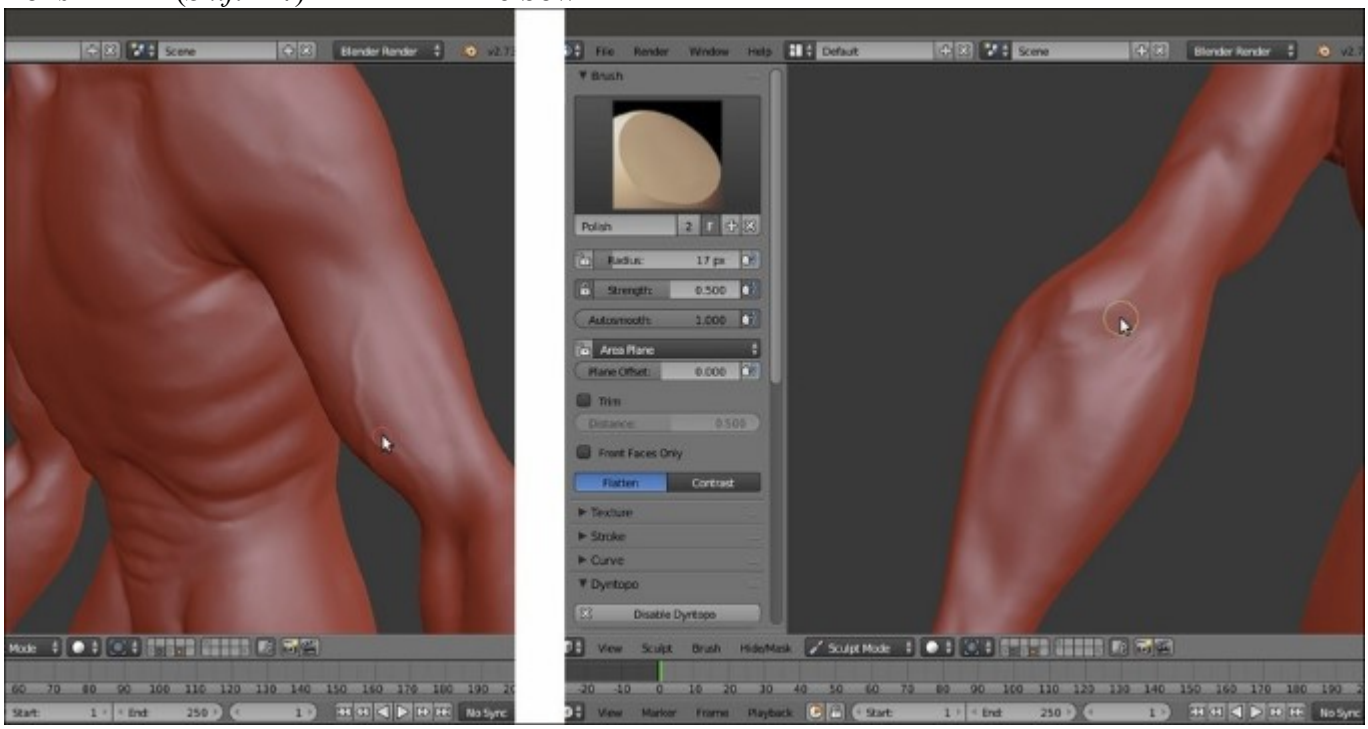
1. Go to the **Dyntopo** subpanel under the **Tool Shelf** and click on the **Enable Dyntopo** button; set **Detail Size** to **1.60** px and check the **Smooth Shading** box.
2. Go down to the **Symmetry / Lock** subpanel to be sure that **Mirror** is enabled for the *x* axis.
3. Click on the **Options** tab and go to the **Options** subpanel to enable the **Fast Navigate** item (the **Threaded Sculpt** item should be already enabled by default).
4. Go back to the **Tools** tab and click on the **Brush** windows; select the **Crease** brush (press the *Shift* + *C* or *5* keys), zoom to the **Gidiosaurus's head**, and start to add **expression folds**:

Adding expression folds with the Crease brush

5. Move to the **throat**, in the **Tool Shelf** panel, switch the effect of the brush from **Subtract** to **Add** through the buttons at the bottom of the **Brush** subpanel (or simply by pressing the *Ctrl* key while sculpting), and add veins to the area:

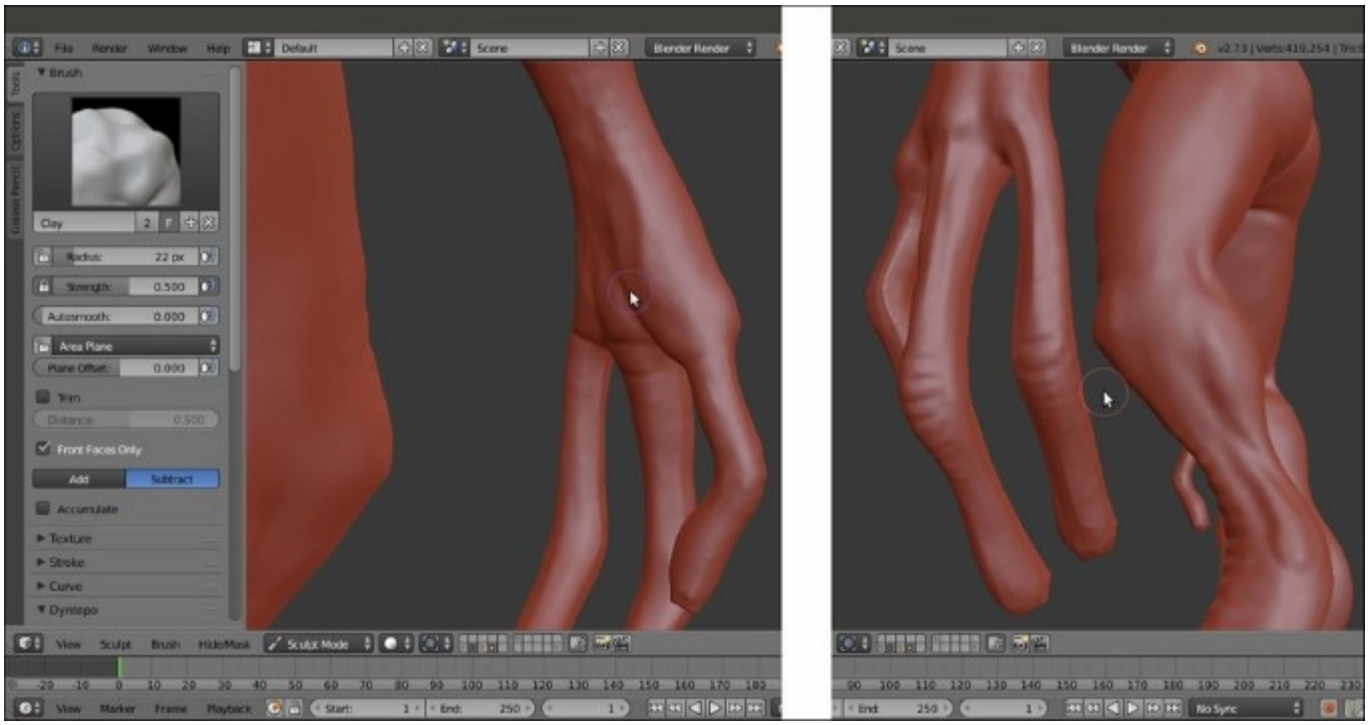
Adding veins under the jaw and on the neck by using the Crease brush again, but with inverted effect

6. By using the same technique, add **veins** also on the **shoulders** and the **biceps**; then, select the **Polish** brush (*Shift* + *4*) and refine the **elbow** a bit:



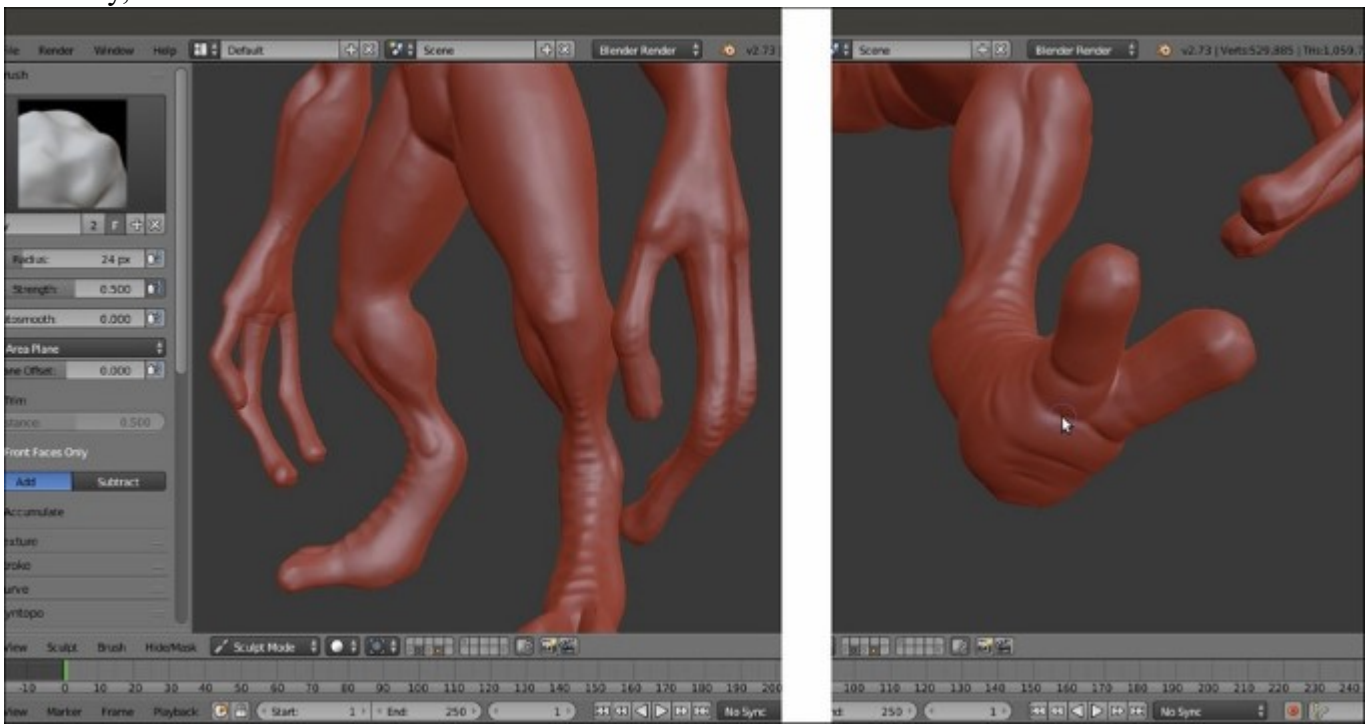
Adding the veins on the arm muscles and polishing the elbow's bulging muscle

7. By using the **Clay** brush (*C* or *3* keys) and also the **Crease** (*Shift* + *C* or *5* keys) and **Pinch** (*P* or *Shift* + *3*) brushes, refine the shape and the folds of the **palm** and add details to the back of the **fingers**. The **Clay** brush can be used in **Subtract** mode too, to carve shapes:



Detailing the palm and the fingers of the hand

8. Similarly, add details and refine the back of the **foot** and the **sole**:



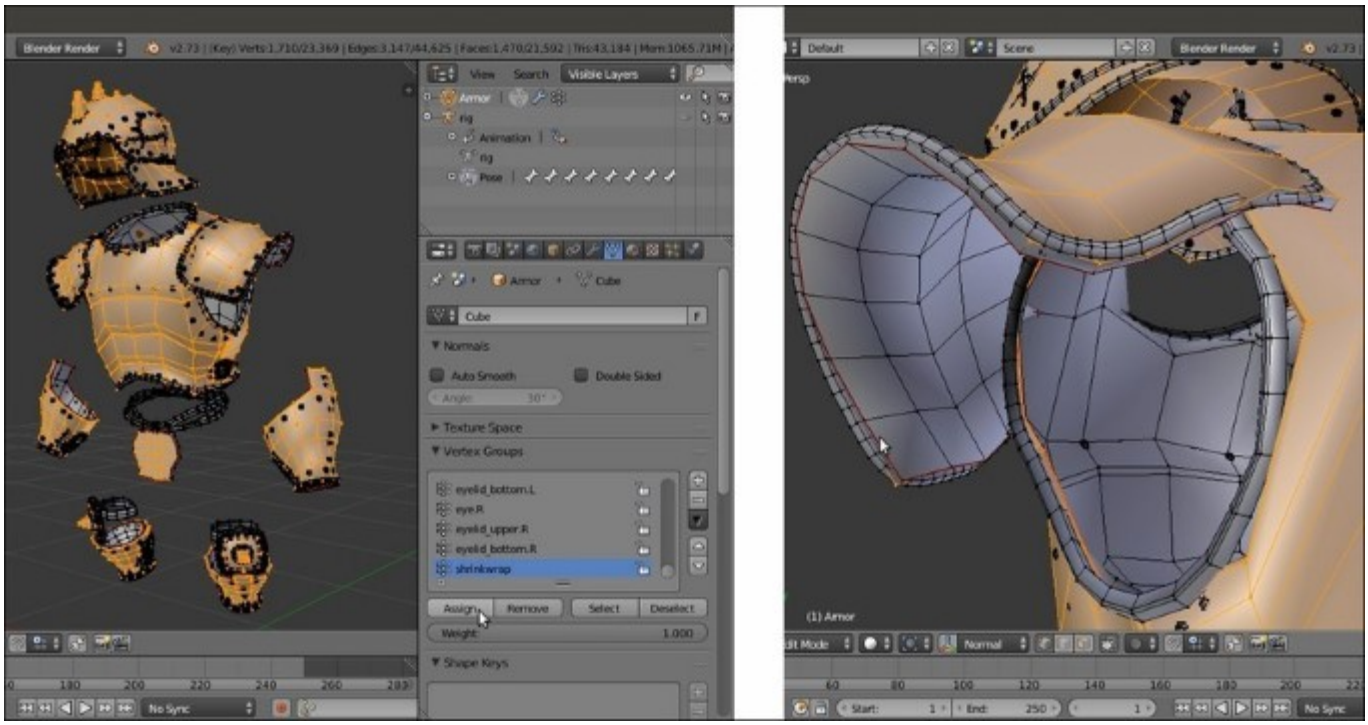
Detailing the feet

9. Use the **Smooth** brush (*S* or *Shift* + *7* keys) to gently soften the character's features; when you are done, save the file.

Smoothing the added features

Now, as we have detailed the body of the **Gidiosaurus**, it would be a good idea to refine the **Armor** also.

10. Switch to the **13th** scene layer; select the **Armor** object and go to the **Shape Keys** subpanel under the **Object Data** window.
11. Select the **Basis** shape key and then click on the **–** icon button to delete it (this leaves the only remaining shape key, **Armor_fix**, as the base one, so permanently applying the morph to the mesh); then, also select the **Armor_fix** shape key and delete it.
12. Repeat the previous steps for the **rivets** and the **Armor_decorations** objects as well.
13. Through the **Outliner** window, *Shift*-select the **rivets**, **Armor_decorations**, and **Armor** objects; then, press *Ctrl* + *J* to join them as a single object.
14. Go to the **Vertex Groups** subpanel and add a new vertex group; rename it as **shrinkwrap**.
15. Enter **Edit Mode** and select the vertices on the outside of the **armor body plates**, leaving the **inside faces** of the **plates**, the **bottom** of the **spaulders**, the **decorations**, the **rivets**, and the **tiers**, unselected; if necessary, use the **seams** to help you to divide the outer from the inner parts of the mesh. Click on the **Assign** button at the bottom of the **Vertex Groups** subpanel:

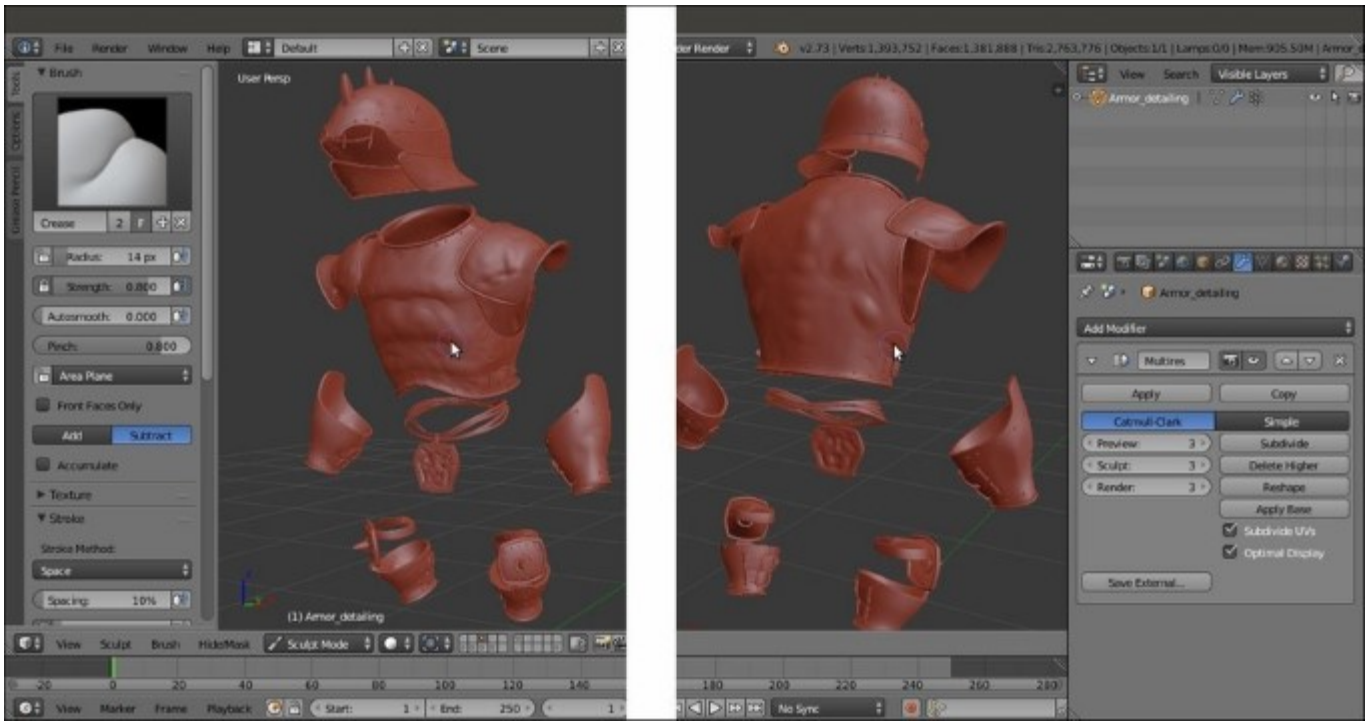


Selecting the outer parts of the Armor

16. Split the 3D view into two windows and change the left one into a **UV/Image Editor** window.
17. Go to the **UV Maps** subpanel under the **Object Data** window, click on the + icon button to add a new UV coordinates layer, and rename it as **UVMap_norm**. Then, click on the camera icon on the right-hand side of the name to make it the active UV layer.
18. Put the mouse pointer in the 3D viewport and press *U*; in the **UV Mapping** pop-up menu, select the **Smart UV Project** item; in the pop-up panel, click on the **Island Margin** value (default = **0.00**) and set it to **0.001**. Leave the other values as they are and click on the big **OK** button at the bottom of the panel.

The UVMap_norm UV coordinates layer for the Armor object

19. Go out of **Edit Mode** and minimize the **UV/Image Editor** window as much as possible; press *Shift* + *D* to duplicate the **Armor** object and move the duplicated one to the **3rd** scene layer.
20. Enable the **3rd** scene layer; go to the **Object Modifiers** window and delete the **Armature** and the **Subdivision Surface** modifiers; in the **Outliner**, rename the new object (now **Armor.001**) as **Armor_detailing**.
21. Assign a **Multiresolution** modifier. Click on the **Subdivide** button until it reaches level **3**; then, check the **Optimal Display** item and go in **Sculpt Mode**. Using the same procedure as before, add scrapes, bumps, deformations, and so on, to the **armor** surface; add some kind of engraving also, for example, on the **groinguard**.



Sculpting the Armor_detailing object

22. Save the file.

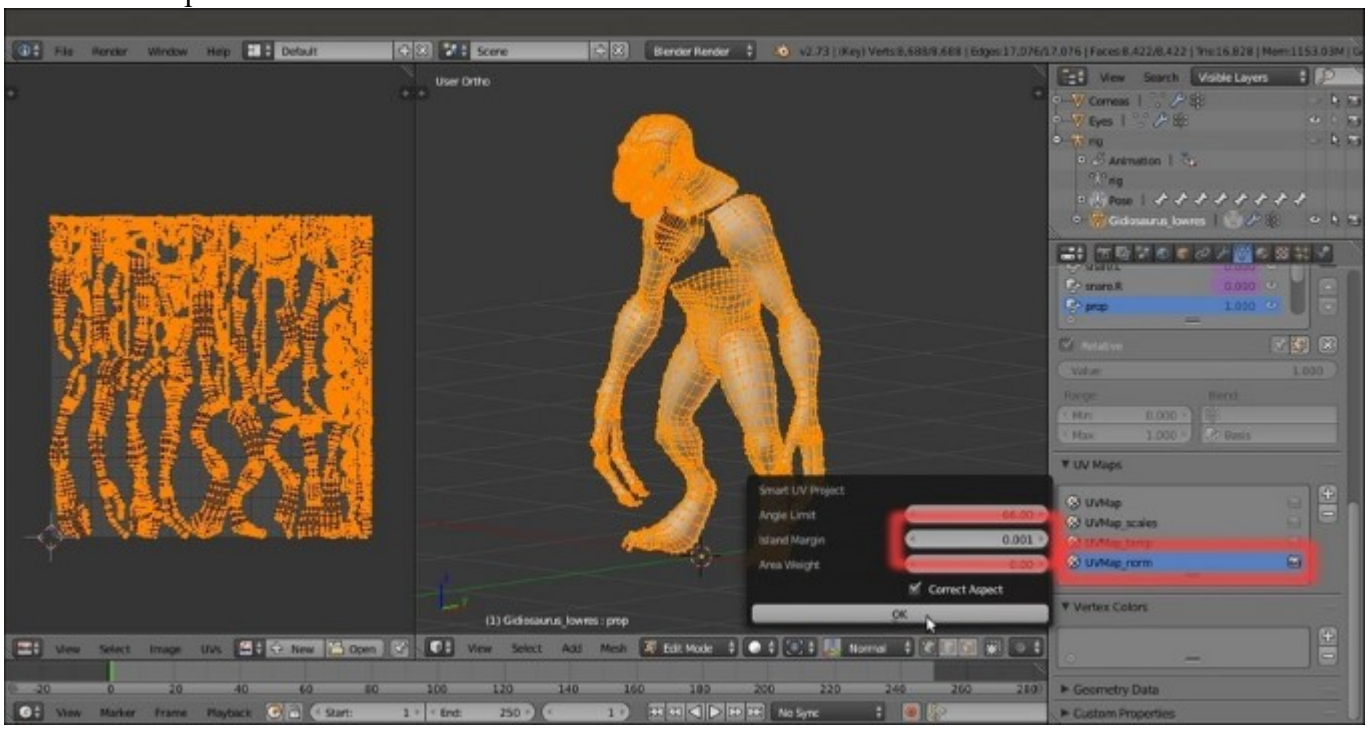
Baking the normals of the sculpted mesh on the low resolution one

At this point, we can transfer all the details sculpted on our high resolution meshes (the **Gidiosaurus** and the **Armor** objects) to the low resolution assets; to do this, we have to **bake** these details as **normal maps**.

Getting ready

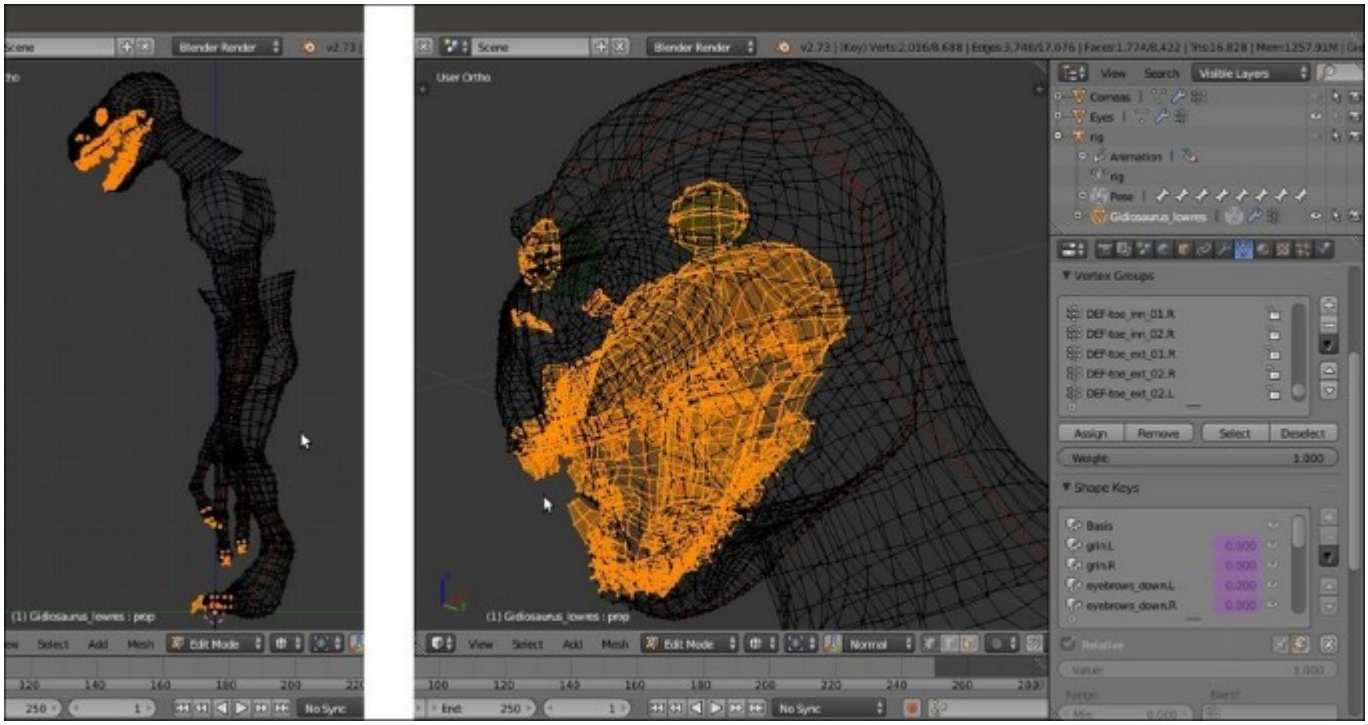
Continue from the previous `Gidiosaurus_details_sculpt.blend` file:

1. Split the 3D viewport into **two** windows and change the left one into a **UV/Image Editor** window.
2. Go to the **11th** scene layer and select the **Gidiosaurus_lowres** object; press the *Tab* key to enter **Edit Mode** and, if necessary, press *A* to select all the vertices.
3. Go to the **UV Maps** subpanel under the **Object Data** window, click on the + icon button to add a new UV coordinates layer, rename it as **UVMap_norm**, and click on the camera icon to make it the active UV layer.
4. Put the mouse pointer in the 3D viewport and press *U*. In the **UV Mapping** pop-up menu, select the **Smart UV Project** item; in the pop-up panel, click on the **Island Margin** value (default = **0.00**) and set it to **0.001**. Leave the other values as they are and click the big **OK** button at the bottom of the panel.



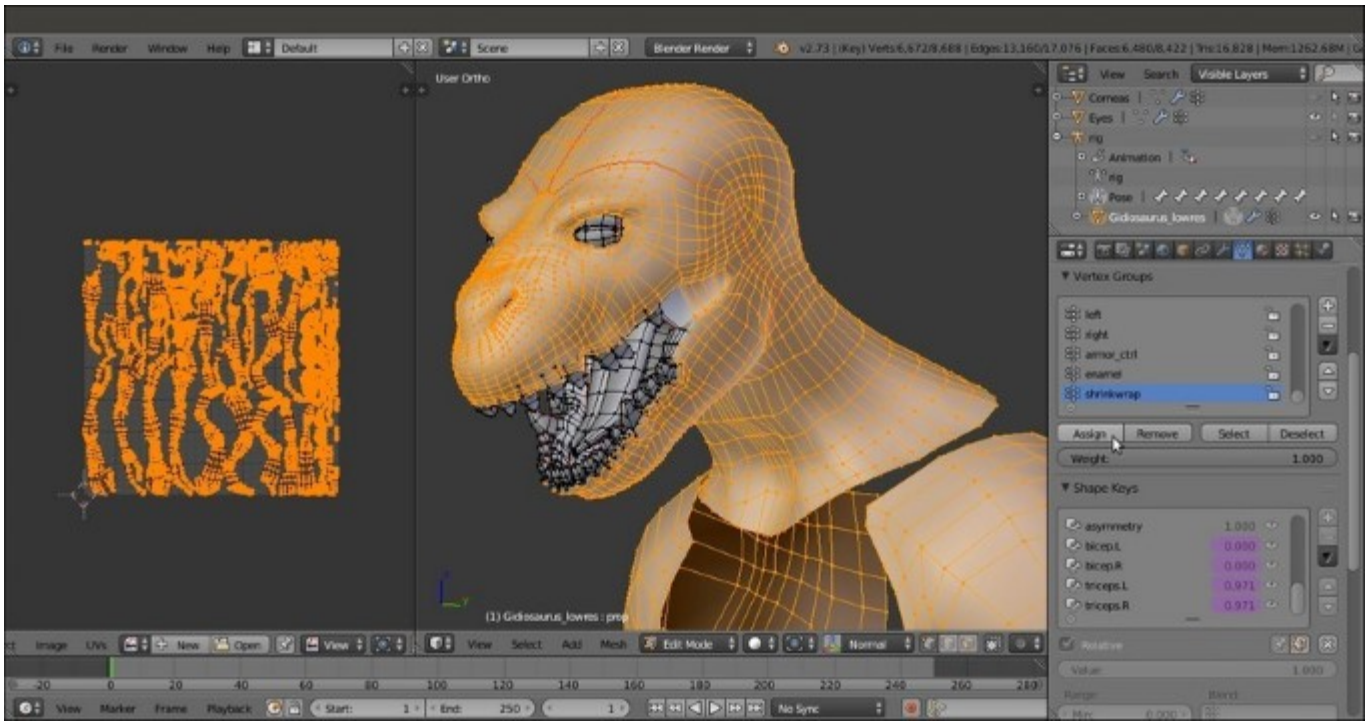
The UVMap_norm UV coordinates layer for the low resolution Gidiosaurus mesh

5. Deselect all the vertices (the *A* key again) and zoom to the **head**; *Shift*-select the vertices of all the parts that don't actually exist in the high resolution sculpted model such as the inside of the **mouth**, the **mouth inner rims**, the **tongue**, the **eyelids**, the inside of the **nostrils**, the **teeth**, and the **talons**:



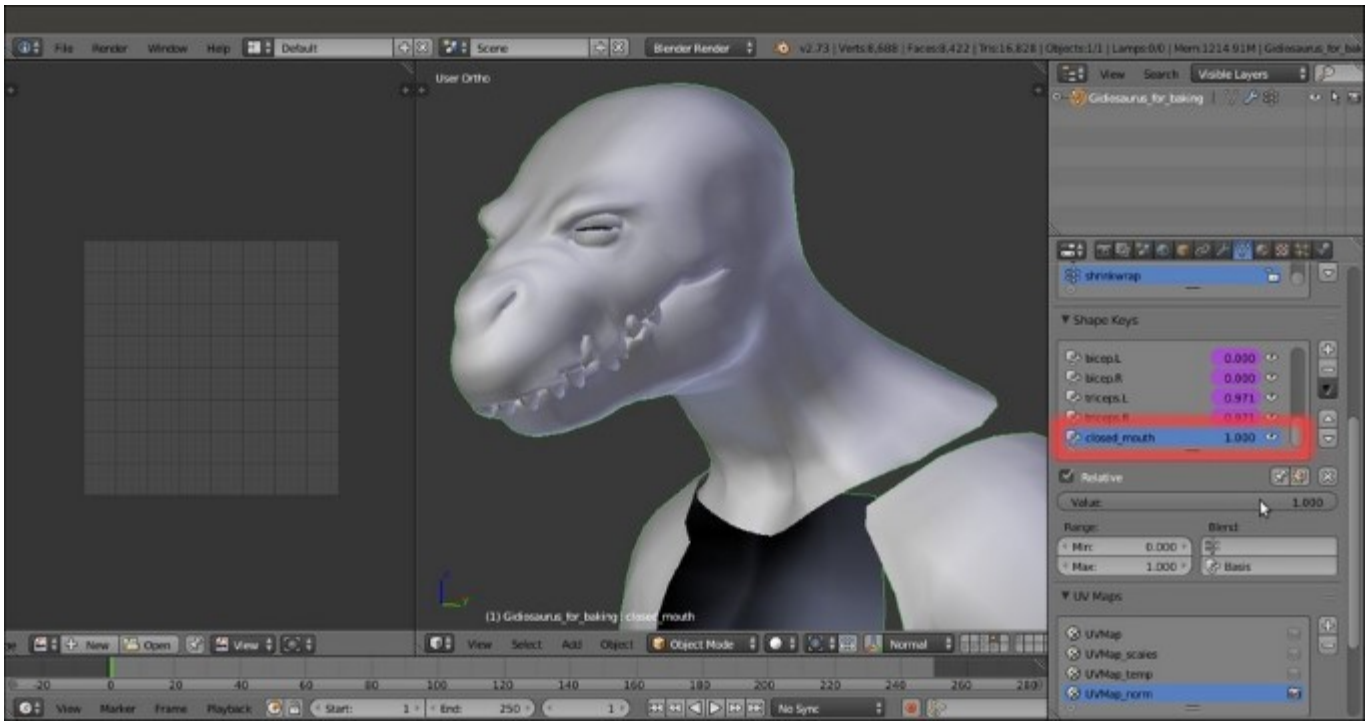
Selecting the low resolution mesh parts that don't have a counterpart in the high resolution sculpted mesh

6. Go to the **Vertex Groups** subpanel under the **Object Data** window and click on the + icon button to add a new vertex group; rename it as **shrinkwrap**.
7. Press *Ctrl* + *I* to invert the selection and then click on the **Assign** button below the vertex group list window in the **Vertex Groups** subpanel:



Assigning the inverted selection to the "shrinkwrap" vertex group

8. Go out of **Edit Mode** and press *Shift* + *D* to duplicate the **Gidiosaurus_lowres** object; move the duplicate to the **4th** scene layer and in the **Outliner** window, rename it as **Gidiosaurus_for_baking**.
9. In the **Outliner**, enable the 3D viewport visibility of the **rig**; select and move the **ctrl_mouth** bone upward to close the **Gidiosaurus's** mouth and then hide the **11th** scene layer.
10. Reselect the **Gidiosaurus_for_baking** object and go to the **Object Modifiers** window; click on the **Apply as Shape Key** button of the **Armature** modifier.
11. Go to the **Shape Keys** subpanel under the **Object Data** window to find a new shape key at the bottom of the list: **Armature**, with the value of **0.000**.
12. Rename the new shape key as **closed_mouth** and set the value to **1.000**:



The closed_mouth shape key

13. Go to the **Object Modifiers** window and assign a **Shrinkwrap** modifier to the **Gidiosaurus_for_baking** object; as **Target**, select the **Gidiosaurs_detailing** object and then click on the **Vertex Group** slot to select the **shrinkwrap** vertex group.

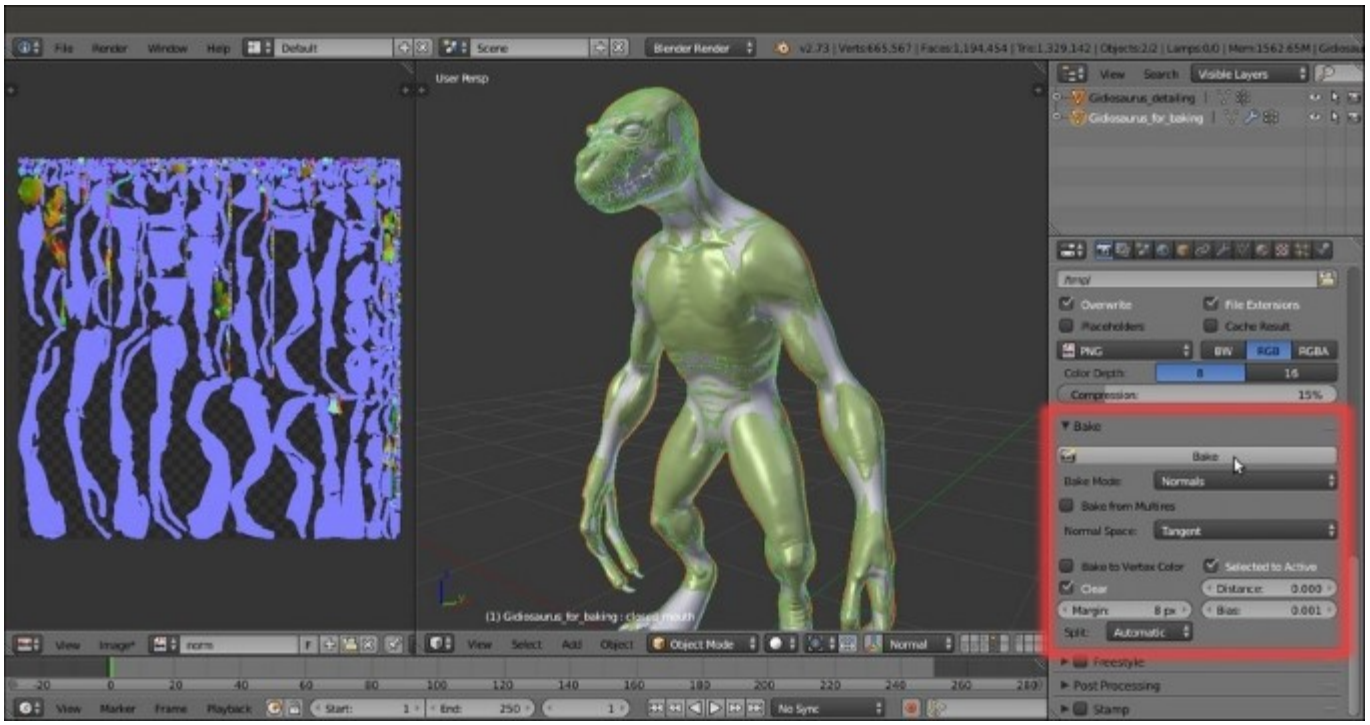
In the following screenshot, you can see the effect of the **Shrinkwrap** modifier on the low resolution mesh with the **Subdivision Surface** modifier enabled also for the 3D viewport:

The "shrinkwrapped" low resolution Gidiosaurus mesh

How to do it…

After this quite *intensive* file preparation, let's go with the baking itself:

1. Enter **Edit Mode** and select all the mesh vertices; in the **UV/Image Editor** window, add a new **3072** x **3072** blank image and rename it as `norm`.
2. Go out of **Edit Mode**, enable the **14th** scene layer, select the **Gidiosaurus_detailing** object, and then *Shift*-select the **Gidiosaurus_for_baking** object.
3. Go to the **Render** window, scroll the panel down and, in the **Bake** subpanel, check the **Selected to Active** item. Set **Margin** to **8** pixels, the **Bake Mode** to **Normals**, and the **Normal Space** to **Tangent**; click on the **Bake** button to start the baking:



The baked normals' image map, the two overlapping and selected objects, and the Bake subpanel

4. Click on the **Image** item in the **UV/Image Editor** window toolbar to save the baked image as `norm.png` inside the `texture_making` folder.

How it works…

To close the **mouth** (to conform it to the sculpted mesh), we moved the control bone in the rig and then applied the **Armature** modifier as a **shape key**; be aware that a modifier cannot be applied to a mesh with shape keys (you get a warning message), so we had to use the **Apply as Shape Key** option or delete all the shape keys with drivers and redo them later. In this case, however, it wouldn't have been necessary to duplicate the **Gidiosaurus** low resolution mesh, but we did it anyway to keep things simpler and cleaner.

Right before the baking, a **Shrinkwrap** modifier has been assigned to the lowres **Gidiosaurus_for_baking** object, to conform its surface to the high resolution sculpted **Gidiosaurus_detailing** object and avoid any possible intersection between the two meshes (that would give ugly artifacts in the baked image); we used the **shrinkwrap** vertex group to keep the vertices that don't have a counterpart on the high resolution mesh (**teeth**, **eyelids**, **inner mouth**, and so on) out of the modifier influence.

As you can see in the following OpenGL screenshot, comparing the sculpted and the low res **Gidiosaurus** meshes, the result of the baked normals on the low resolution object is pretty good and effective:

Comparison between the high resolution sculpted mesh and the low resolution object with the baked normal map

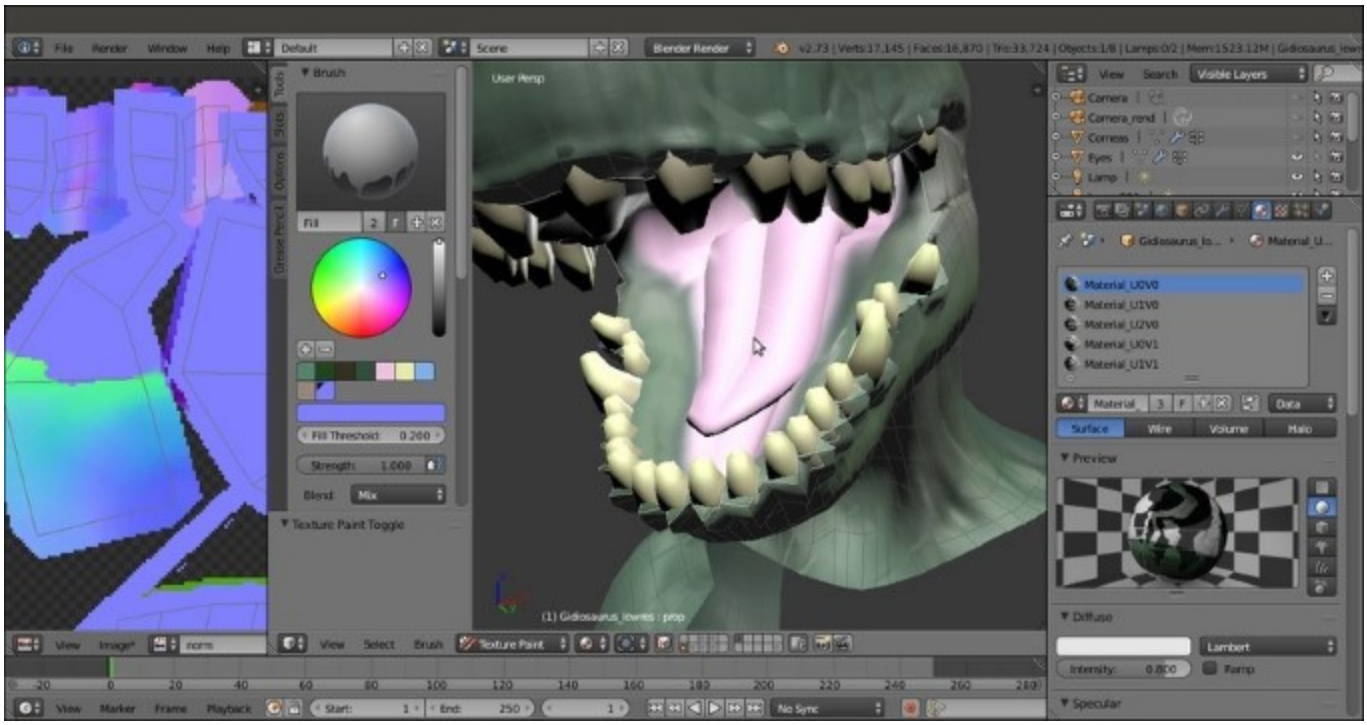
There's more…

As we reopen the **mouth** by lowering the **close_mouth** shape key value to **0.000** or also by simply assigning the baked normal map to the **Gidiosaurus_lowres** object, we see that something is wrong inside the **mouth** (and, actually, also on the **teeth** and **talons**): the normals have been calculated for those parts too, but they show wrong and weird artifacts because there were no counterparts to take the normals from in the sculpted high resolution mesh.

Artifacts of the normal map in some mesh parts

The solution in this case is very simple: we must paint the areas on the baked normal map corresponding to the afflicted parts, such as the **teeth**, the **tongue**, and so on, with a *flat normal* color (**R 0.498**, **G 0.498**, **B 1.000**) to *flatten* and therefore erase the unwanted details.

We can do this directly in Blender, by selecting the vertices of the areas to be painted on and enabling the **mask tool** in the 3D viewport toolbar:



Flattening the unwanted artifacts by painting on the normal map

Alternatively, we can do it in an external painting software program such as Gimp; in this case, just delete the vertices of the parts that you don't want to change in the mesh and export the UV layer of all the remaining parts to be used as a guide to paint.

The Armor textures

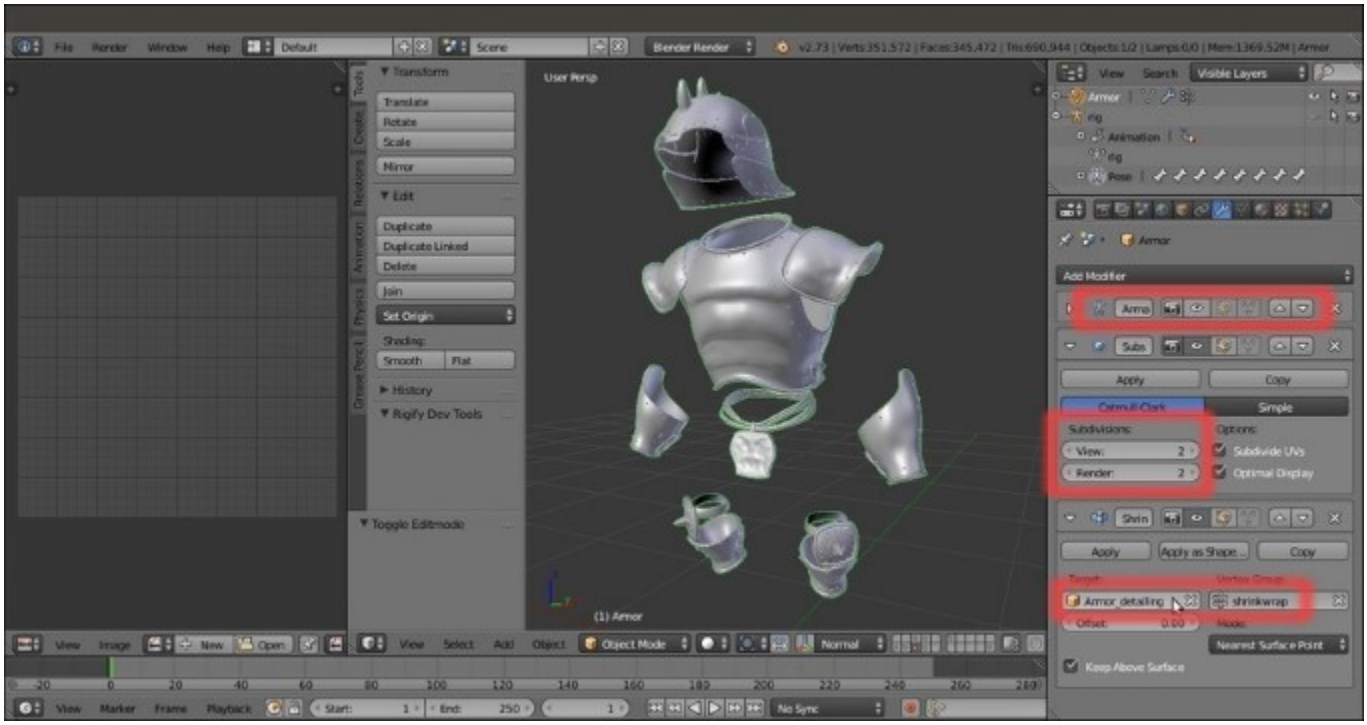
The same procedure used in the previous recipe must be used for the **Armor** object, to bake the normals of the sculpted high resolution version on the low poly one.

Getting ready

So, in short, we will do the following:

1. Enable the **13th** scene layer; select the **Armor** object and go to the **Object Modifiers** window.
2. Temporarily, disable the **Armature** modifier both for rendering, and the viewport, and be sure that the **Subdivision Surface** modifier levels are both set to **2**.
3. Assign a **Shrinkwrap** modifier with a target to the **Armor_detailing** object; in the **Vertex Group** slot, select the **shrinkwrap** vertex group and, just to be sure, also check the **Keep Above Surface** item.

Also, in this case, thanks to the **shrinkwrap** vertex group, only the outside of the **armor** mesh is conformed to the sculpted mesh; the insides are not important and can even be deleted (only for the baking and, of course, on a duplicated **armor** object, as we did with the **Gidiosaurus_for_baking** object). In any case, they will be barely visible.



The Armor object prepared for the baking

How to do it…

Let's now bake the sculpted geometry in a few steps:

1. Enter **Edit Mode** and select all the mesh vertices; in the **UV/Image Editor** window, add a new **3072** x **3072** blank image and rename it as `norm2`.
2. Go out of **Edit Mode**, enable the **3rd** scene layer and select the **Armor_detailing** object, and then *Shift*-select the **Armor** object.
3. Go to the **Render** window, scroll the panel down; in the **Bake** subpanel, check the **Selected to Active** item, and set the **Margin** to **8** pixels, the **Bake Mode** to **Normals**, and the **Normal Space** to **Tangent**. Then, click on the **Bake** button.
4. Click on the **Image** item on the **UV/Image Editor** window toolbar to save the baked image as `norm2.png`, inside the `texture_making` folder.
5. Save the file as `Gidiosaurus_baking_normals.blend`.

In the following OpenGL screenshot, you can see the comparison between the sculpted and the low resolution **Armor** objects with the assigned normal map:

A comparison between the sculpted and the normal map versions of the Armor

There's more…

Inside the `texture_making` folder provided with this cookbook, there is also an already seamless `iron_tiles.png` image to be used for the **Armor**; it has been made seamless in **Gimp**, but after the

mapping on the model, we'll need to fix some visible seams again by using the **Clone** brush of the Blender **Paint Tool**.

I won't go through all the required steps here, because this would be a repetition of recipes already explained in Chapter 10, *Creating the Textures*.

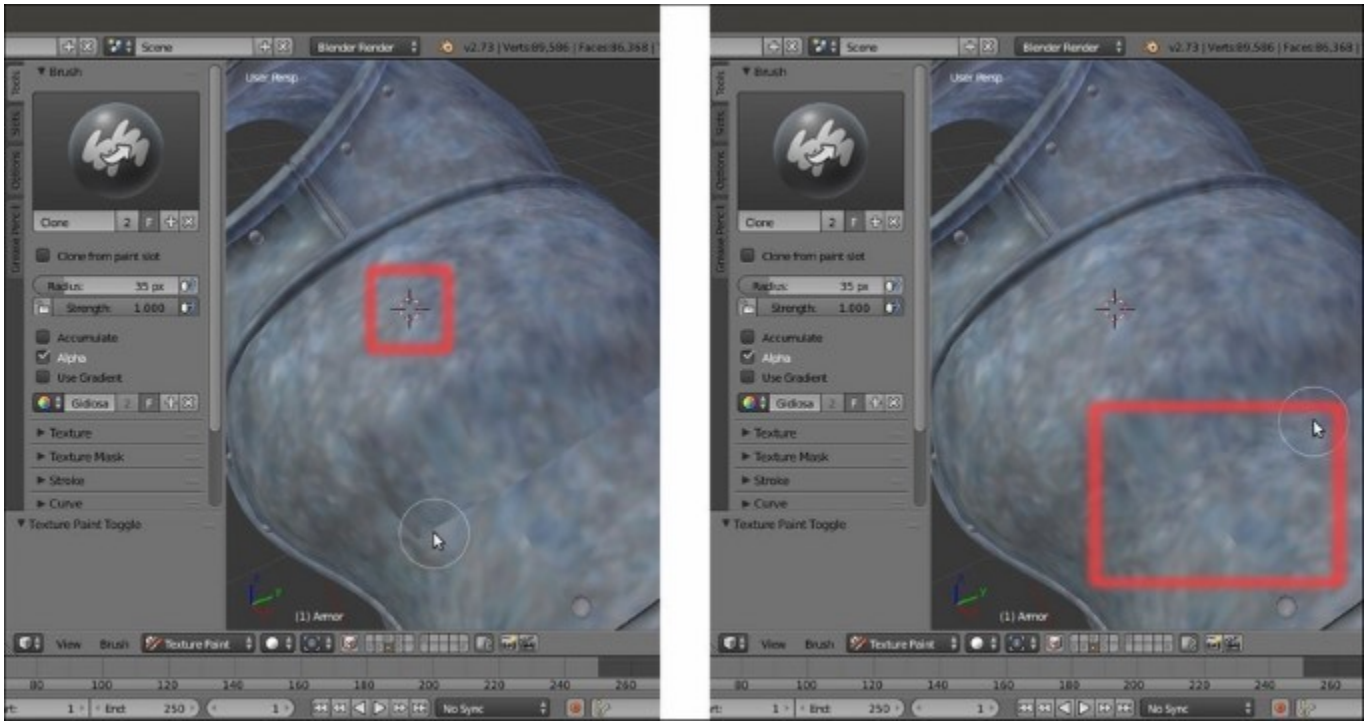
Note

Just remember that all we have to do is to bake the seamless `iron_tiles.png` image, which is mapped on the **UVMap_rust** coordinates layer, on the **UDIM UVMap** coordinates layer; in this case, shared into **two** tiles spaces and then fix the visible seams on the baked images.

So, we have to add two materials to the **Armor** object; each one with its own image texture and assigned to the vertices corresponding to each tile, and then also add the `iron_tiles.png` image to each affected material.

In short, we have to replicate the steps of the *Preparing the model to use the UDIM UV tiles*, *Baking the tileable scales texture into the UV tiles*, and *Painting to fix the seams and to modify the baked scales image maps* recipes from Chapter 10, *Creating the Textures*.

To use the **Clone** brush (press the *1* key to call it after entering **Texture Paint** mode), press *Ctrl* + left-click on the area of the mesh you want to clone from; this will place the **3D Cursor** in that location. Then, left-click on the seams to clone the texture from the area under the **3D Cursor**:



The Clone brush is cloning the texture area at the 3D Cursor location

Besides the **Clone** brush, in this case, it is also possible to fix the seams with the **Smear** brush (*4* key).

When you are done, save the two iron images as `iron_U0V0.png` and `iron_U1V0.png` inside the `texture` folder.

See also

Be aware that the first following link is for Blender version **2.6** (seems there is very little official documentation for version **2.7** at the moment), and a few things in the **Paint Tool** have changed; in any case, I think it can still be an interesting reading:

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Textures/Painting
- http://www.blender.org/manual/render/blender_render/textures/painting.html

Adding a dirty Vertex Colors layer and baking it to an image texture

Let's see now how to add a *dirty* map through the **Vertex Colors** tool and how to bake it to an image texture; such a texture map can be useful for the creation of the shaders (which we'll see in the next chapter).

Getting ready

To do this, we are going to use an already set `.blend` file:

1. Start Blender and open the `Gidiosaurus_baking_normals.blend` file; save it as `Gidiosaurus_baking_dirty.blend`.
2. Put the mouse pointer in the 3D view and press the *Z* key twice to switch into **Solid** viewport shading mode; click on the **14th** scene layer to enable the visibility of the **Gidiosaurus_detailing** object.

How to do it…

Let's first go with the creation of the **Vertex Colors** layer:

1. In the **Outliner**, select the **Gidiosaurus_detailing** object and then click on the mode button in the 3D viewport toolbar to select the **Vertex Paint** mode:

Selecting the Vertex Paint mode item

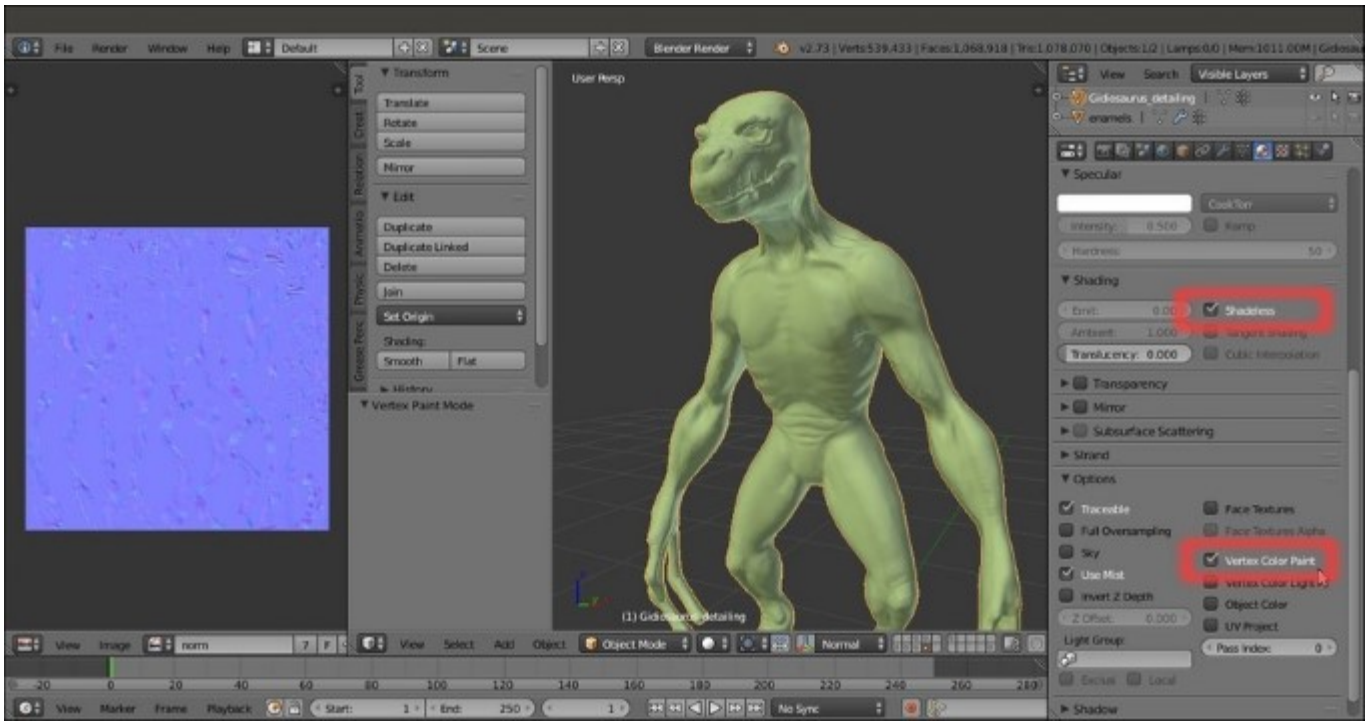
2. Now, click on the **Paint** item in the 3D viewport toolbar and from the menu, select the **Dirty Vertex Colors** item; the **Gidiosaurus** mesh, first filled with a plain white color, gets shaded in grayscale tones:

Using the Dirty Vertex Colors tool

3. Expand the last operation panel at the bottom of the **Tool Shelf** and press *Ctrl*+click on the **Dirt Angle** slot to enter the value **90°**; the grayscale shading on the mesh gets a lot more darker and contrasted:

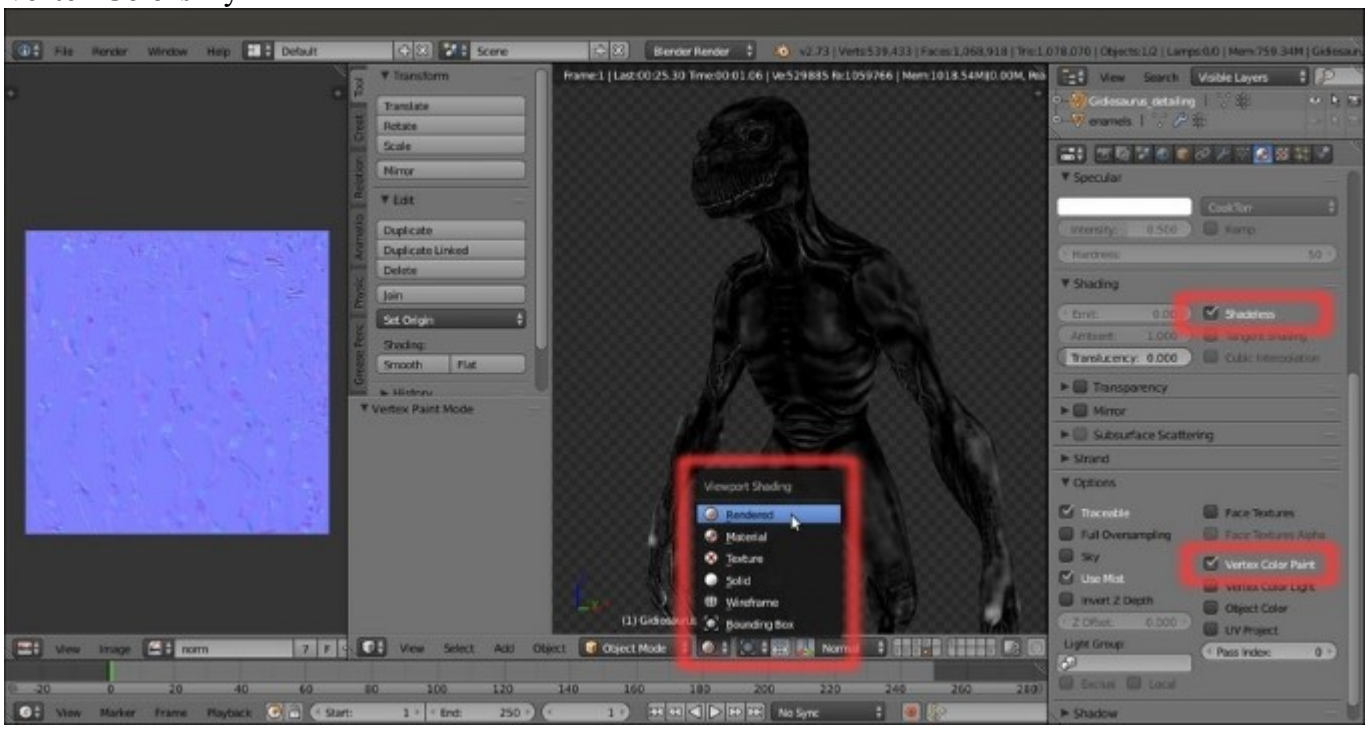
Tweaking the settings for the Dirty Vertex Colors tool

4. Go back in **Object Mode** and then move to the **Material** window, where the **Body** material is already assigned to the high resolution mesh. Scroll down the panel to reach the **Shading** subpanel and enable the **Shadeless** item; then, reach down the **Options** subpanel and enable the **Vertex Color Paint** item:



The Shadeless and the Vertex Color Paint items

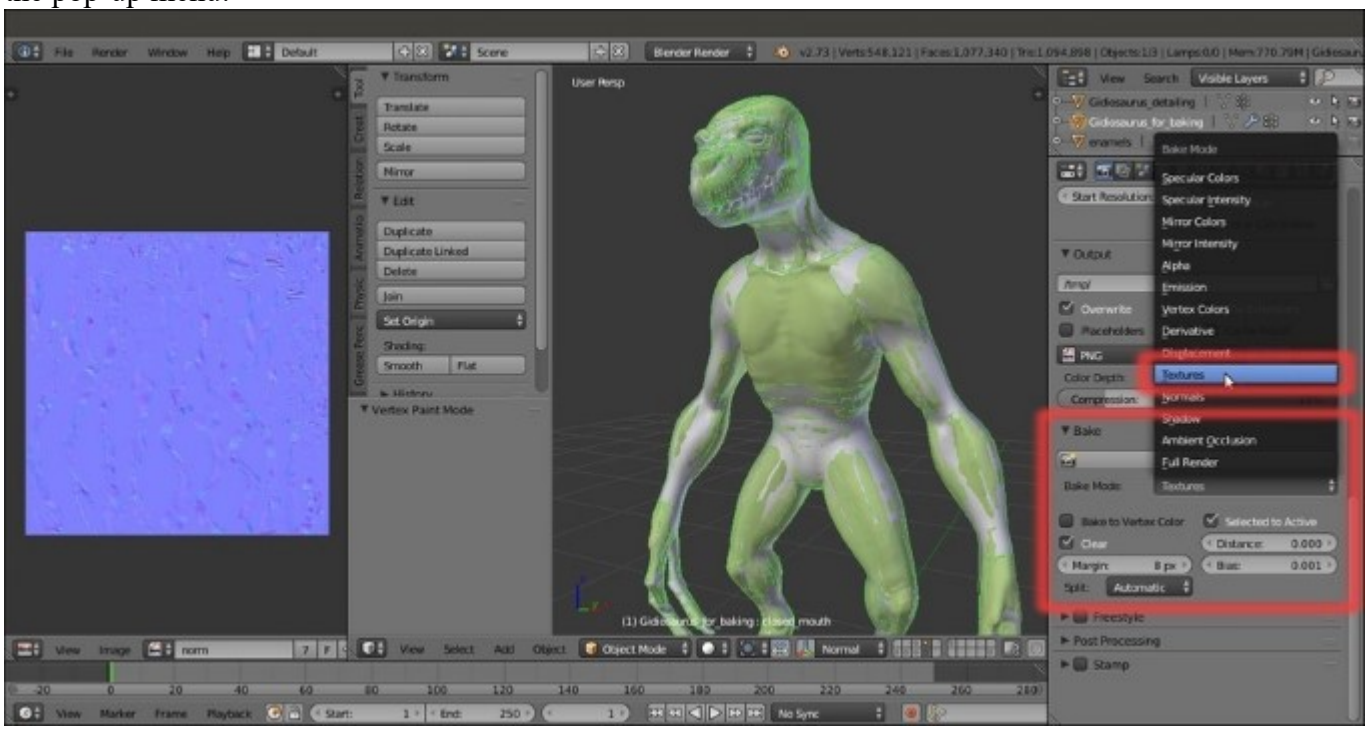
To understand the effect of the items we enabled in the **Material** window, just switch to the **Rendered** viewport shading mode; the mesh surface is self-illuminating and showing the dirty **Vertex Colors** layer:



The Dirty Vertex Colors layer visualized in the Rendered preview

Note that the **Shadeless** item is not actually mandatory for the baking, but is only required to see the object in the **Rendered** viewport shading mode as in the previous screenshot.

5. Also, enable the **4th** scene layer (*Shift*+left-click) and in the **Outliner** window, select the **Gidiosaurus_for_baking** object. Go to the **Object Data** window to be sure that the **UVMap_norm** layer is the active one and then go to the **Render** window and scroll down to the bottom, to the **Bake** subpanel; click on the **Bake Mode** slot to select the **Textures** item from the pop-up menu:

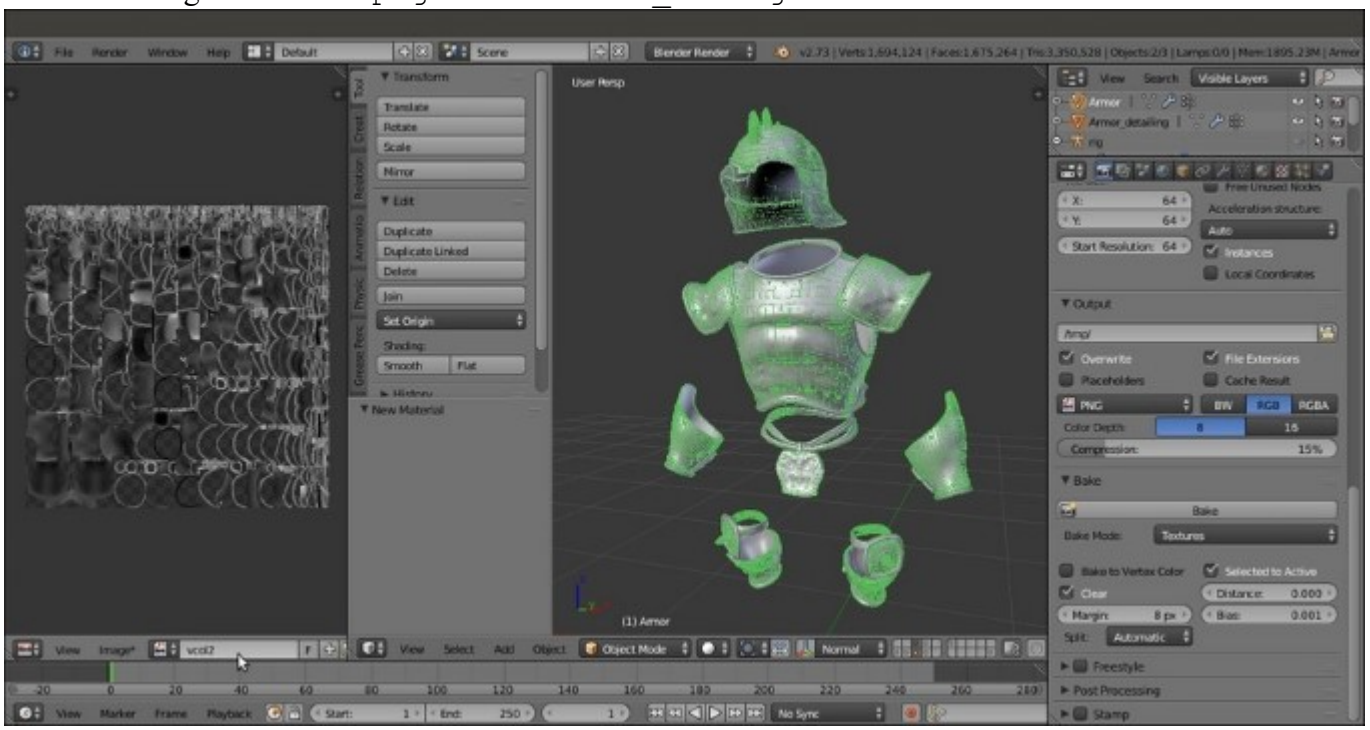


Baking the Dirty Vertex Colors layer to Textures

6. With the **Gidiosaurus_for_baking** object still selected, enter **Edit Mode** and select all the vertices; in the **UV/Image Editor** window, add a new **3072** x **3072** blank image renamed as `vcol`.
7. Go out of **Edit Mode** and in the **Outliner**, select the **Gidiosaurus_detailing** object; then, *Shift*-select the **Gidiosaurus_for_baking** object and go to the **Bake** subpanel under the **Render** window to click on the **Bake** button:

The final baked "vcol.png" image map

8. Save the baked image as `vcol.png` into the `texture_making` folder.
9. Enable the **3rd** scene layer, select the **Armor_detailing** object, and repeat the procedure; save the baked image as `vcol2.png` in the `texture_making` folder and also save the file:



The baked vertex color layer for the Armor

How it works…

The **Vertex Colors** tool can add a color to each vertex of the mesh, so it's actually possible to paint an object without the need for an image texture; the denser the mesh, the better this works.

The **Dirty Vertex Colors** tool uses the proximity and the depth of folds and creases on the mesh surface to calculate grayscale values to be assigned to the vertices; thanks to the **Vertex Color Paint** item, enabled in the **Material** window, this grayscale shows up in the rendering and so it's also possible to bake it into an image.

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Materials/Special_Effects/Vertex_Paint
- http://www.blender.org/manual/render/blender_render/materials/special_effects/vertex_paint.html

The Quick Edit tool

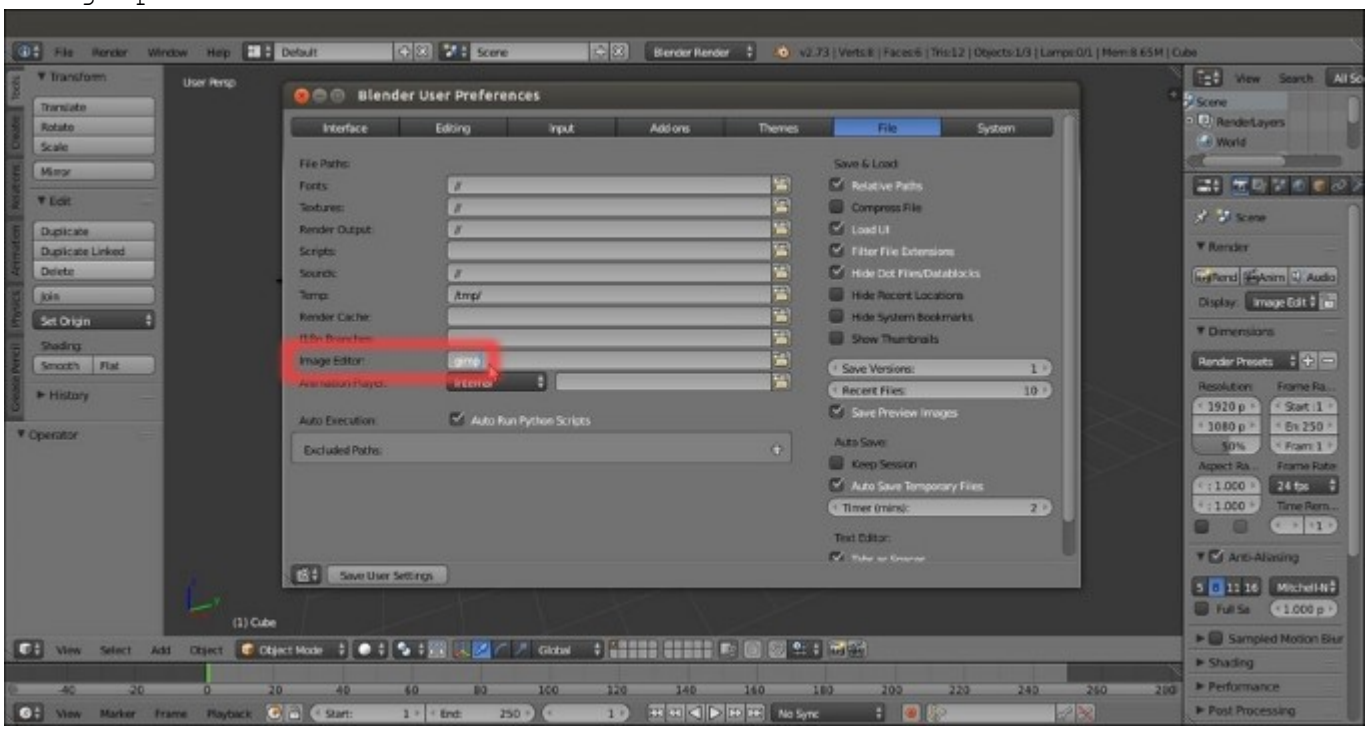
It's time to talk a bit about a very useful Blender tool: the **Quick Edit** tool.

Through this tool, it's possible to export a screenshot of the model in our favorite 2D painting software (**Gimp** or **Photoshop**, or whatever), paint on it using a new alpha background layer, and reassign the painted layer to the model in Blender, which is UV-mapped on the selected UV coordinates layer. All this, in just a few clicks.

Getting ready

In our case, we don't actually need to use this tool to refine the textures for the **Gidiosaurus**, so this recipe is going to be just an example. By the way, to fully understand how to use the tool, I suggest you to follow all the steps; just don't save the file at the end (or save it with a different name in a different directory if you want to keep it). So, carry on with the following:

1. Start Blender and call the **User Preferences** panel (*Ctrl* + *Alt* + *U*); go to the **File** tab and, in the **Image Editor** slot (*Path to an image editor*), write the path to your 2D image painting software installation (this is also done by clicking on the open/browse button at the right end of the slot). The path, of course, changes based on your OS; in my case, in **Linux Ubuntu**, it's enough to write `gimp`:



The Image Editor path in the File tab of the User Preferences panel under Linux Ubuntu

2. Click on the **Save User Settings** button at the bottom-left of the panel and close it.

How to do it…

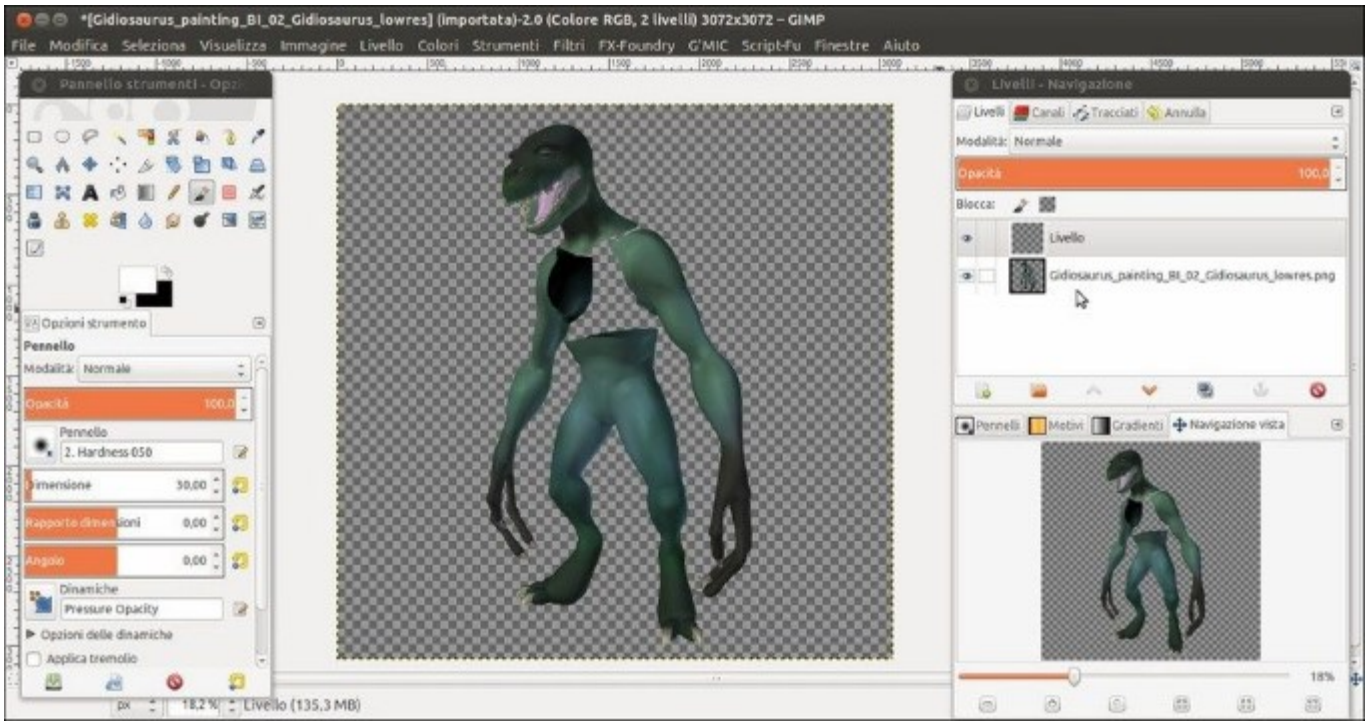
Once you've set the path to the image editor, let's load our **Gidiosaurus** file:

1. Load the `Gidiosaurus_painting_BI.blend` file and maximize it as much as possible in the 3D viewport.

   You can use both the **User** or the **Camera** view; it doesn't make any difference for the tool to work. By the way, it would be a good idea to use the **Camera** view so as to have a fixed point of view for any other case.
2. If necessary, press the *T* key to call the **Tool Shelf** panel; select the **Gidiosaurus** object and then go in **Texture Paint** mode.
3. Go to the **External** subpanel under the **Tools** tab; set a size for the screenshot to be exported (by default, it's **512** x **512** pixels; I set it to **3072** x **3072** pixels) and then click on the **Quick Edit** button:

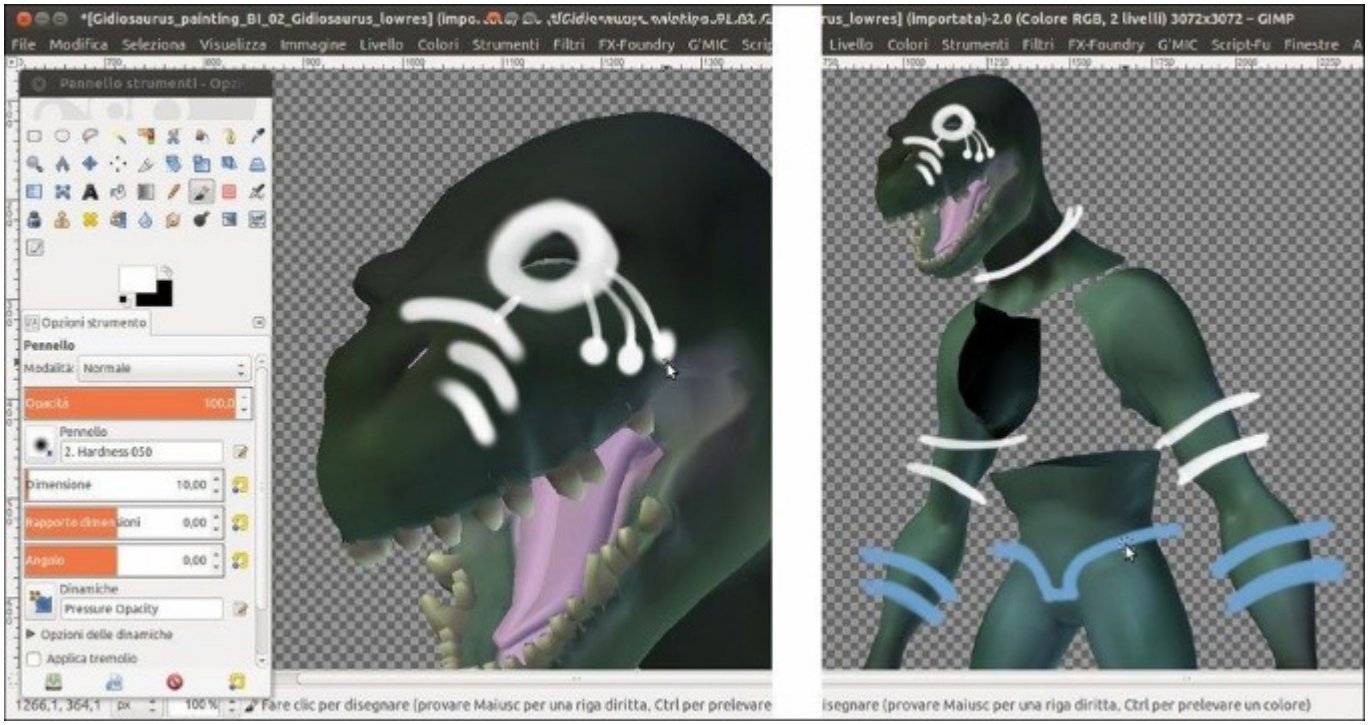
The External Image Editor subpanel

After a while, the image editor automatically starts (in my case, it's **Gimp 2.8**) and opens the screenshot of the model:



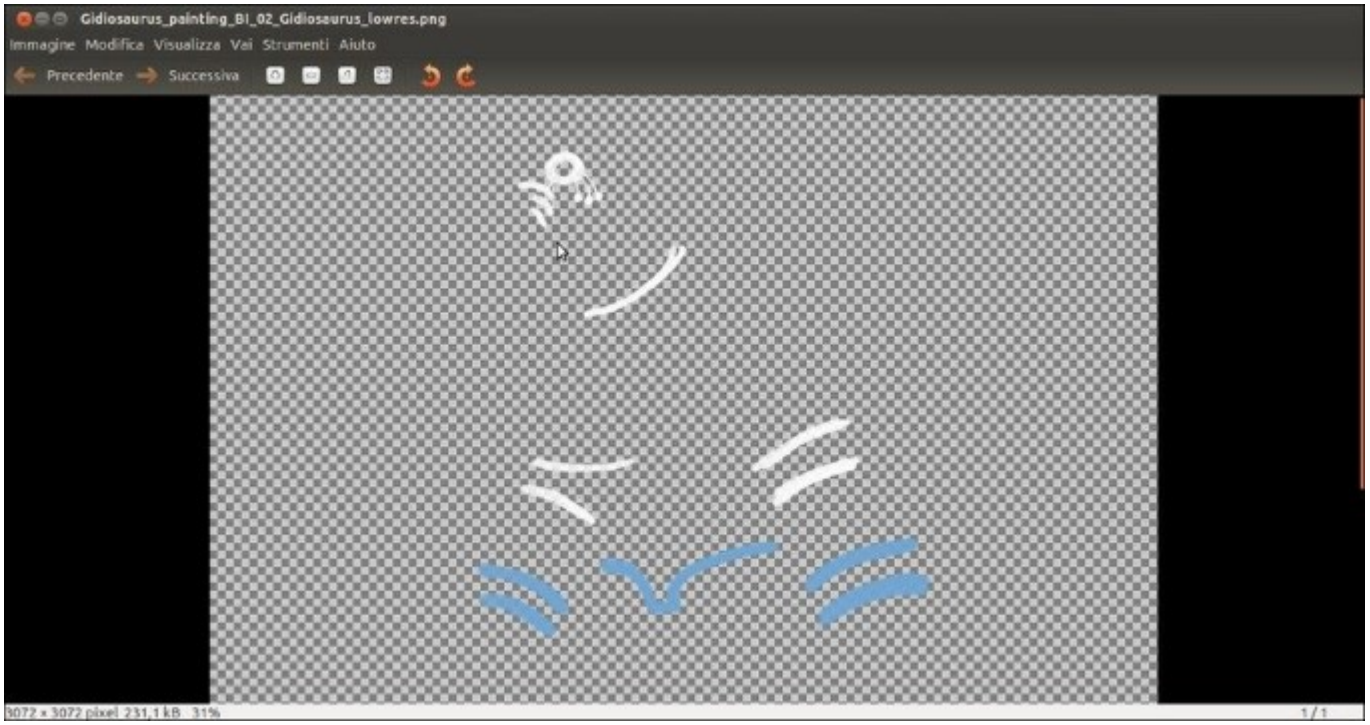
The screenshot previously visible in the Blender Camera view opened in Gimp

4. Add a new transparent layer and start to paint on it, adding some kind of tribal make-up decoration to the **Gidiosaurus**:



Tribal painting on the Gidiosaurus warrior

5. When you are done, deselect the visibility for the export layer and export the transparent painted one *by saving it with the same name as the exported one*. That is, the **Quick Edit** tool exported the screenshot by saving a `.png` image inside the blend file directory with the name `Gidiosaurus_painting_BI_02_Gidiosaurus_lowres.png`; export the painted layer by saving it as `Gidiosaurus_painting_BI_02_Gidiosaurus_lowres.png` as well:



The Gimp layer with the tribal painting "a solo"

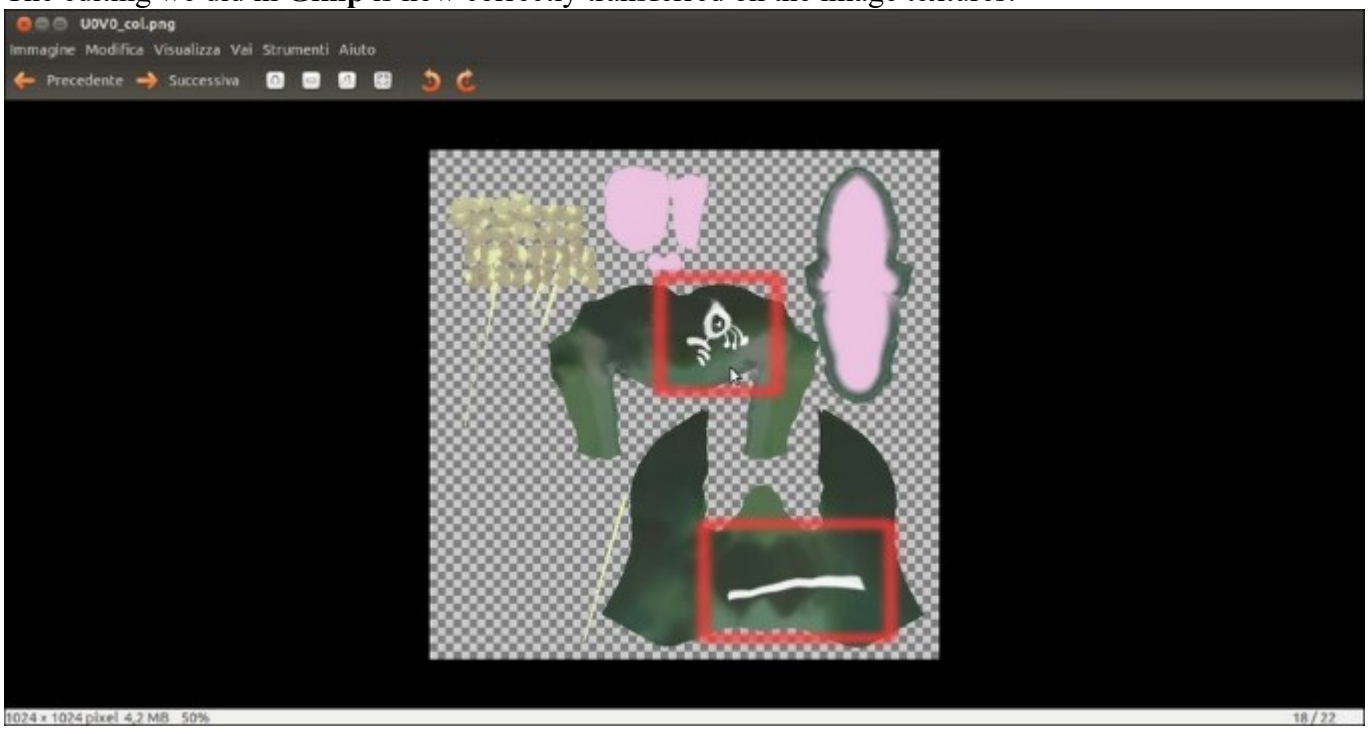
This is necessary for Blender to find it in the next step.

6. Back in Blender, click on the **Apply** button under the **External** subpanel and watch the new layer added to the model in the 3D viewport:

7. To make the textures editing permanent, click on the **Save All Images** button, both under the **Slots** tab and the **Image** item on the **UV/Image Editor** window toolbar:

The tribal painting transferred on the 3D model

The editing we did in **Gimp** is now correctly transferred on the image textures:



The tribal painting transferred on the image map

How it works…

As you have seen, the **Quick Edit** tool worked like a charm on all the **5** different materials assigned to the **Gidiosaurus** model for the painting. Be careful that, at least at the moment, this doesn't seem to work with **nodes materials** (which we'll see in the next chapter).

Chapter 12. Creating the Materials in Cycles

In this chapter, we will cover the following recipes:

- Building the reptile skin shaders in Cycles
- Making a node group of the skin shader to reuse it
- Building the eyes' shaders in Cycles
- Building the armor shaders in Cycles

Introduction

In Chapter 10, *Creating the Textures*, and in Chapter 11, *Refining the Textures*, we have prepared all the necessary texture images for the **Gidiosaurus** skin and for the iron **Armor** (the creation process for some textures, specifically the two textures for the character's **eyes**, hasn't been described, but basically it's a process similar to what we have already seen).

In this chapter, we'll see how to use these textures and how to set up the materials for the **Gidiosaurus** and the **Armor** in the **Cycles Render** engine.

A rendered example of the Cycles' shader final result

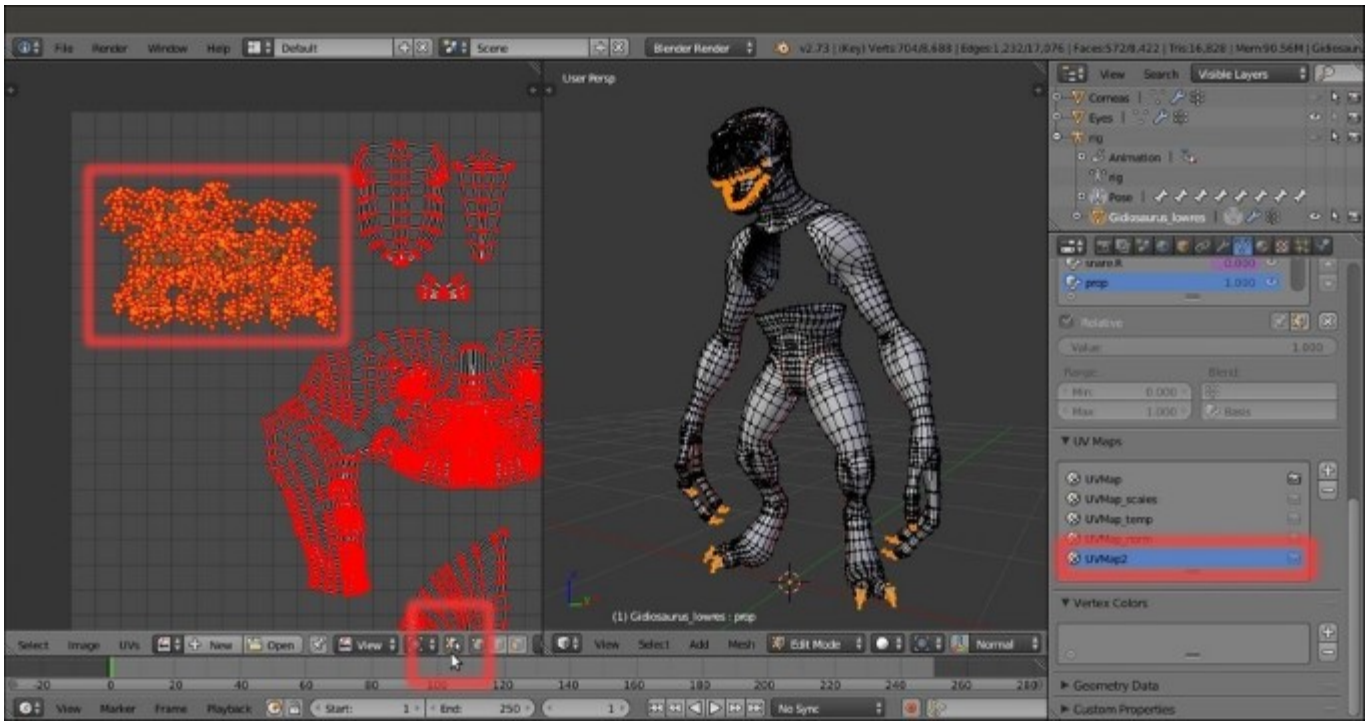
Building the reptile skin shaders in Cycles

So, let's start with the **Gidiosaurus** skin.

Getting ready

But first, as usual, we must prepare the file:

1. As the very first step, go to the `texture_making` folder and move the textures `vcol.png`, `vcol2.png`, `norm.png`, and `norm2.png` to the `textures` folder.
2. Then start Blender and open the `Gidiosaurus_baking_normals.blend` file we saved in Chapter 11, *Refining the Textures*.
3. Switch the left **UV/Image Editor** window with a **Node Editor** window and press the *N* key to get rid of the **Properties** sidebar. Put the mouse pointer in the 3D viewport to the right and press the *T* key to get rid of the **Tool Shelf** panel, then press the *Z* key twice to go in **Solid** viewport shading mode.
4. Enable the **3rd** scene layer, select and delete the **Armor_detailing** object (press the *X* key, then left-click to confirm).
5. Enable the **4th** scene layer and select and delete the **Gidiosaurus_for_baking** object as well. Enable the **14th** scene layer, and also select and delete the **Gidiosaurus_detailing** and the **enamels** objects.
6. Enable the **11th** scene layer and right-click on the **Gidiosaurus_lowres** object to select it.
7. It's not mandatory but, in case it is not already disabled, it is best to go to the **Object Modifiers** window and disable the **Armature** modifier visibility in the viewport by clicking on the eye icon button.
8. Go to the **UV Maps** subpanel under the **Object Data** window and select the **UVMap** coordinates layer (the first one); press *Tab* to enter **Edit Mode**, then click on the + icon button to the right side of the **UV Maps** subpanel to add a new coordinates layer; rename it **UVMap2**.
9. Go to the left window and change it into a **UV/Image Editor**; one by one, select the UV islands of the **talons** (at the moment these are placed inside the other **UDIM** tile spaces), and move them to the default **U0V0** tile, to overlap the location of the **teeth** islands. The reason for this will be clear later, when we will reuse the same color map both for the teeth and for all the talons.
10. If necessary, remember to disable the *Keep UV and edit mode mesh selection in sync* button on the **UV/Image Editor** toolbar:



Moving the talon islands to overlap the teeth islands inside the default U0V0 tile space

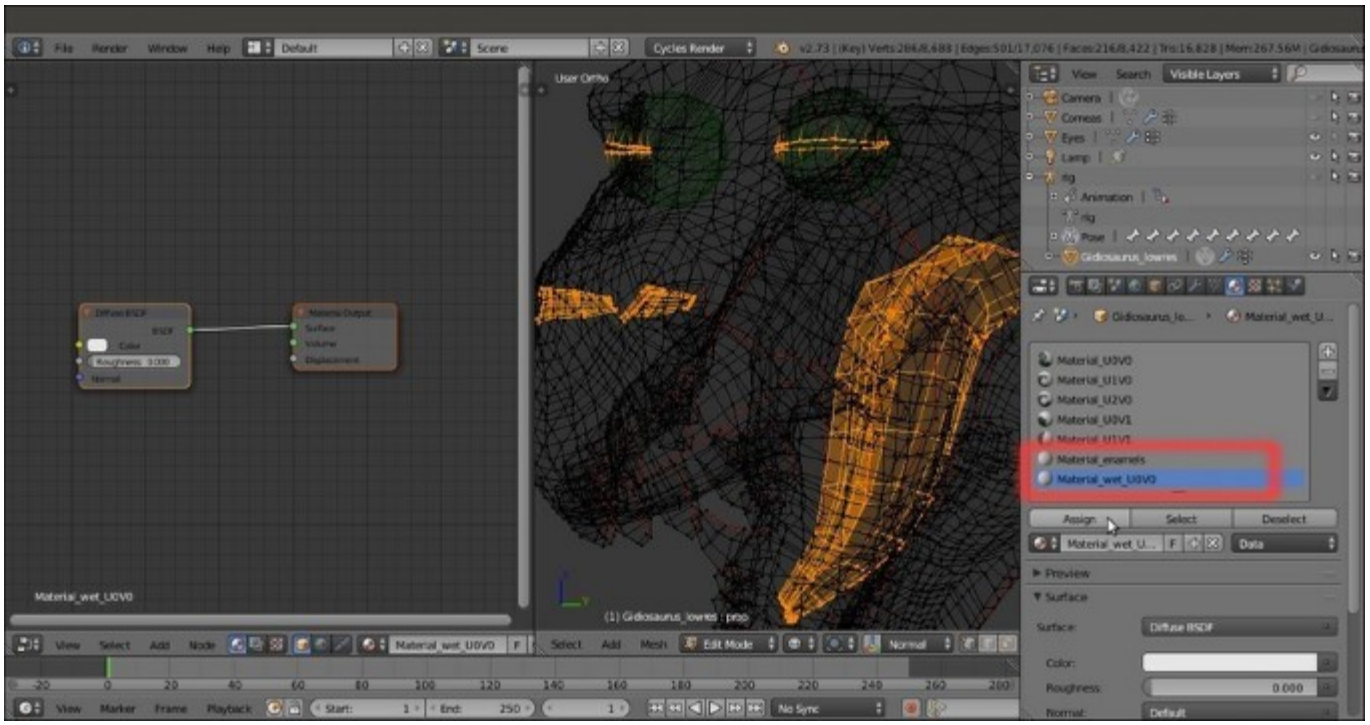
11. Go out of **Edit Mode** and switch the **UV/Image Editor** window back to a **Node Editor** window.
12. Click on the *Engine to use for rendering* slot on the main top header to switch to the **Cycles Render** engine.
13. Also, enable the **6th** scene layer to show the **Lamps**. Go to the **Outliner**, unhide and delete the **Lamp.001** object and select the **Lamp** object; in the **Object Data** window, change the type to **Spot** and then click on the **Use Nodes** button. Set the **Strength** to **10.000** and the **Color** to **R 1.000**, **G 1.000**, **B 0.650**, then set the **Size** to **0.500** and enable the **Multiple Importance** item (the **Multiple Importance Sampling** helps in reducing noise for big lamps and sharp glossy reflections, at the cost of the samples rendering a bit slower).
14. Put the mouse pointer in the 3D viewport and press *N* to call the **Properties** sidepanel; in the top **Transform** subpanel, type: **Location X** = **6.059204**, **Y** = **-9.912249** and **Z** = **7.546275**, and **Rotation X** = **55.788944°**, **Y** = **0°** and **Z** = **30.561825°**.
15. Go to the **Render** window and, under the **Sampling** subpanel, set the **Samples** to **400** for **Render** and **300** for **Preview**.
16. Go down to the **Light Paths** subpanel and disable the **Reflective Caustics** item, then set **Filter Glossy** to **1.000**.
17. Re-select the **Gidiosaurus_lowres** object and go to the **Material** window:

The 5 materials under the Cycles render engine

As you can see in the previous screenshot, the **Gidiosaurus_lowres** object has already assigned the **5** materials corresponding to the **5 UDIM** tile spaces (see Chapter 5, *Unwrapping the Low Resolution Mesh*, and Chapter 10, *Creating the Textures*).

The materials have been created under **Blender Internal** so, switching to **Cycles**, they show but aren't *initialized* as **node materials** yet; besides this, just before starting the creation of the first Cycles material, we must add **two** more materials.

18. Click the + icon button **twice** (*Add a new material slot*) to the right side of the **Material** window to add **two** new material slots.
19. Select the penultimate slot and click on the **New** button; rename the new material as `Material_enamels`. Select the last slot, click on the **New** button and rename it as `Material_wet_U0V0`.
20. Press *Tab* to enter **Edit Mode** and select all the vertices of the **teeth** and the **talons**; assign them to the `Material_enamels` slot.
21. Zoom on the **Gidiosaurus** head; select the vertices of the inner **nostrils**, of the inner edges of the **eyelids** and of the **tongue**, as shown in the following screenshot, and assign them to the `Material_wet_U0V0` slot:



Selecting the vertices of the "wet" areas of the character's head to assign them to the "Material_wet_U0V0" slot

22. Save the file as `Gidiosaurus_shaders_start.blend` file.

How to do it…

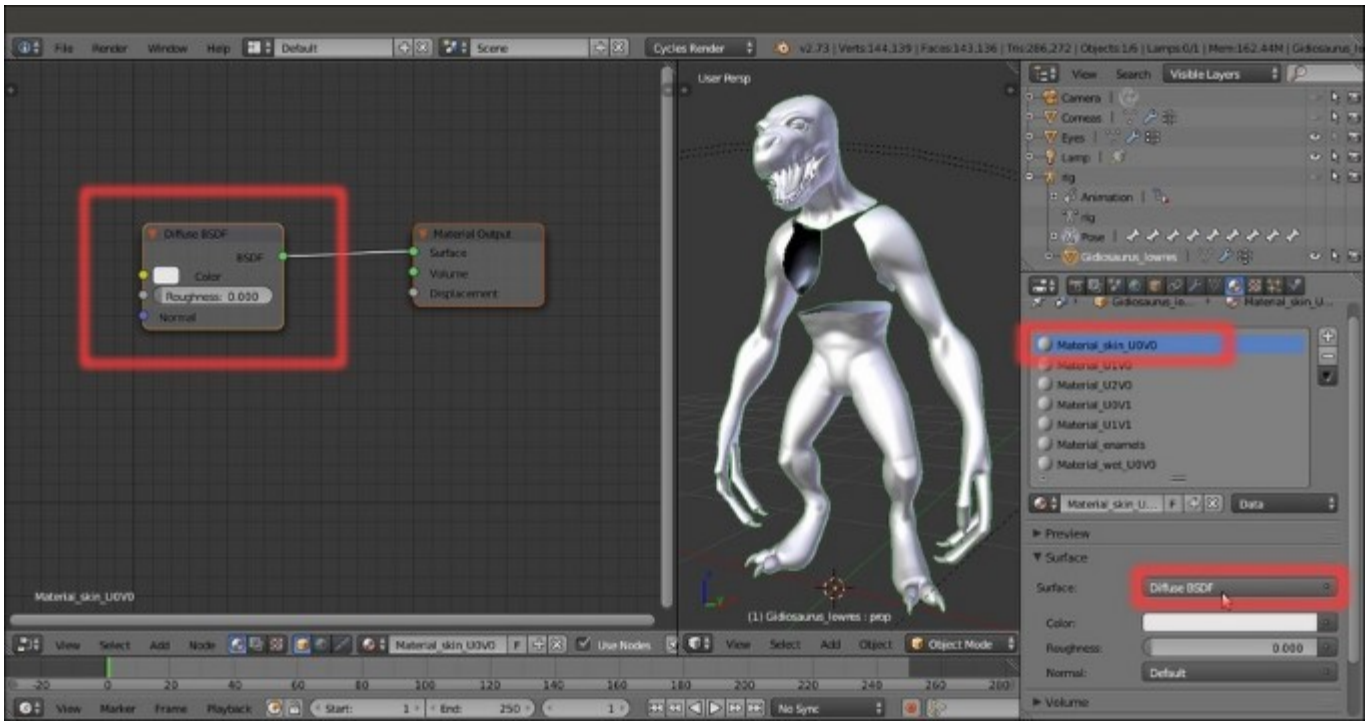
We know that the skin of our character is shared in **5** different materials; we are going to focus on the **head** (`Material_U0V0`), as the more representative one.

Once we are happy with the result, we will also copy (with all the due differences) the material to the other **body** parts.

Therefore, the steps are as follows:

1. In the materials list inside the **Material** window, select the `Material_U0V0` (the first top one) and press *Ctrl* + left-click on it to rename it as `Material_skin_U0V0`; then, move down and click on the **Use Nodes** button inside the **Surface** subpanel.

   Immediately, a **Diffuse BSDF** shader node (already connected to a **Material Output** node) appears inside the **Node Editor** window to the left of the screen and listed in the **Surface** slot inside the **Surface** subpanel to the right:

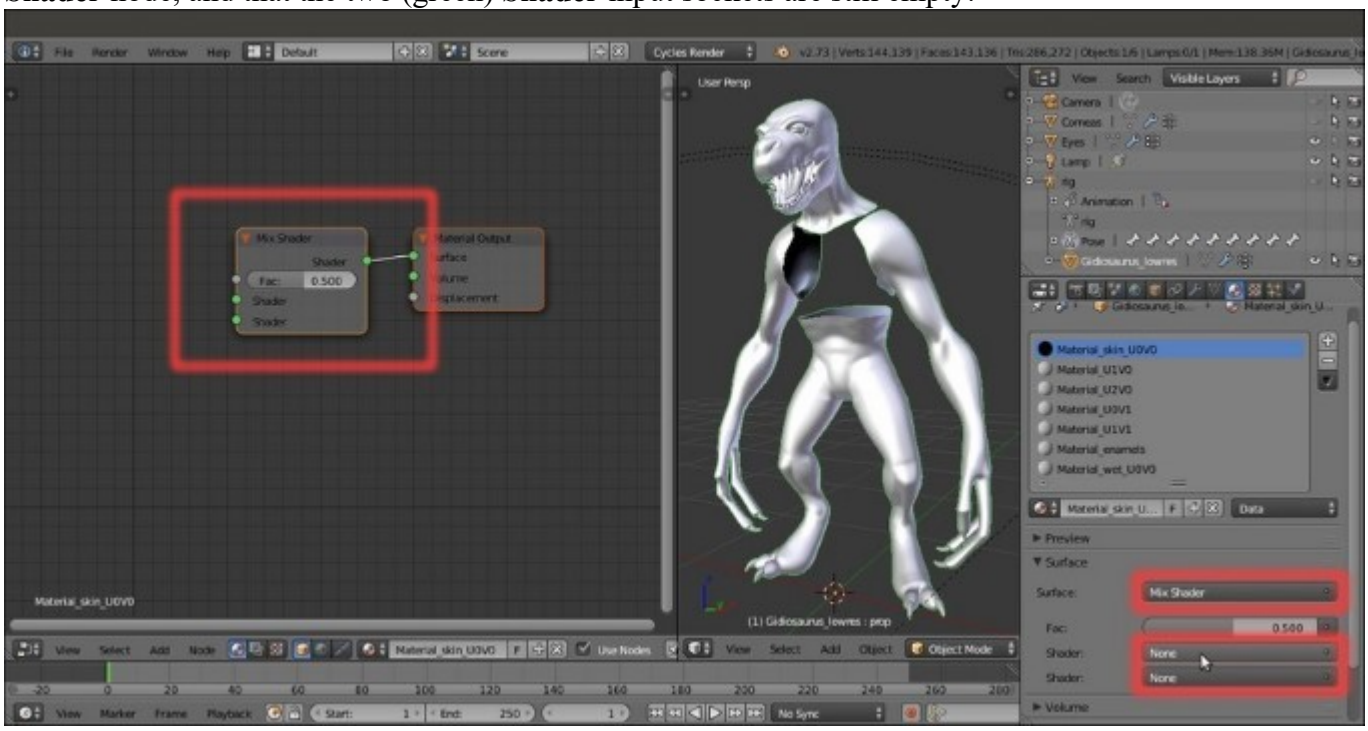


The Diffuse BSDF shader node connected to the Material Output node

2. In the **Surface** subpanel, under the **Material** window, click on the **Surface** slot that now shows the **Diffuse BSDF** shader: in the pop-up menu that appears, select a **Mix Shader** node:

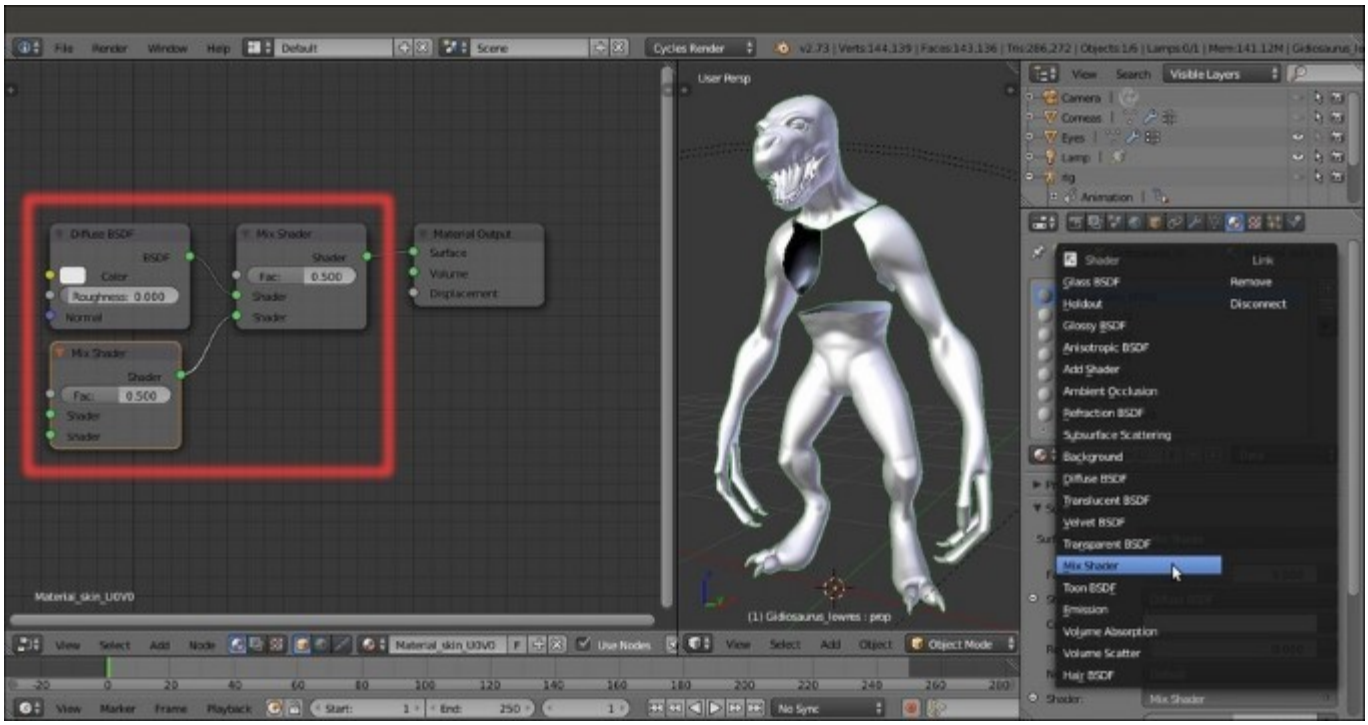
Switching the Diffuse BSDF shader node with a Mix Shader node through the Material window drop-down list

The **Surface** slot now shows the **Mix Shader** node item, and right below there are two new **Shader** slots that at the moment show the **None** item; in fact, looking at the nodes inside the **Node Editor** window, we see that the **Diffuse BSDF** shader node has been replaced by a **Mix Shader** node, and that the two (green) **Shader** input sockets are still empty:



The Mix Shader node with its two shader input sockets in the UV/Image Editor window and in the Material window

3. Click on the first **Shader** slot under the **Surface** subpanel to select, again from the pop-up menu, a **Diffuse BSDF** shader node; click on the second **Shader** slot and select a **Mix Shader** node; both the two new nodes are added and connected to the proper input socket, as we can see in the **Node Editor** window:



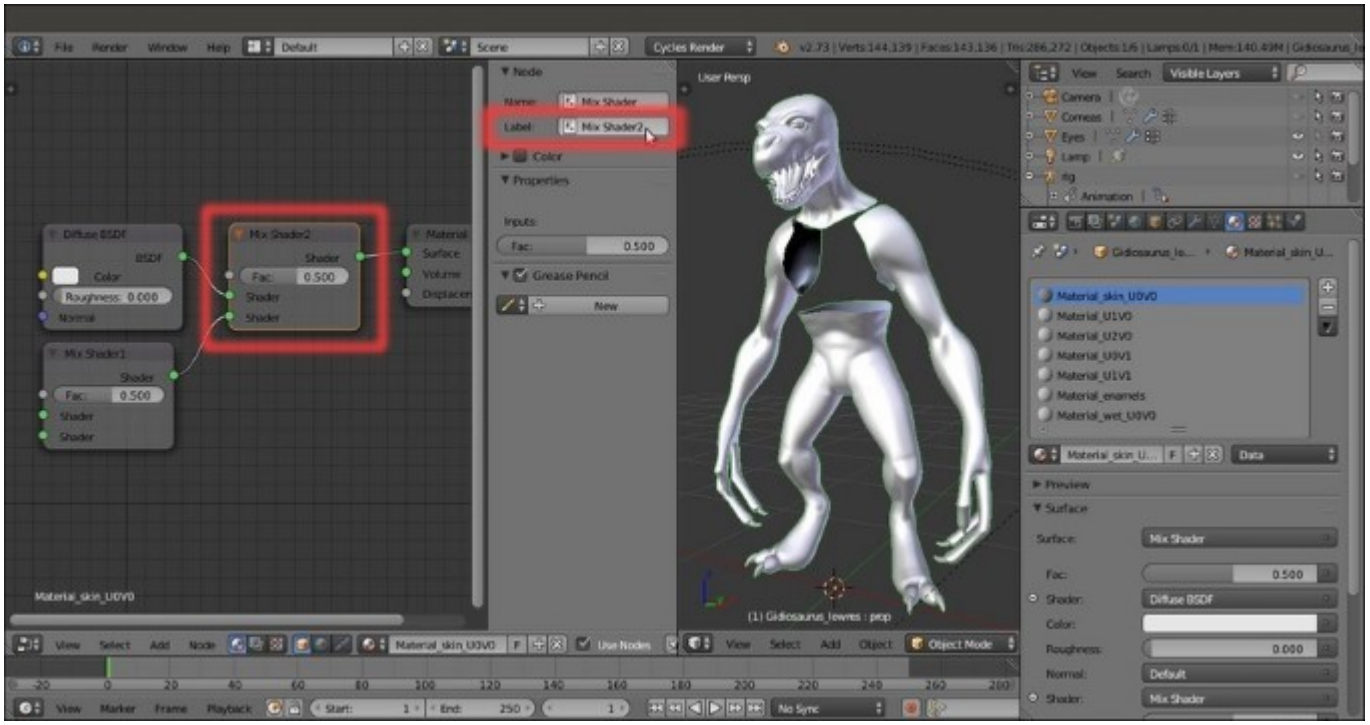
Two new nodes connected to the two shader input sockets of the Mix Shader node

At this point, to avoid confusion, it's already better to start to label the various nodes with meaningful names.

4. Put the mouse pointer inside the **Node Editor** window and press the *N* key to call the **Properties** sidepanel.
5. Select the last **Mix Shader** node we added to the material and then go to click on the **Label** slot inside the top **Name** subpanel of the side **Properties** panel: type **Mix Shader1**:

Labeling the nodes

6. Select the other **Mix Shader** node (the *old* one) and repeat the procedure by labeling it as **Mix Shader2**:

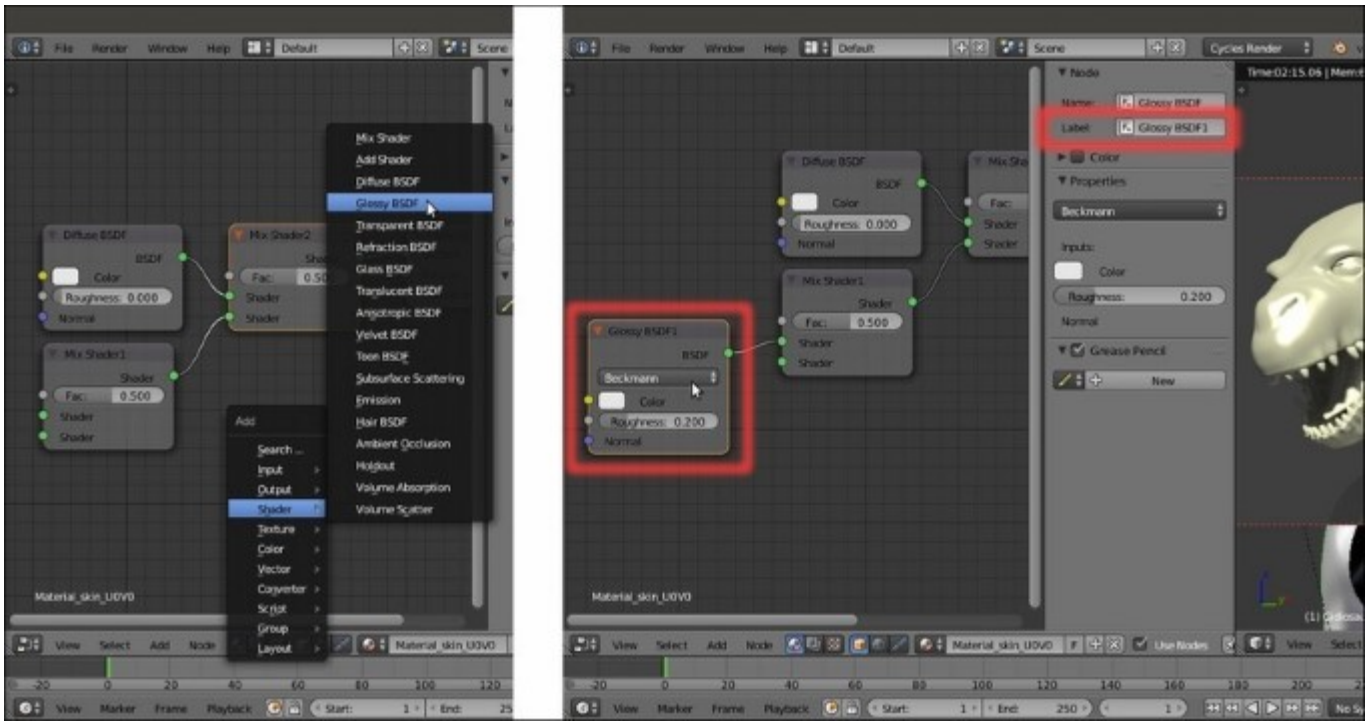


Labeling the nodes again

7. Put the mouse pointer on the 3D viewport and press the *0* key on the numpad to enter the **Camera** view.
8. Press *Shift* + *B* and by left-clicking draw a box around the **head** of the **Gidiosaurus** character to crop the area that can be rendered.
9. Zoom to the red square by scrolling the mouse wheel and then press *Shift* + *Z* to switch the **Viewport Shading** mode to **Rendered**:

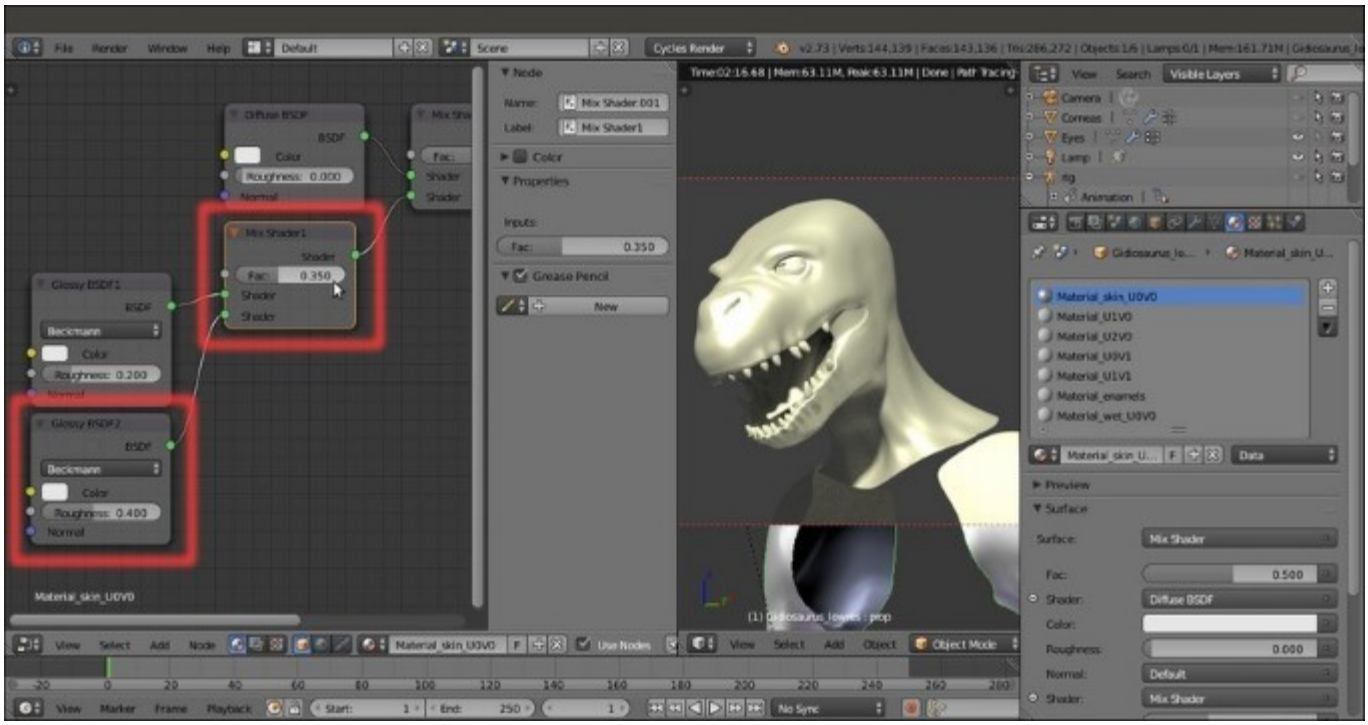
Cropping the renderable area and zooming to it

10. Put the mouse pointer inside the **Node Editor** window and press *Shift* + *A*. In the pop-up panel that appears, navigate to **Shader** and then click on the **Glossy BSDF** item to add the node; as it appears, move the mouse to place it to the left side of the **Mix Shader1** node.
11. Label it as **Glossy BSDF1**, connect its output to the first top **Shader** input socket of the **Mix Shader1** node, and set **Distribution** to **Beckmann**:



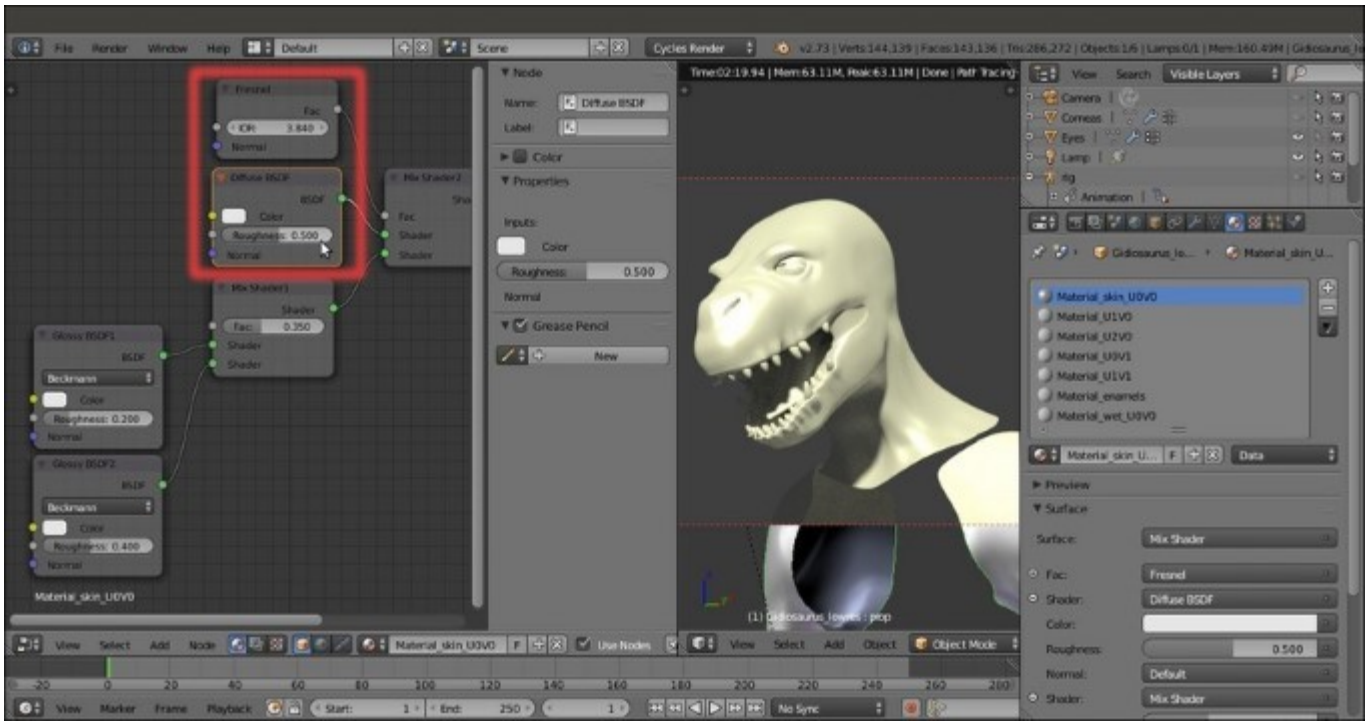
Adding a Glossy BSDF shader node and labeling it

12. Add a second **Glossy BSDF** shader node (*Shift + A* | **Shader | Glossy BSDF**) and place it right under the previous one; label it as **Glossy BSDF2**, connect its output to the second **Shader** input socket of the **Mix Shader1** node, and set **Distribution** to **Beckmann** as well and the **Roughness** to **0.400**.
13. Set the factor value (**Fac**) of the **Mix Shader1** node to **0.350**:



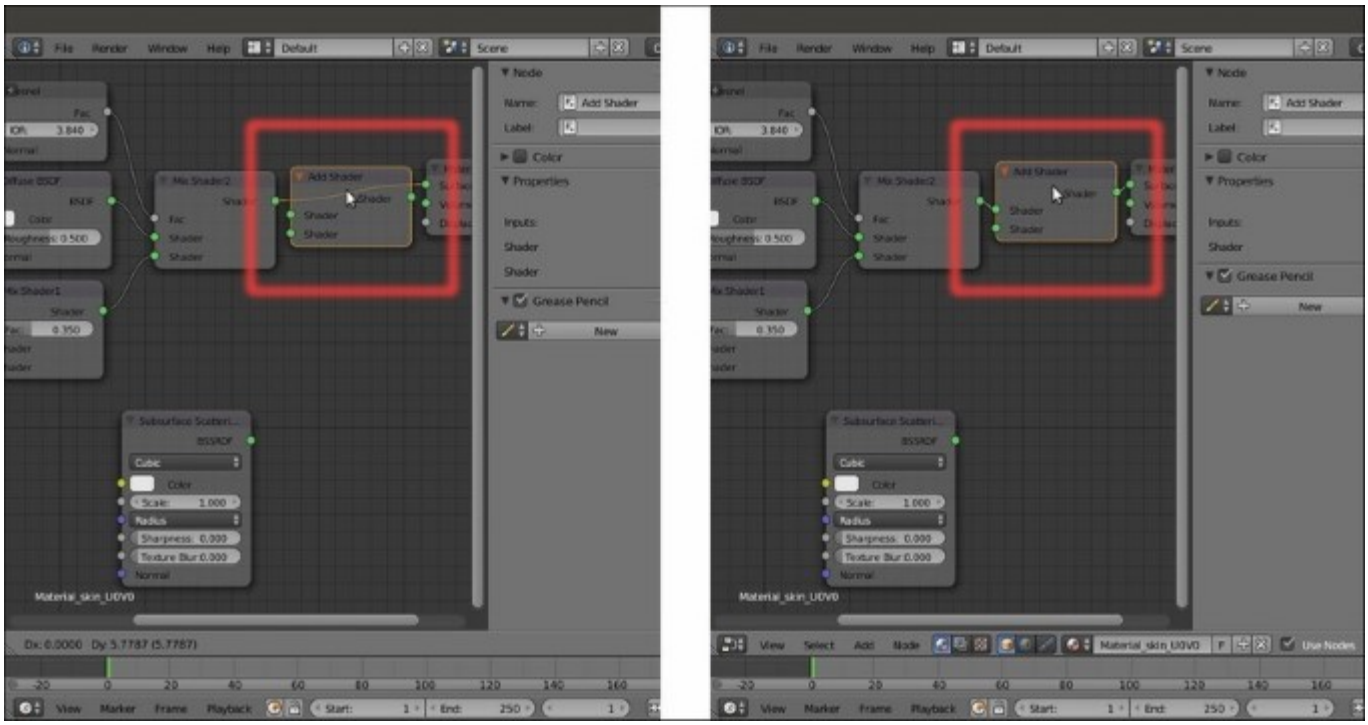
Adding a second Glossy shader node and blending it with the first one through the Fac value of the Mix Shader1 node

14. Add a **Fresnel** node (*Shift + A* | **Input** | **Fresnel**) and connect its **Fac** output to the **Fac** input socket of the **Mix Shader2** node; set the **IOR** value to **3.840**. Set the **Roughness** value of the **Diffuse BSDF** shader node to **0.500**:



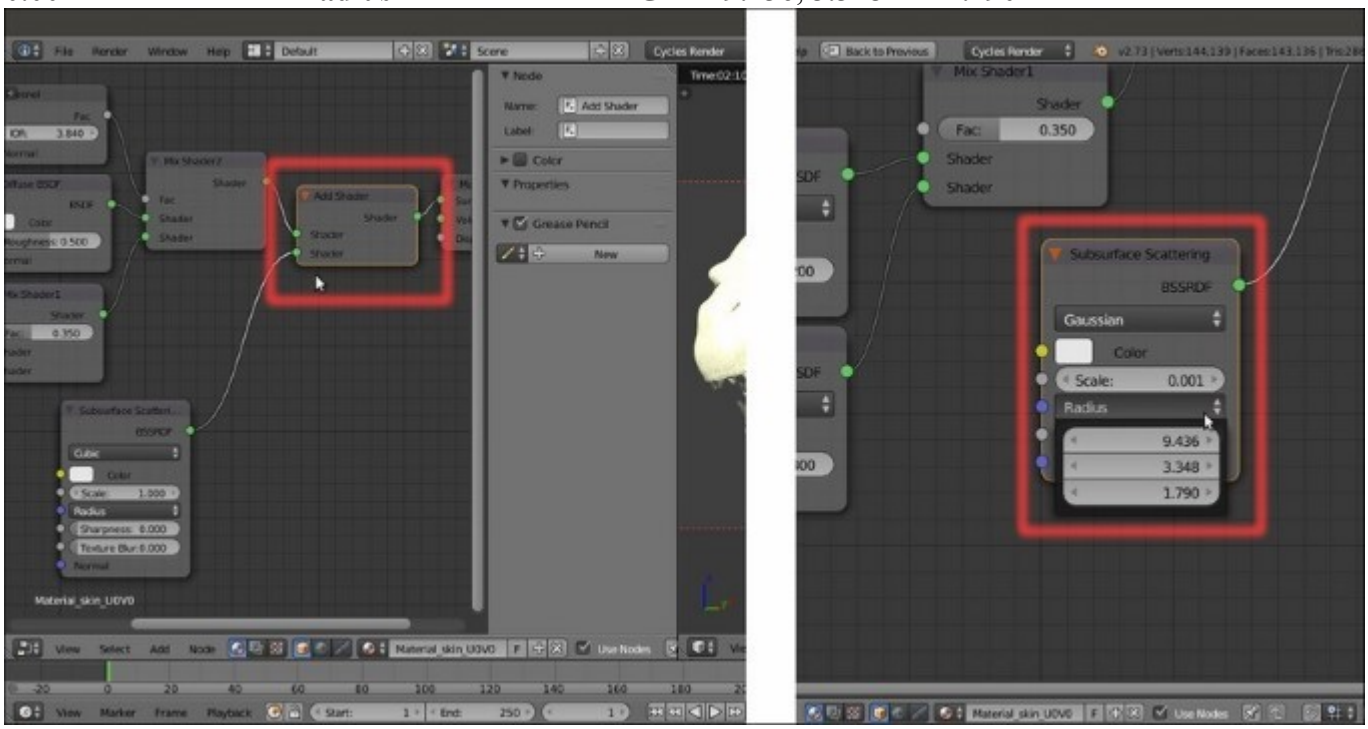
Adding a Fresnel node to set the Index of Refraction value to blend the diffuse with the glossy components

15. Add a **Subsurface Scattering** node (*Shift* + *A* | **Shader** | **Subsurface Scattering**) and an **Add Shader** node (*Shift* + *A* | **Shader** | **Add Shader**). Move this last one to the link that connects the **Mix Shader2** node to the **Material Output** node in order to paste it automatically between the two nodes (automatically when the connection line becomes highlighted):



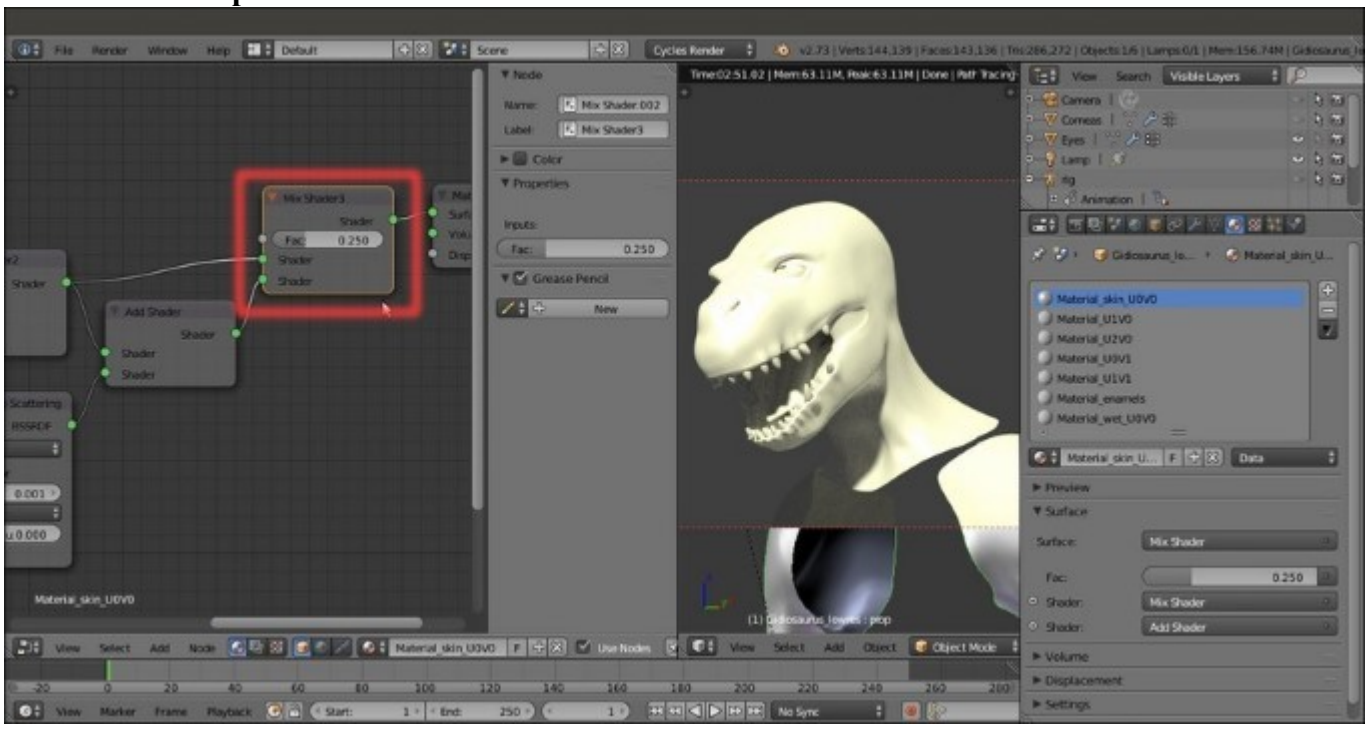
Automatically joining the Add Shader node

16. Connect the output of the **Subsurface Scattering** node to the second **Shader** input socket of the **Add Shader** node. In the **SSS** node, change **Fallof** from **Cubic** to **Gaussian**, set the **Scale** to **0.001** and click on the **Radius** button to set the **RGB** to **9.436**, **3.348** and **1.790**:



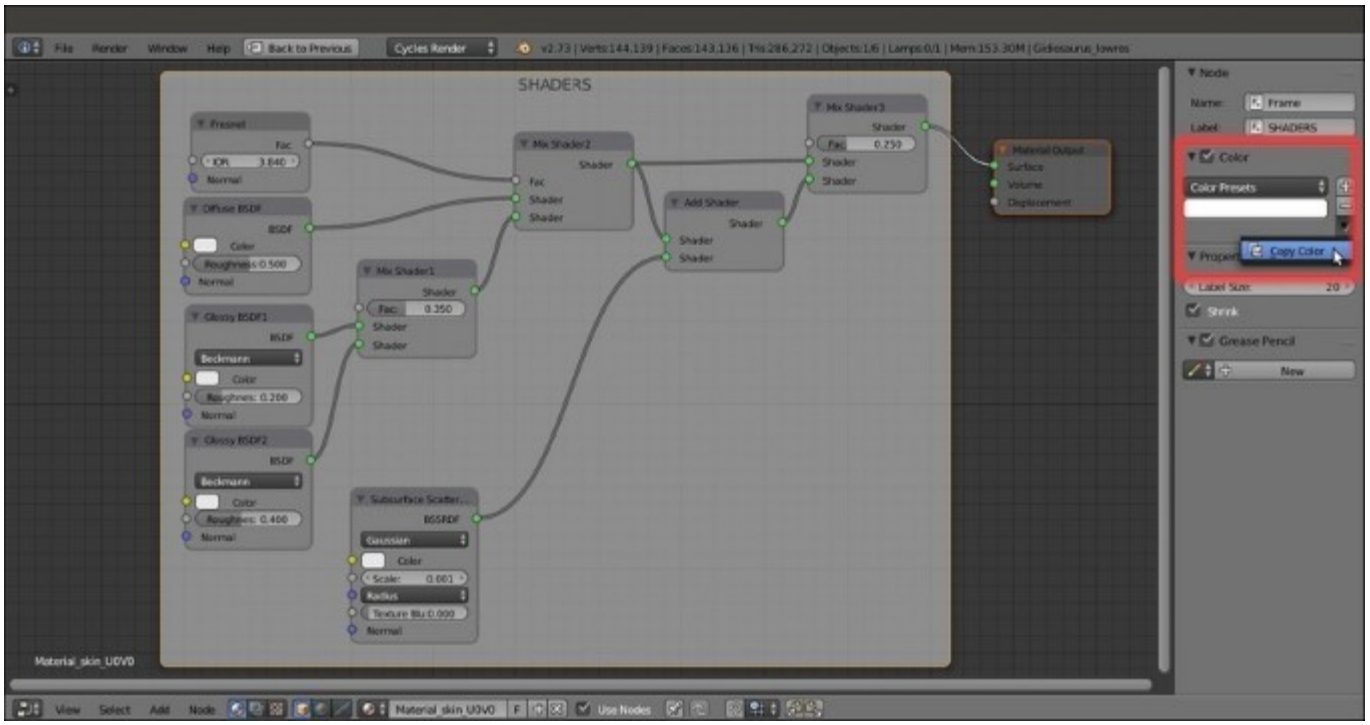
Connecting and setting the SSS node

17. Add a new **Mix Shader** node (*Shift* + *A* | **Shader** | **Mix Shader**) and label it as **Mix Shader3**. Connect the output of the **Mix Shader2** node to the first **Shader** input socket of the **Mix Shader3** node, and the output of the **Add Shader** node to its second **Shader** input socket. Set the **Fac** of the **Mix Shader3** node to **0.250** and connect its output to the **Surface** input socket of the **Material Output** node:



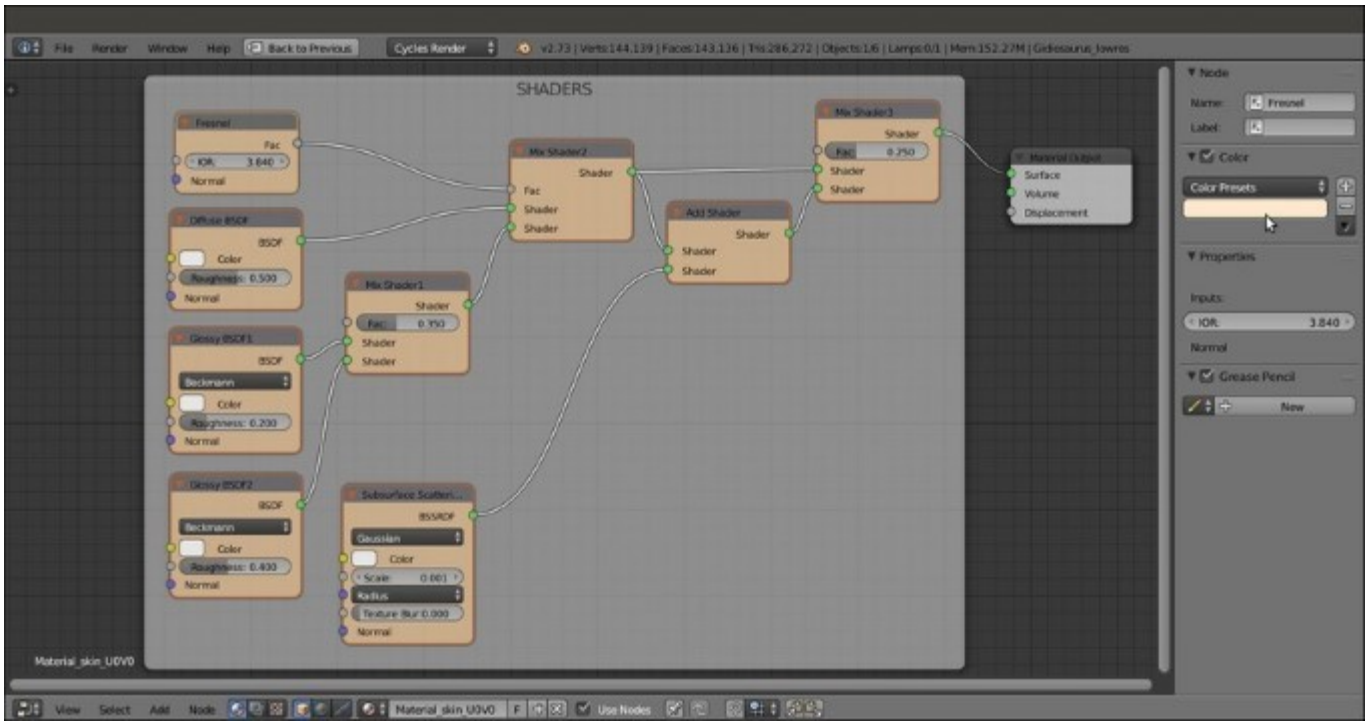
A little trick to tweak the influence of the Add shader node

18. Add a **Frame** (*Shift* + *A* | **Layout** | **Frame**), box-select all the nodes (except the **Material Output** node) and then press *Ctrl* + *P* to parent them to the frame; label the frame as **SHADERS**.
19. Select the **SHADERS** frame and go to the **Properties** sidepanel. Expand the **Color** subpanel (right under the **Node** subpanel) by clicking on the little horizontal black arrow, and enable the **Color** checkbox.
20. Click on the color slot and set a light color of your choice (I set it to **RGB 1.000**, which is totally white). Then click on the **+** icon button to the side and in the **Name** slot of the **Add Node Color Preset** pop-up panel, write **Frame**, then click the big **OK** button.
21. Select the **Material Output** node and then *Shift*-select the **Frame** again, then go to the **Color** subpanel and click on the big vertical arrow under the **+** and **–** icon buttons to the side. Click on the **Copy Color** item to copy the color of the **Frame** to the **Material Output** node:



The SHADERS frame with the nodes and the Copy Color tool under the N sidepanel

22. Select any one of the other nodes, for example the **Fresnel** node, enable the **Color** checkbox and set a new color of your choice (for these nodes, I set it to **R 1.000, G 0.819**, **B 0.617**, which is a light brown).
23. Click on the **+** icon button to the side and in the **Name** slot of the **Add Node Color Preset** pop-up panel, write **Shaders**, then click the big **OK** button.
24. Now box-select all the other nodes inside the frame and click on the **Copy Color** item to copy the color from the **Fresnel** node to all the other selected nodes at once:



Copying the label color from one node to all the other selected nodes

At this point we have completed the basic shader for the skin; what we have to do now is to add the textures we painted in both Chapter 10, *Creating the Textures*, and Chapter 11, *Refining the Textures*.

So:

25. Put the mouse pointer into the **Node Editor** window and add an **Image Texture** node (*Shift* + *A* | **Texture** | **Image Texture**); label it as **COL** and then use *Shift* + *D* to duplicate it; move the duplicated one down and change its label to **SCALES**.

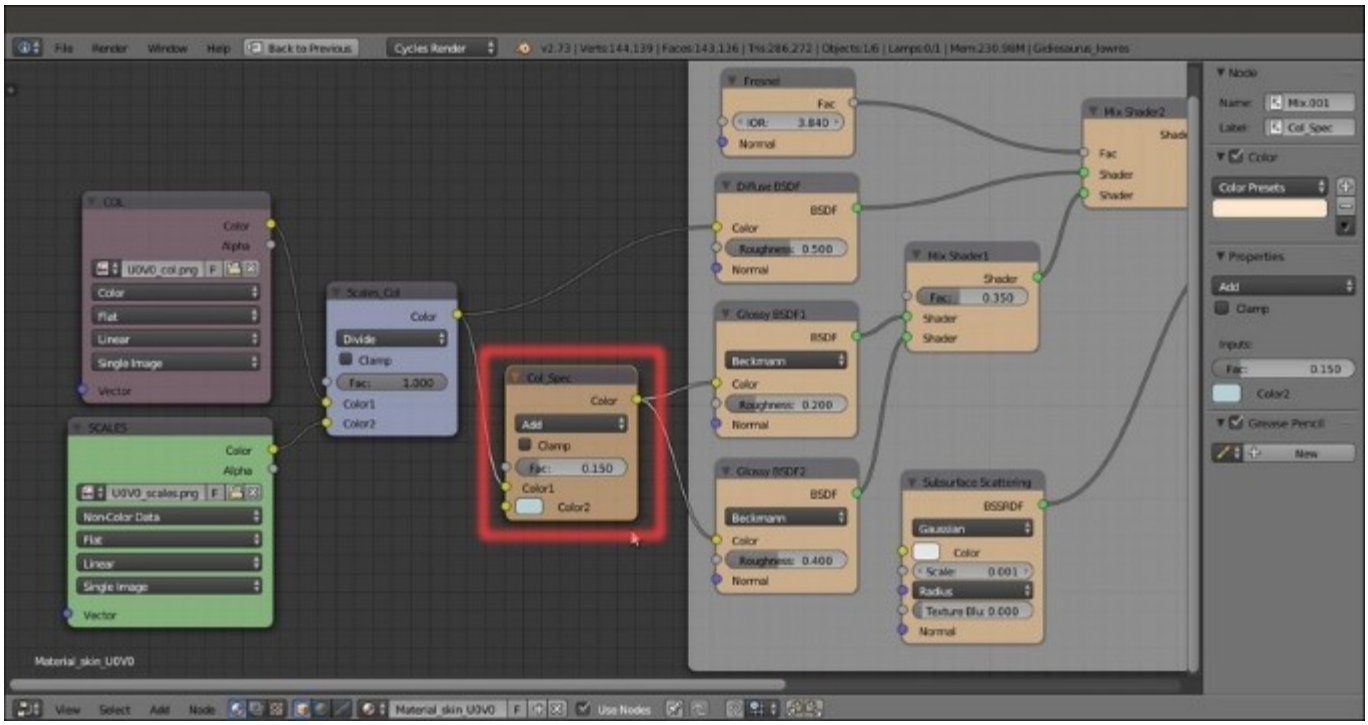
    As you label the newly added nodes, also assign colors to them to make them more easily readable inside the **Node Editor** window, and save these colors as presets as we did at step 20.
26. Click on the **Open** button of the **COL** node and browse to the `textures` folder. There, load the image `U0V0_col.png`.
27. Click on the **Open** button of the **SCALES** node and browse to the `textures` folder. There, load the image `U0V0_scales.png`; set the **Color Space** to **Non-Color Data**.
28. Add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and label it as **Scales_Col**; connect the **Color** output of the **COL** node to the **Color1** input socket of the **Scales_Col** node and the **Color** output of the **SCALES** node to its **Color2** input socket. Set the **Fac** to **1.000** and the **Blend Type** to **Divide**.
29. Connect the output of the **Scales_Col** node to the **Color** input socket of the **Diffuse BSDF** shader node inside the **SHADERS** frame.

The result so far is visible in the real-time rendered preview to the right:

The rendered result of the two combined image texture nodes

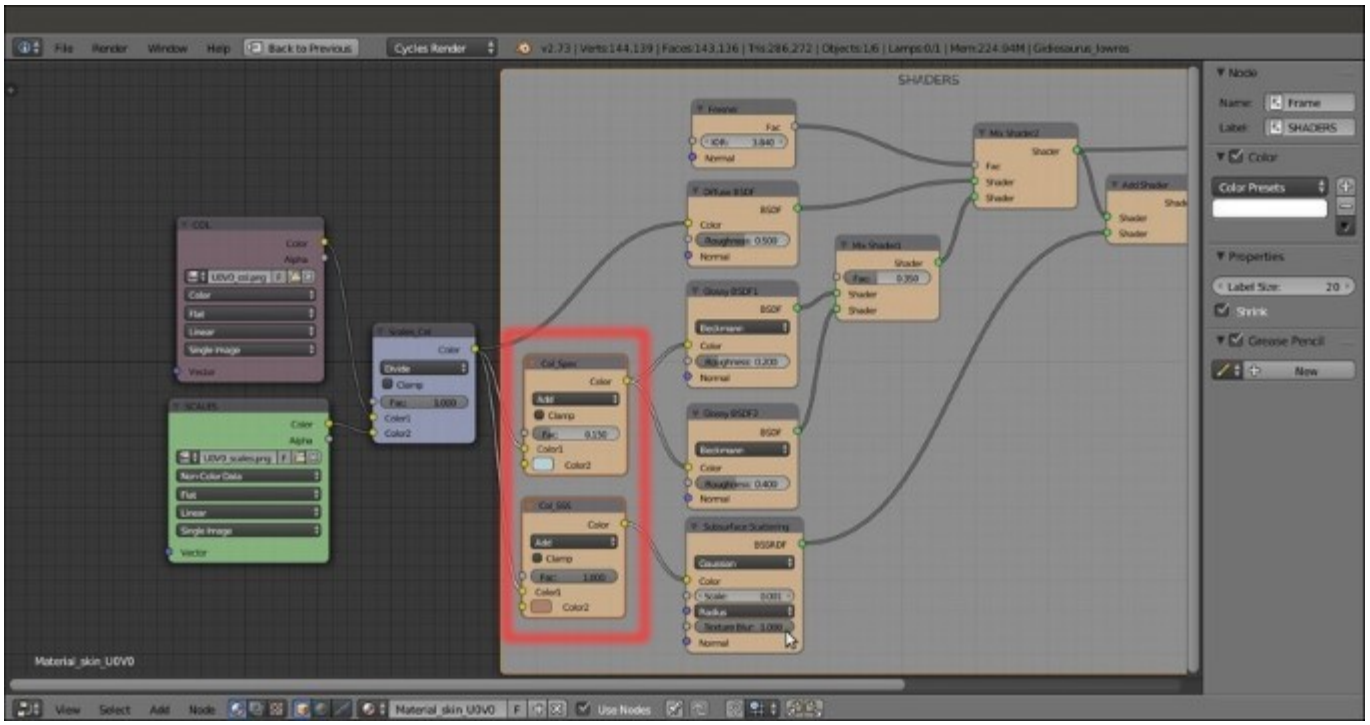
As you can see, the glossy component is strong in this one! We must lessen the effect, to obtain a more natural look.

30. Add a new **MixRGB** node (*Shift + A* | **Color** | **MixRGB**) and label it as **Col_Spec**; set the **Color2** to **R 0.474**, **G 0.642**, **B 0.683**, then also connect the output of the **Scales_Col** node to the **Color1** input socket of the **Col_Spec** node.
31. Set the **Fac** value to **0.150** and the **Blend Type** to **Add**, then connect its output to the **Color** input sockets of both the **Glossy BSDF1** and **Glossy BSDF2** nodes:



Varying the textures color output for the glossy component

32. Press *Shift* + *D* to duplicate the **Col_Spec** node and label the duplicate as **Col_SSS**; set the **Fac** value to **1.000** and the **Color2** to **R 0.439**, **G 0.216**, **B 0.141**. Connect the **Color** output of the **Scales_Col** node to the **Color1** input socket of the **Col_SSS** node and the output of this latter node to the **Color** input socket of the **Subsurface Scattering** node; increase its **Texture Blur** to the maximum value.
33. *Shift*-select the **Col_Spec** and the **Col_SSS** nodes and then also the **SHADERS** frame, and press *Ctrl* + *P* to parent them:

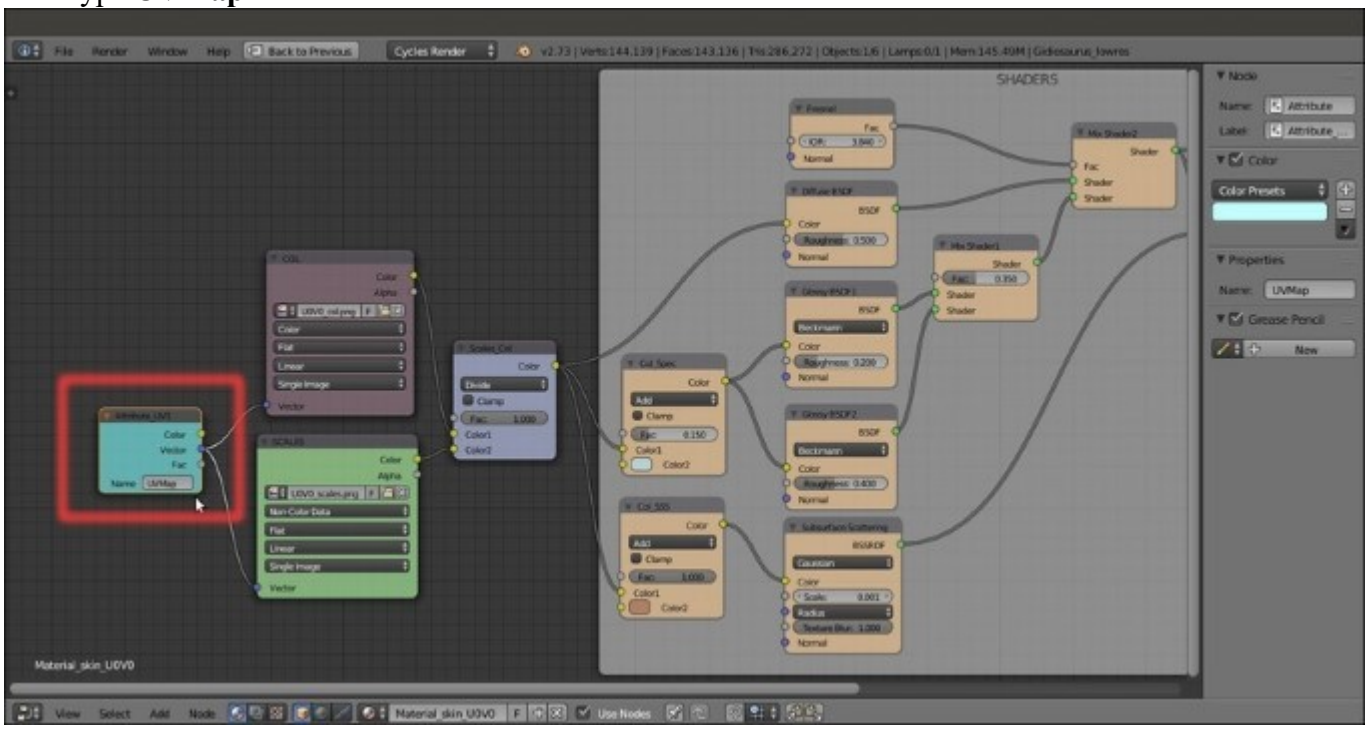


Varying the textures color output also for the SSS node

The new result looks a lot better:

A better result

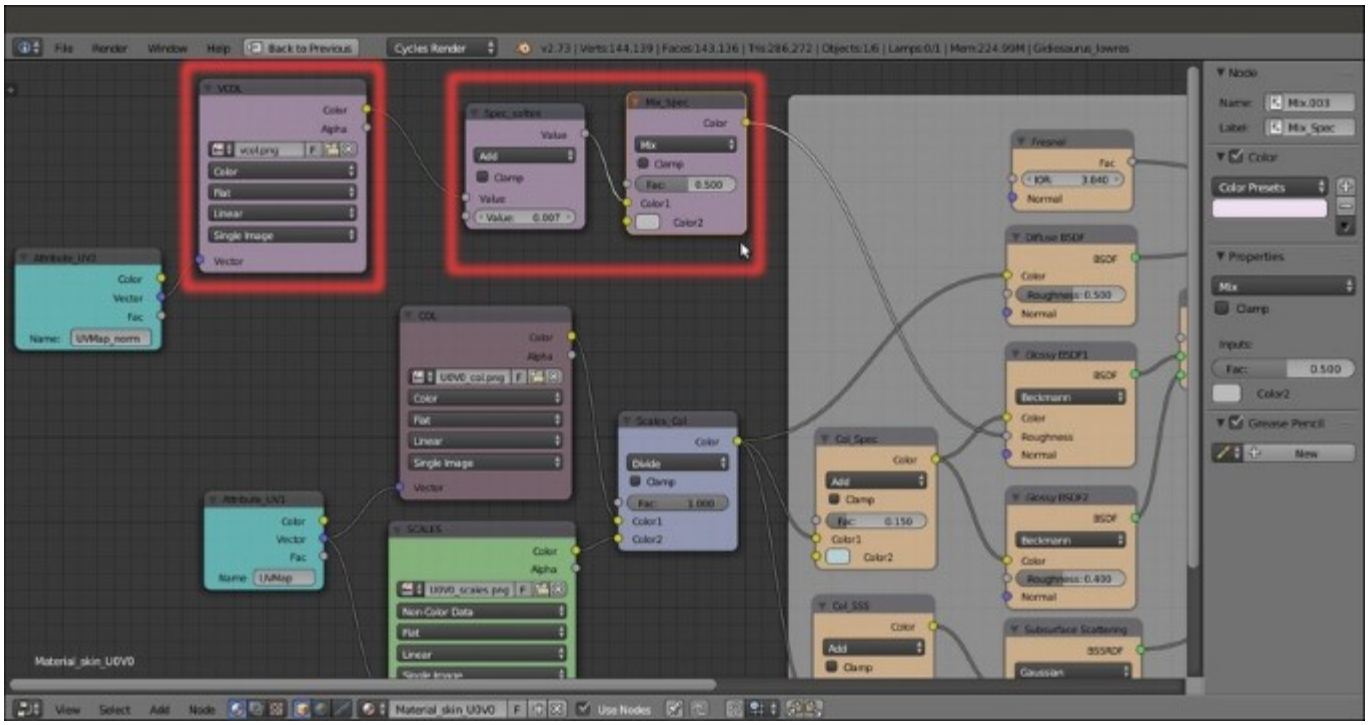
34. Add an **Attribute** node (*Shift* + *A* | **Input** | **Attribute**) and label it as **Attribute_UV1**. Connect its **Vector** output to the **Vector** input sockets of the **COL** and **SCALES** nodes and in the name field type **UVMap**:



Adding the Attribute node to establish the UV coordinates layer to be used

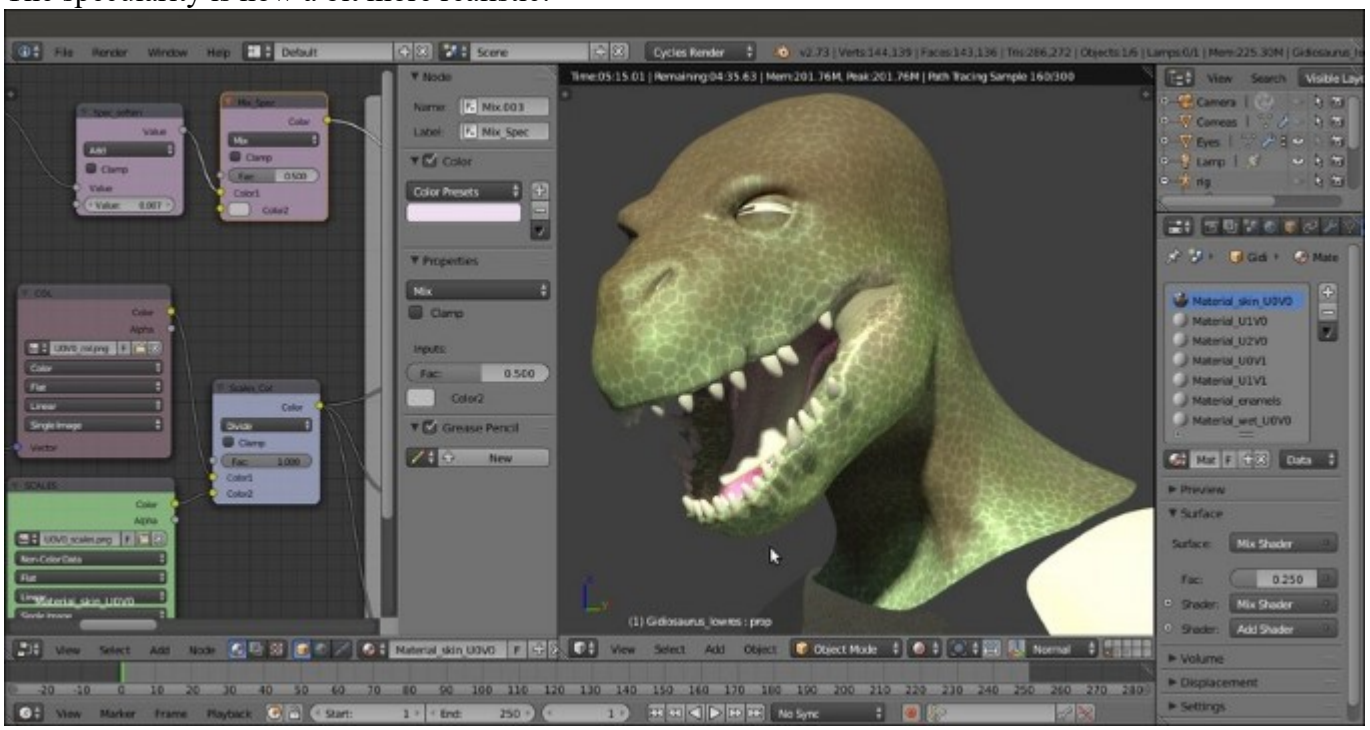
By the way, the glossy component is still a little unnatural.

35. Add a new **Image Texture** node (*Shift* + *A* | **Texture** | **Image Texture**) and label it as **VCOL**. Click on the **Open** button, browse to the `texture` folder and load the image `vcol.png`.
36. Press *Shift* + *D* to duplicate the **Attribute** node, change the label to **Attribute_UV2**, and change the **Name** field to **UVMap_norm**. Connect its **Vector** output to the **Vector** input of the **VCOL** node.
37. Add a **Math** node (*Shift* + *A* | **Converter** | **Math**) and a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**); connect the **Color** output of the **VCOL** node to the first **Value** input socket of the **Math** node; label this one as **Spec_soften** and set the second **Value** to **0.007**. Connect its **Value** output to the **Color1** input socket of the **MixRGB** node, which is now labeled as **Mix_Spec**.
38. Connect the **Color** output of the **Mix_Spec** node to the **Roughness** input socket of the **Glossy BSDF1** node:



Using the baked Vertex Color image to "soften" the character's skin specularity

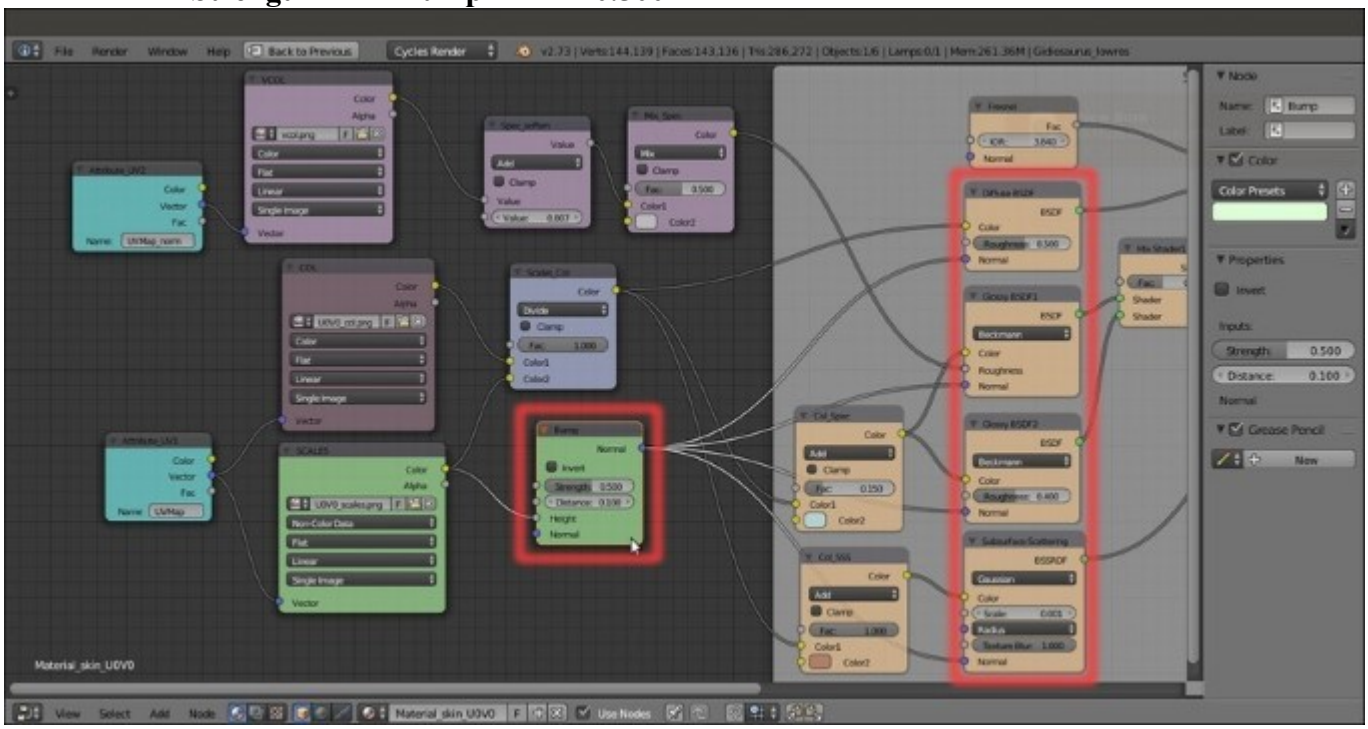
The specularity is now a bit more realistic:



And the rendered result of this operation

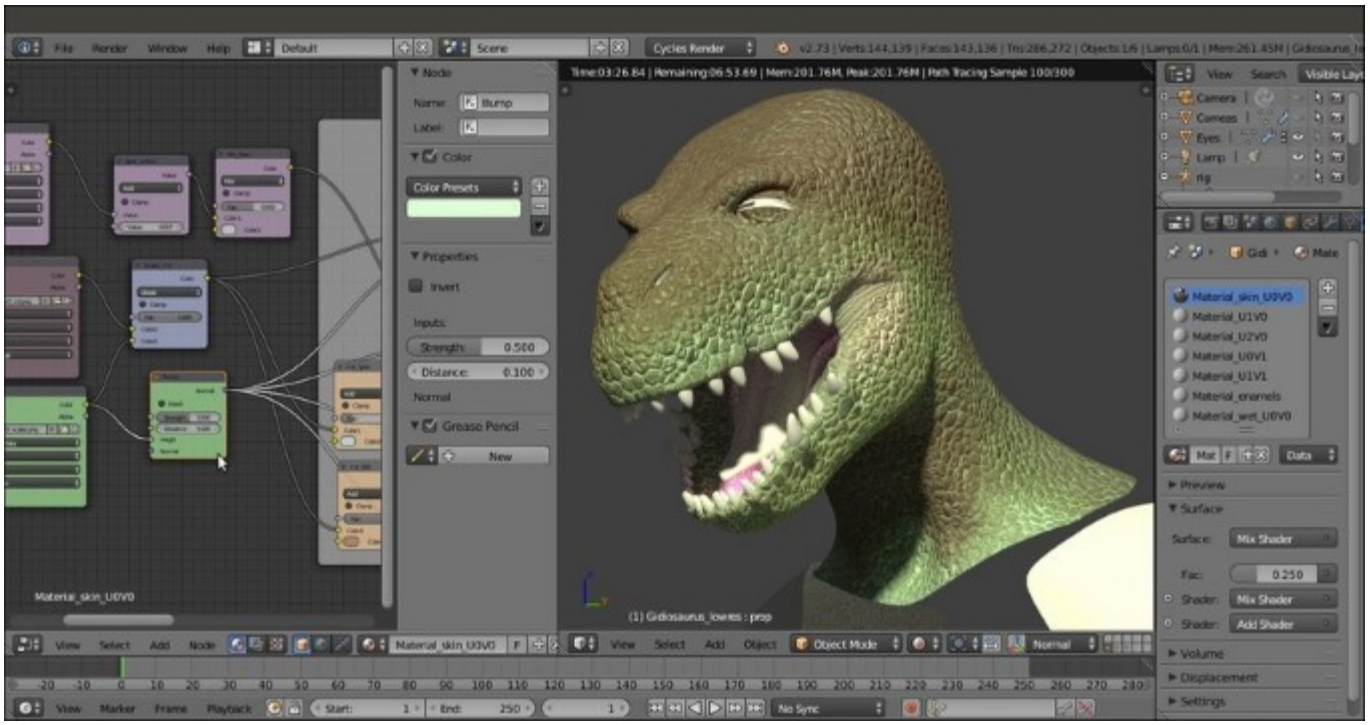
Anyway, it's still missing the contribution of the bump effect.

39. Add a **Bump** node (*Shift* + *A* | **Vector** | **Bump**); connect the output of the **SCALES** node to the **Height** input socket of the **Bump** node and the **Normal** output of this latter node to the **Normal** input socket of the **Diffuse BSDF**, **Glossy BSDF1**, **Glossy BSDF2**, and **Subsurface Scattering** nodes. Set the **Strength** of the **Bump** node to **0.500**:



Adding the bump pattern to the shaders

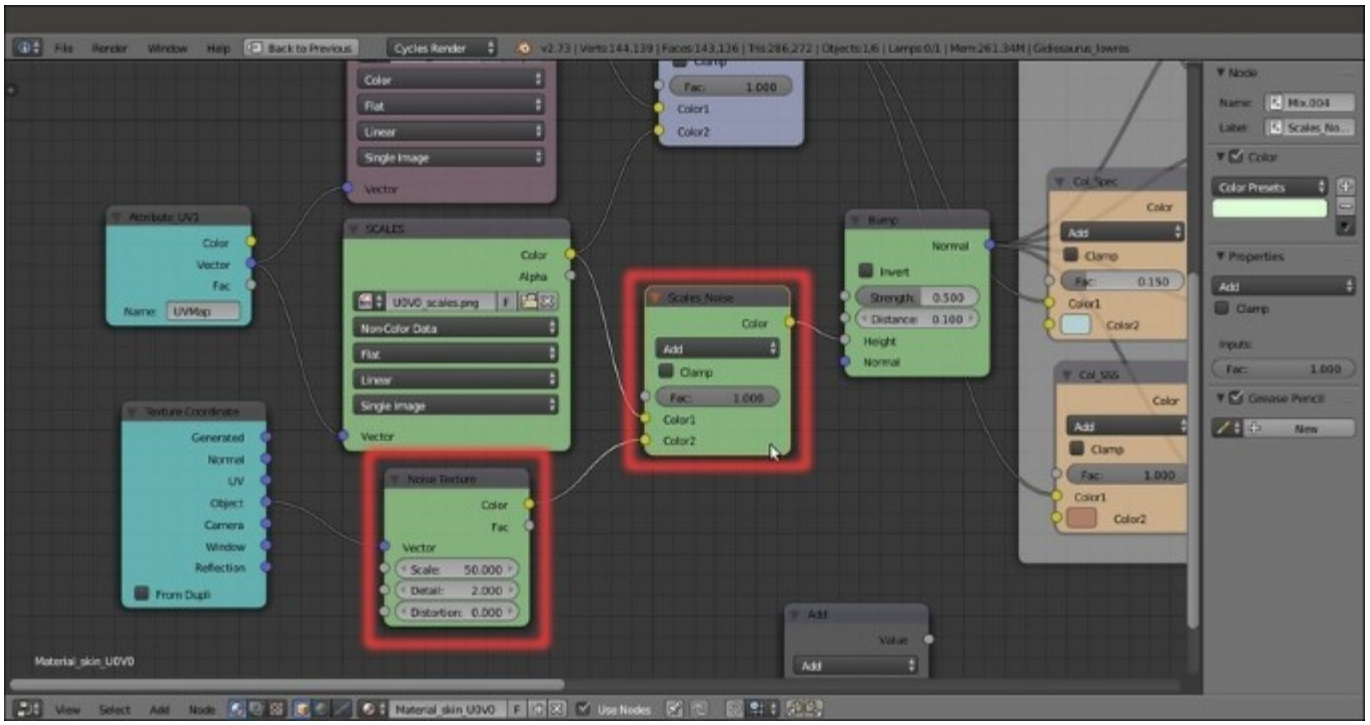
Now we start to see something!



The bump effect in the rendered preview

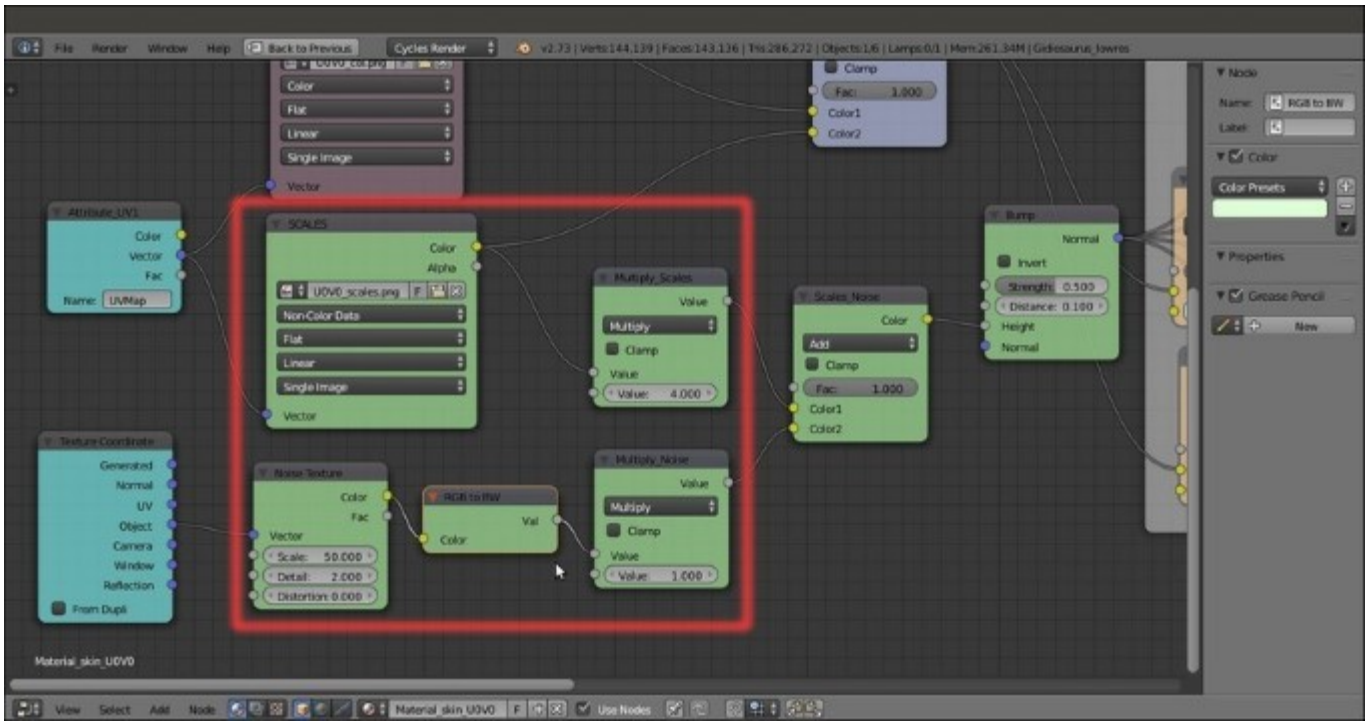
By the way, the bump pattern is too even and, therefore, unrealistic; we must therefore *break* it in some way.

40. Add a **Noise Texture** node (*Shift* + *A* | **Texture** | **Noise Texture**) and a **Texture Coordinate** node (*Shift* + *A* | **Input** | **Texture Coordinate**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Noise Texture** node, then set the **Scale** of the texture to **50.000**.
41. Add a **Math** node (*Shift* + *A* | **Converter** | **Math**) and a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**). Connect the **Color** output of the **SCALES** node to the **Color1** input socket of the **MixRGB** node, and the **Color** output of the **Noise Texture** to the **Color2** input socket.
42. Set the **MixRGB** blend type to **Add**, the **Fac** value to **1.000** and label it as **Scales_Noise**. To see the effect, connect its **Color** output to the **Height** input socket of the **Bump** node (but this is going to change very soon, so it's not mandatory at this step):



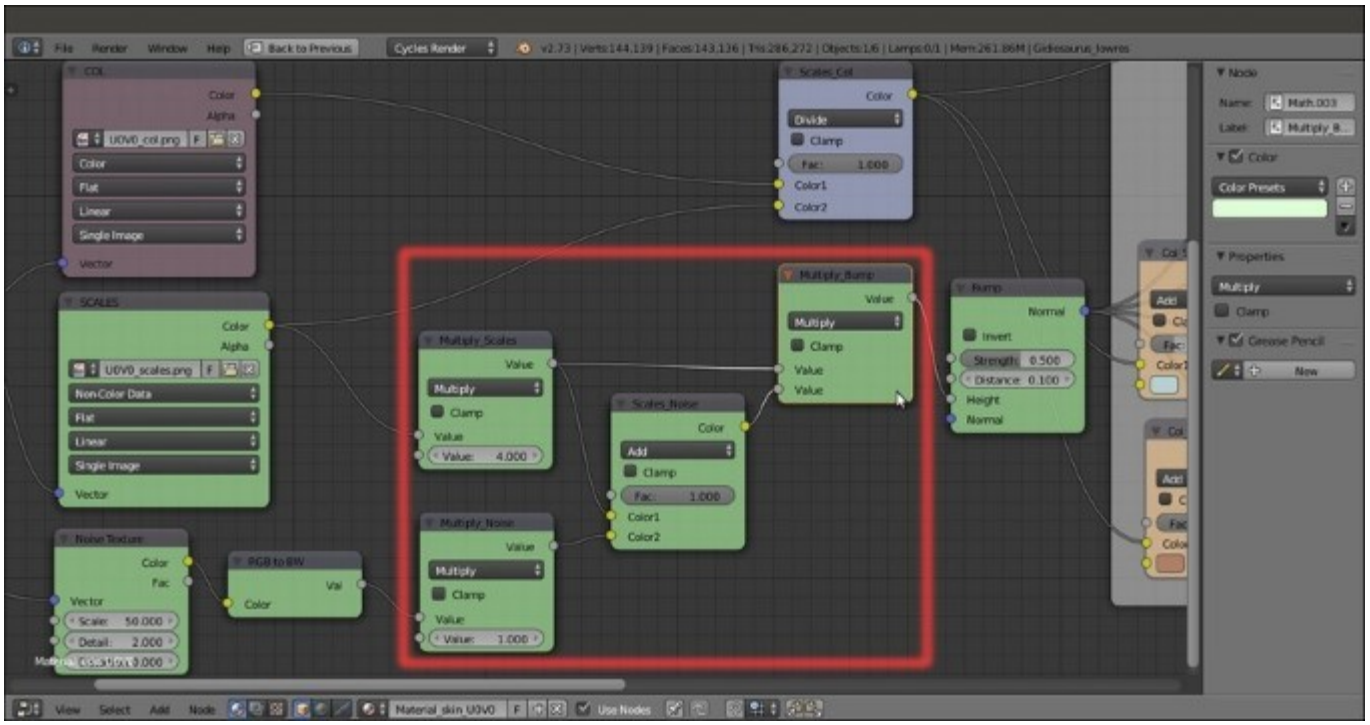
Adding some noise to the bump pattern part 1

43. Select the **Math** node and move it on the link connecting the **Noise Texture** node with the **Scales_Noise** node to paste it in between them: set the **Operation** to **Multiply**, the second **Value** to **1.000**, and label it as **Multiply_Noise**.
44. Press *Shift* + *D* to duplicate the **Multiply_Noise** node, change the label to **Multiply_Scales** and the second **Value** to **4.000**; paste it between the **SCALES** node and the **Scales_Noise** node.
45. Add an **RGB to BW** node (*Shift* + *A* | **Converter** | **RGB to BW**) and paste it between the **Noise Texture** node and the **Multiply_Noise** one:



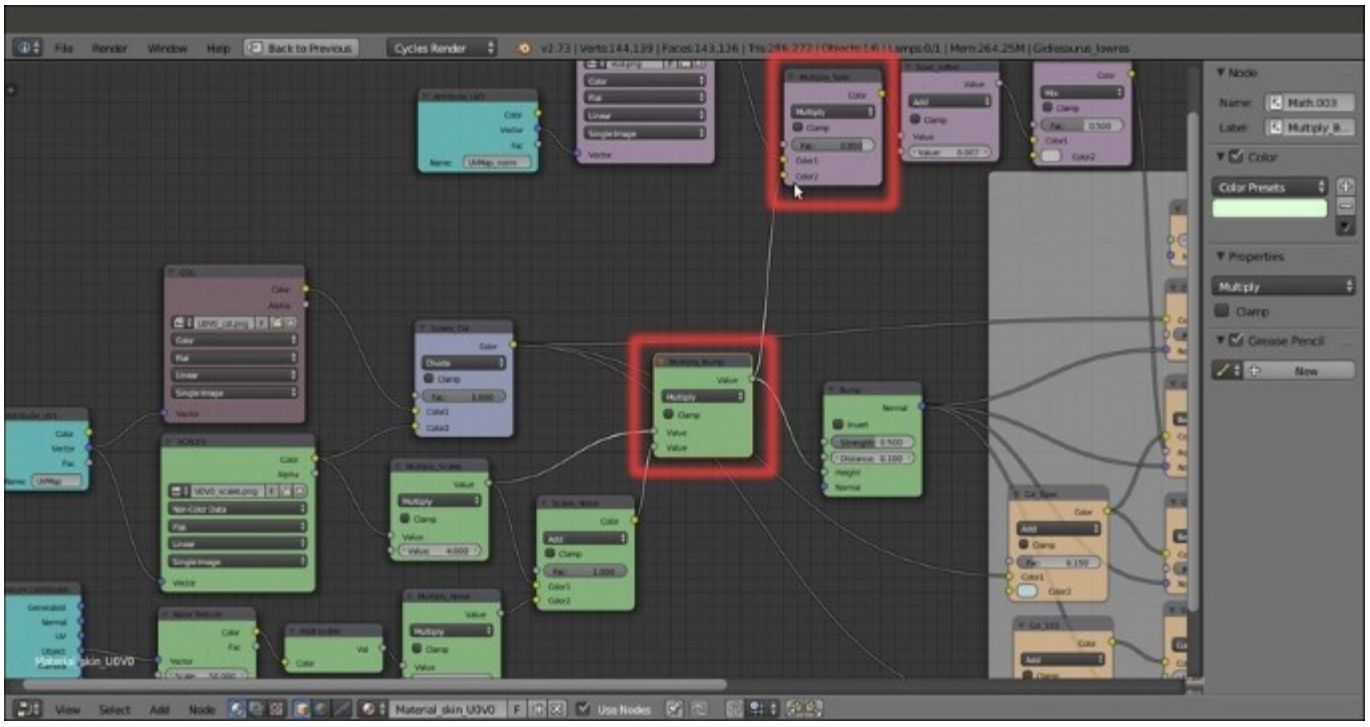
Adding some noise to the bump pattern part 2

46. Press *Shift* + *D* to duplicate the **Multiply_Scales** node and change the duplicate label to **Multiply_Bump**; connect the output of the **Multiply_Scales** to the first **Value** input socket of the **Multiply_Bump** node and the output of the **Scales_Noise** node to the second **Value** input socket. Connect the output of the **Multiply_Bump** node to the **Height** input socket of the **Bump** node:



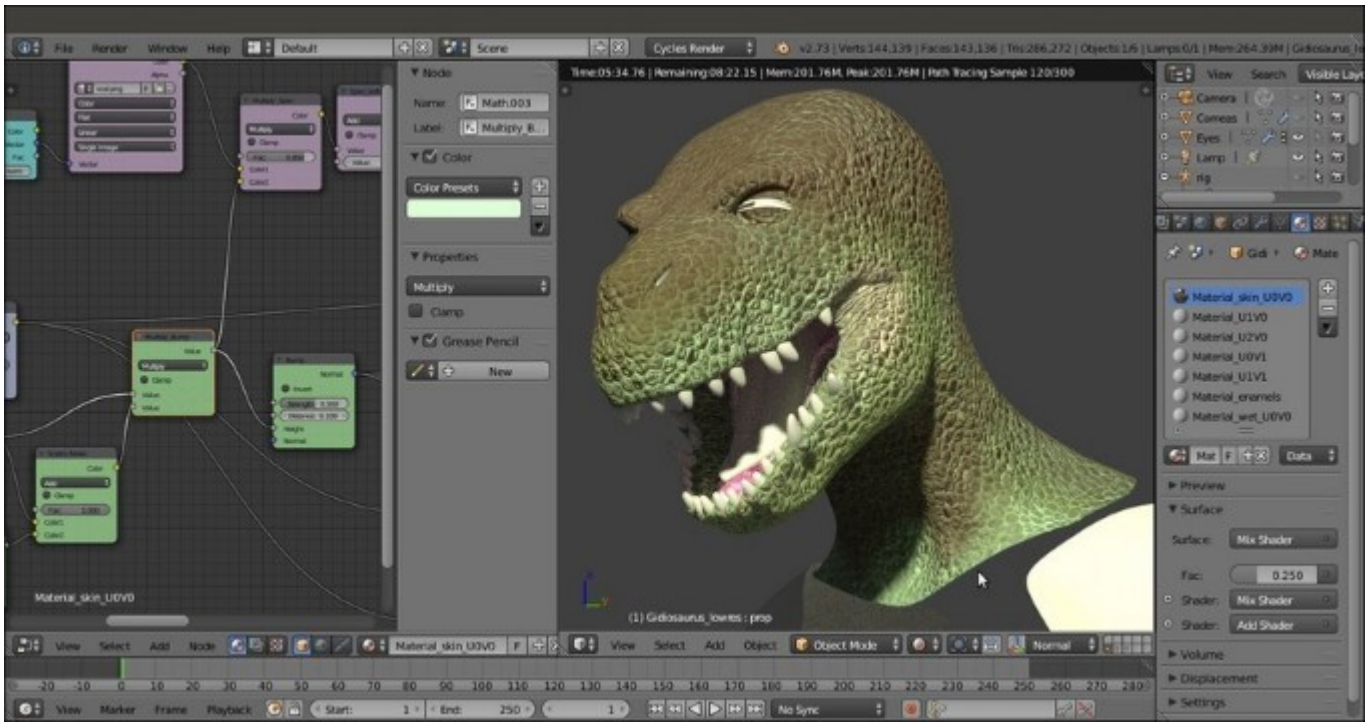
Adding some noise to the bump pattern part 3

47. Add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and paste it between the **VCOL** node and the **Spec_soften** node; label it as **Multiply_Spec**, set the **Blend Type** to **Multiply** and the **Fac** value to **0.850**; connect the output of the **Multiply_Bump** node to the **Color2** input socket of the **Multiply_Spec** node:



Modulating the specularity with the aid of the bump pattern output

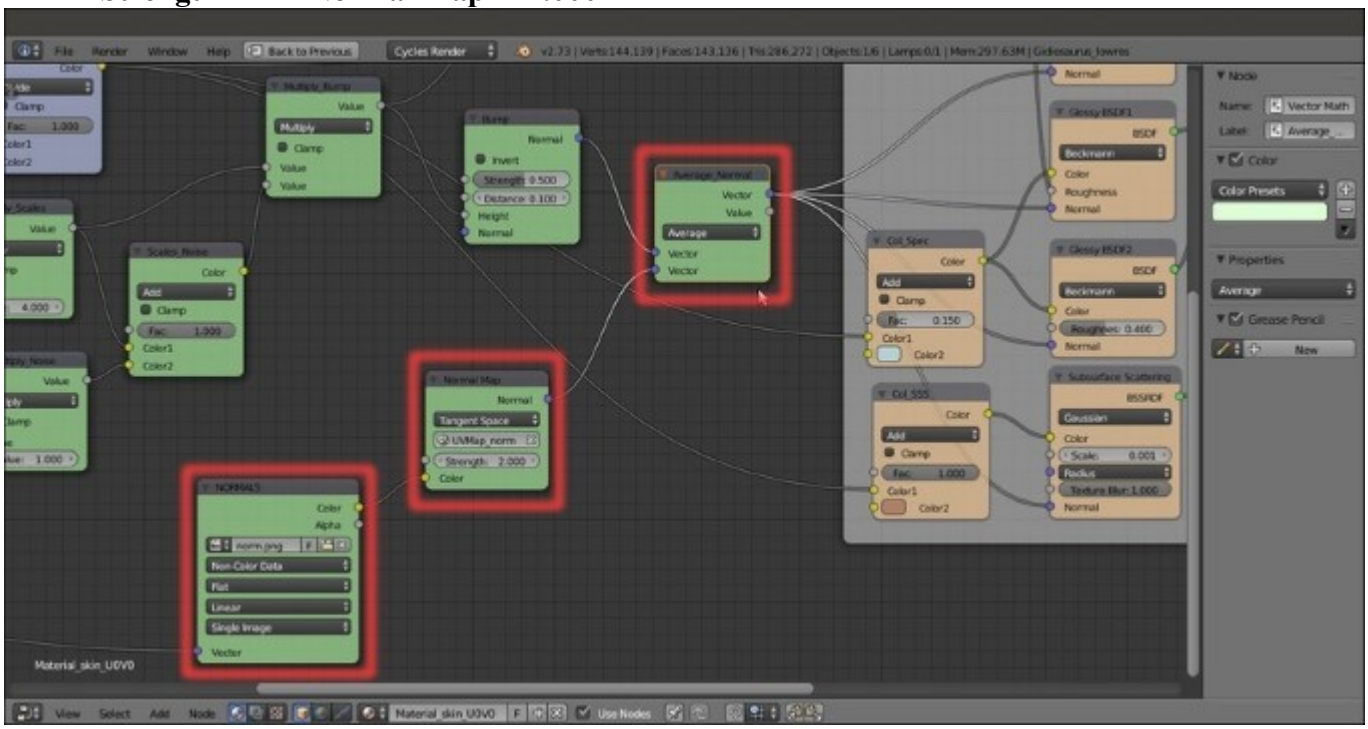
The overall bump effect is almost completed:



The new Rendered bump effect

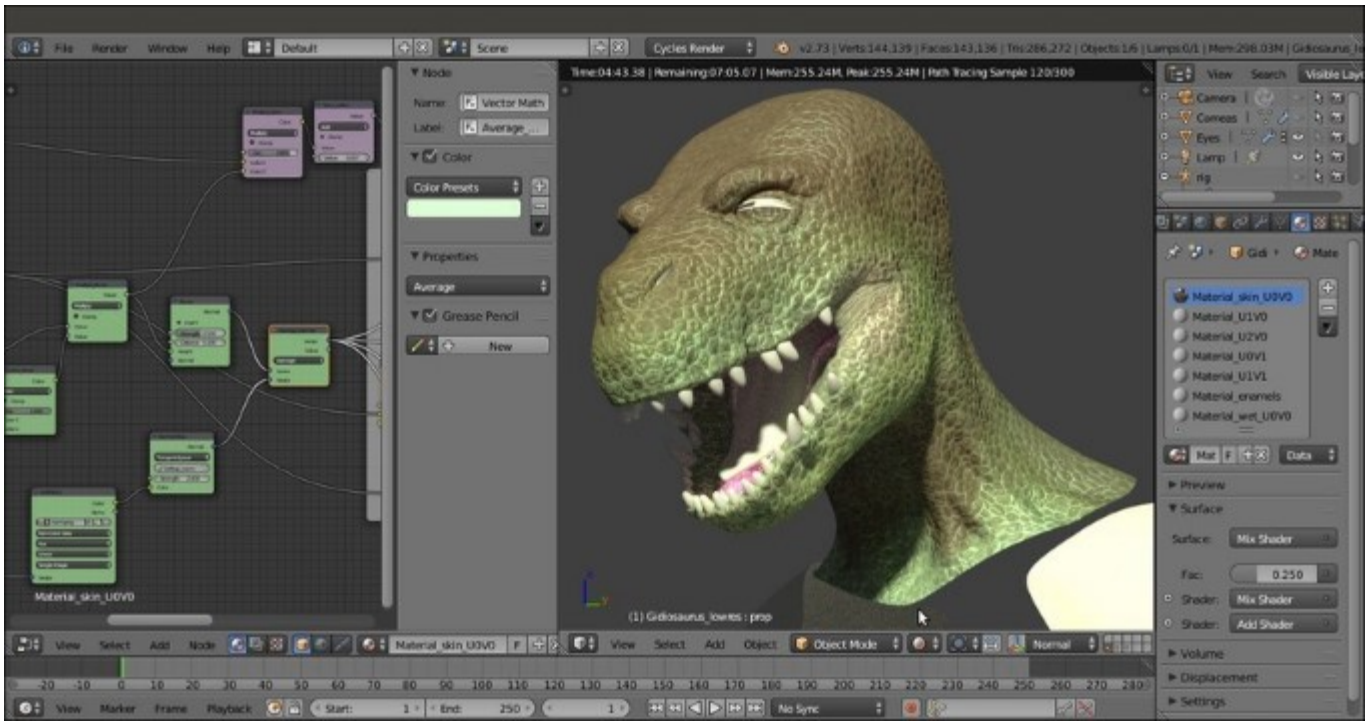
What is still missing now is the normal map we obtained from the sculpted **Gidiosaurus** mesh in Chapter 11, *Refining the Textures*.

48. Add a new **Image Texture** node (*Shift* + *A* | **Texture** | **Image Texture**) and a **Normal Map** node (*Shift* + *A* | **Vector** | **Normal Map**). Label the **Image Texture** node as **NORMALS**, then connect the **Vector** output of the **Attribute_UV2** node to the **Vector** input socket of the **NORMALS** node.
49. Connect the **Color** output of the **NORMALS** node to the **Color** input socket of the **Normal Map** node, then click on the **Open** button on the **NORMALS** node, browse to the `textures` folder and load the image `norm.png`. Set the **Color Space** of the **NORMALS** node to **Non-Color Data** and click on the empty slot in the **Normal Map** node to select the **UVMap_norm** coordinates layer.
50. Add a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**), label it as **Average_Normals** and paste it right after the **Bump** node; connect the output of the **Normal Map** node to the second **Value** input socket of the **Average_Normals** node.
51. Set the **Operation** of the **Average_Normals** node to **Average** and connect its **Vector** output to the **Vector** input sockets of the **Diffuse BSDF**, **Glossy BSDF1**, **Glossy BSDF2**, and **Subsurface Scattering** nodes.
52. Set the **Strength** of the **Normal Map** to **2.000**:



Adding the normal map output to the bump pattern

Finally we have completed the first skin material!

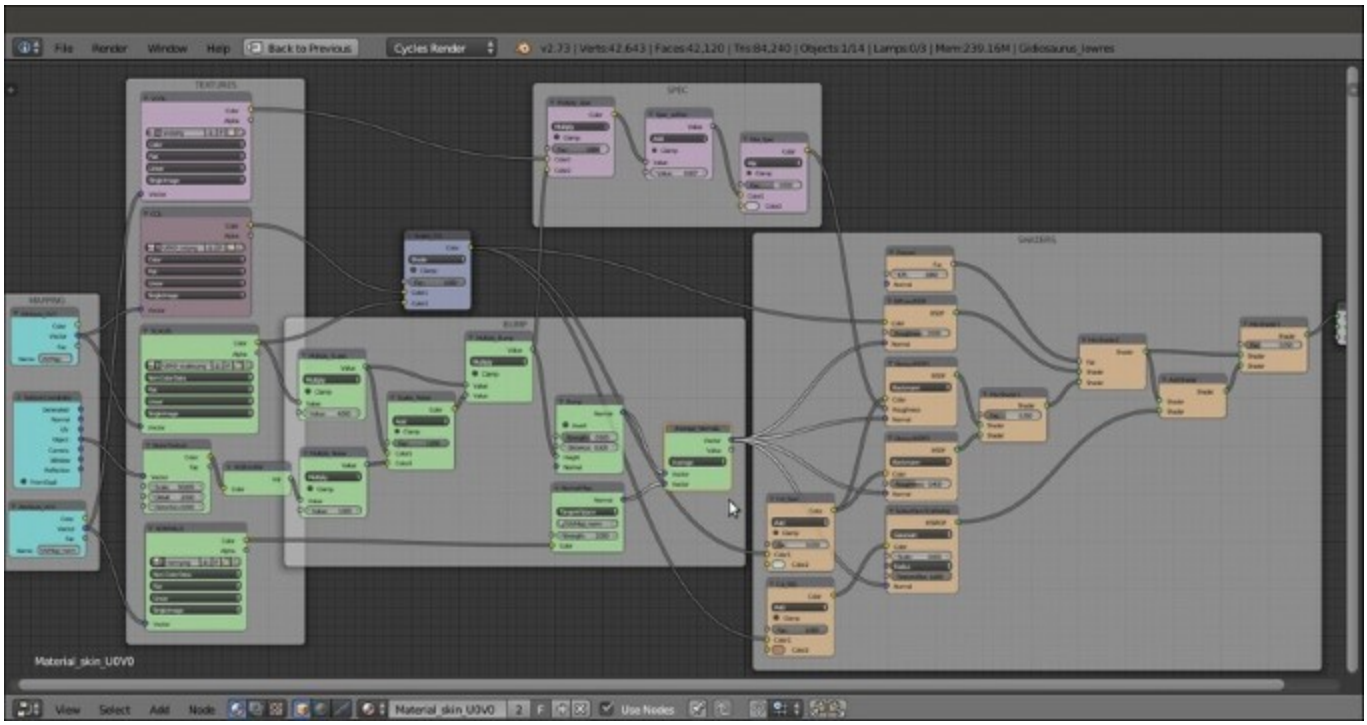


The completed Material_skin_U0V0

53. Save the file as `Gidiosaurus_skin_Cycles.blend`.

How it works…

This material can at first glance appear a bit complex, but actually the design behind it is quite simple as you can see in the following screenshot, where each component has been visually grouped by colors and frames (open the provided `Gidiosaurus_skin_Cycles_01.blend` file to have a better look):



The total skin material network

- From step 1 to step 18 we built the **SHADERS** part of the material, that is, the combination of the diffuse with the glossy component and the addition of the subsurface scattering effect.
- Note that the glossy component (the *specularity*) is obtained by mixing **two** glossy shaders with different roughness values; by setting the factor value of the **Mix Shader1** node to **0.350**, we give prevalence to the **Glossy BSDF1** node effect, which is to the node connected to the first top **Shader** input socket.
- Also, we added the subsurface scattering effect by the **Add Shader** node, and to further tweak the blending of the effect with the rest of the shader, we added the **Mix Shader3** node, to give prevalence to the output of the **Mix Shader2** node (that is the output of the diffuse plus the glossy components).
- From step 19 to step 24 we saw some not mandatory but useful tips for assigning colors to the nodes, in order to visually distinguish and/or group them and make the whole material network more easily readable.
- At step 25 we started to add the textures, first the diffuse color one and then the grayscale scales image that we used here to add details to the coloration (and later for the bump effect). By mixing the scales with the diffuse color through the **MixRGB** node set to a **Divide** blend type, we automatically obtained a scales pattern on the skin itself.
- From step 30 to step 33 we tweaked the diffuse color map to also affect the glossy and the subsurface scattering components, but with different hues.
- Note that at step 34 we used an **Attribute** node to set the UV coordinates layer to be used for the mapping of the textures. It would have been unnecessary in this case, with the **UVMap** coordinates layer being the first one and therefore the default one. Cycles, in fact, in the case of

image textures, automatically uses any existing UV coordinates layer. But, because later we also used a different UV coordinates layer, it was better to specify it.

- From step 35 to step 38 we improved the glossiness effect of the skin, by using the output of the `vcol.png` image we had previously baked and tweaked through the nodes inside the **SPEC** frame.
- From step 39 to step 47 we built the **BUMP** effect, by using the output of the **SCALES** image texture added through a **MixRGB** node to the output of a procedural **Noise Texture**. The **RGB to BW** node simply converts the colored output of the procedural noise to a grayscale output (and if you think we could have used the **Fac** output instead, well, it's not the same thing), and the **Multiply_Scales** and **Multiply_Noise** nodes set the strength of the outputs before the adding process. Through the **Multiply_Bump** node we also added the grayscale output of the combined bump to the glossy component.
- From step 48 to step 52 we also added the effect of the normal map we baked from the sculpted high resolution **Gidiosaurus** mesh to the bump pattern. The normal map is averaged, through the **Vector Math** node, with the bump output. Because of this averaging, the strength value of the normal map had to be set to double (**2.000**) to have full effect.

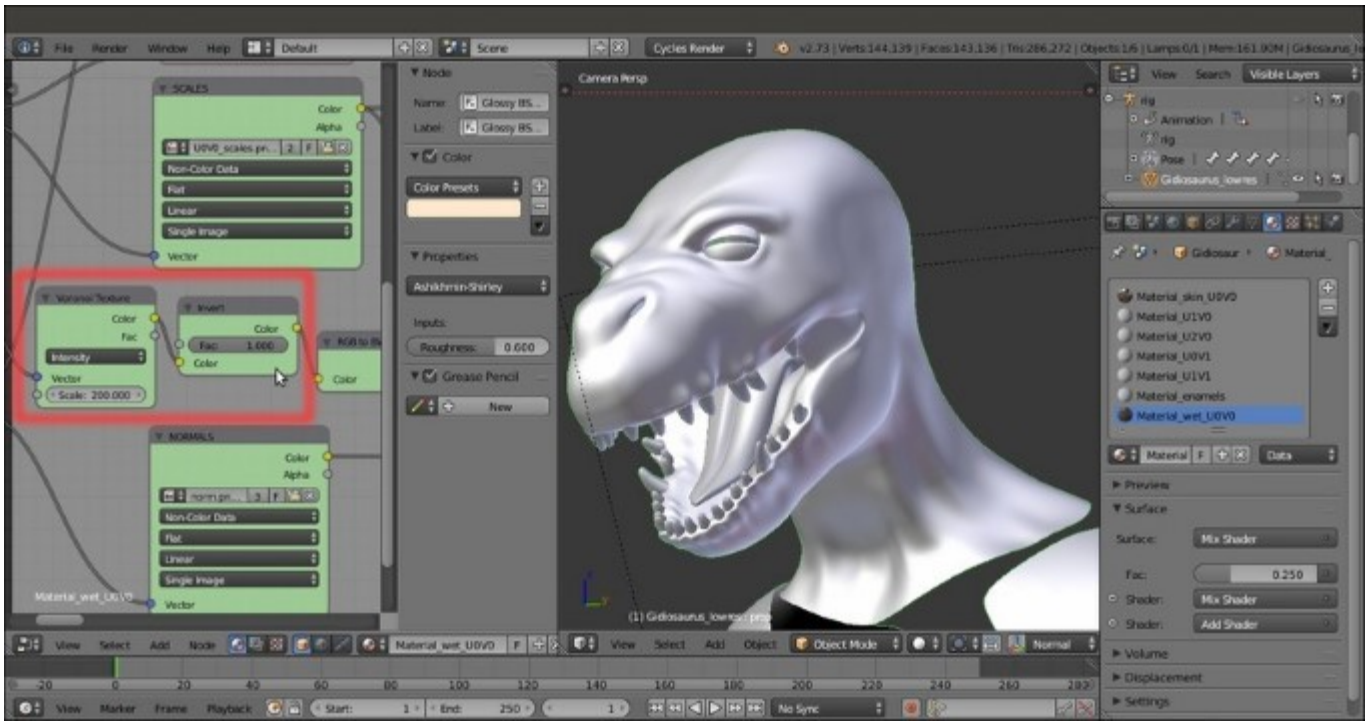
There's more…

Still focusing on the character's **head**, there is a material we can obtain from the skin material with some modification, the material for the wet parts of the character's skin (inner **eyelids**, **tongue**, inner **nostrils**).

Going on from the previously saved file:

1. If you think this is the case, especially if your computer (like mine) isn't very powerful, temporarily disable the **Rendered** preview by moving the mouse cursor inside the 3D viewport and pressing *Shift* + *Z*.
2. In the **Material** window, click on the `Material_wet_U0V0` material to select it.
3. Put the mouse pointer inside the **Node Editor** window, select the default two nodes already assigned to the material and delete them by pressing the *X* key.
4. Now, in the **Material** window, re-select the `Material_skin_U0V0`; put the mouse in the **Node Editor** window, press *A* twice to select everything, and press *Ctrl* + *C*.
5. Re-select the `Material_wet_U0V0`, put the mouse pointer inside the empty **Node Editor** window and press *Ctrl* + *V* to paste the copied material nodes.

   Now we have copied the nodes of the skin material to the material assigned to the parts that need to appear wet; it's enough now to tweak this material a bit to modify the bump pattern and the glossiness:
6. In the **Node Editor**, zoom to the **Noise Texture** node inside the **BUMP** frame; left-click on it to select it and then press the *X* key to delete it.
7. Press *Shift* + *A* and add a **Voronoi Texture** node (*Shift* + *A* | **Texture** | **Voronoi Texture**); left-click on the node and, by keeping the mouse button pressed, move the node a little bit on the frame, so it should automatically be parented to it.
8. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Voronoi Texture** node and the **Color** output of this latter node to the **RGB to BW** node input socket; set the **Voronoi Scale** to **200.000**.
9. Add an **Invert** node (*Shift* + *A* | **Color** | **Invert**) and paste it between the **Voronoi Texture** and the **RGB to BW** nodes:

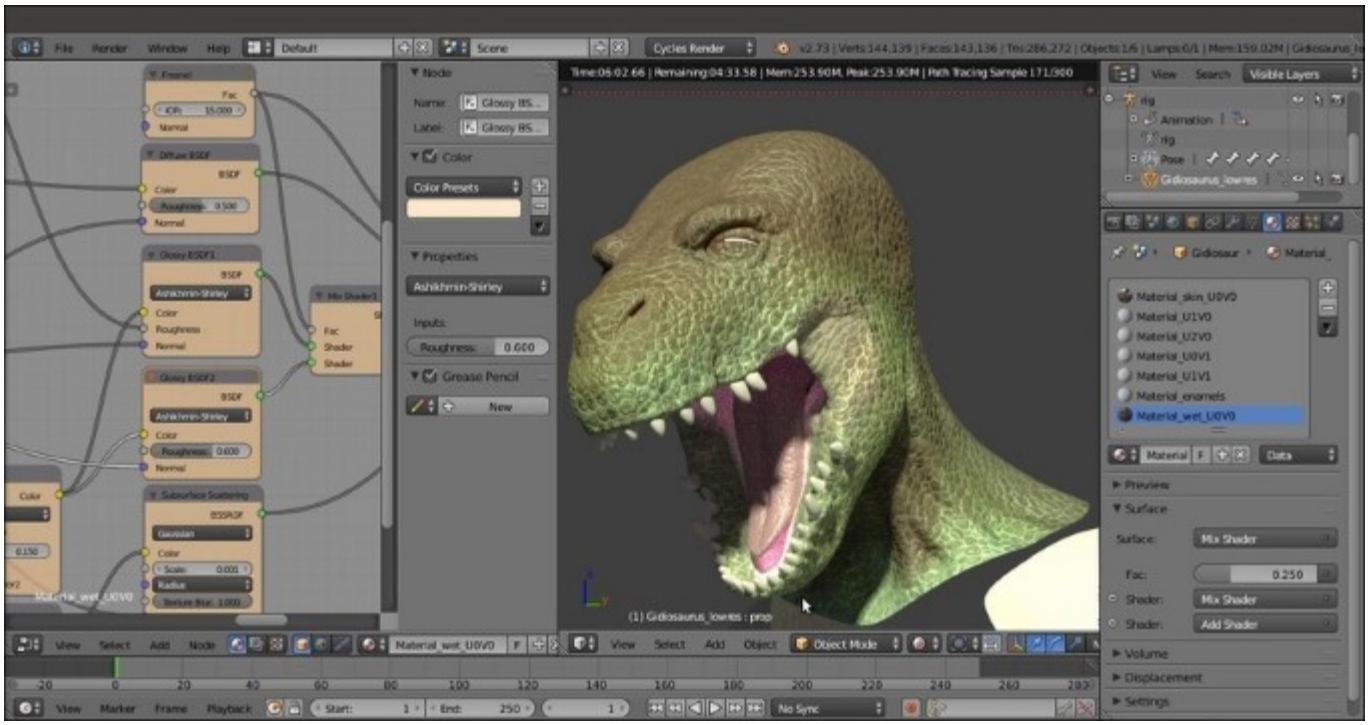


The different texture nodes of the "Material_wet_U0V0"

10. Scroll the **Node Editor** window a bit to the right to find the **Multiply_Noise** node: change the label to **Multiply_Voronoi** and the second **Value** to **0.025**.
11. Find the **Scales_Col** node and change **Blend Type** from **Divide** to **Multiply**.
12. Now go to the **SHADERS** frame; change the **IOR** value of the **Fresnel** node to **15.000** and connect its output to the **Fac** input socket of the **Mix Shader1** node; change the **Distribution** of both the **Glossy BSDF1** and **Glossy BSDF2** nodes to **Ashikhmin-Shirley** and set the **Roughness** of the **Glossy BSDF2** node to **0.600**.

    We substituted the **Noise Texture** node with a **Voronoi Texture** node to give a kind of organic look to the surface of the **tongue** of the creature.

    In the following screenshot, we can see the result of the wet material; note that for the occasion I opened the mouth wide, to make the inside more visible:



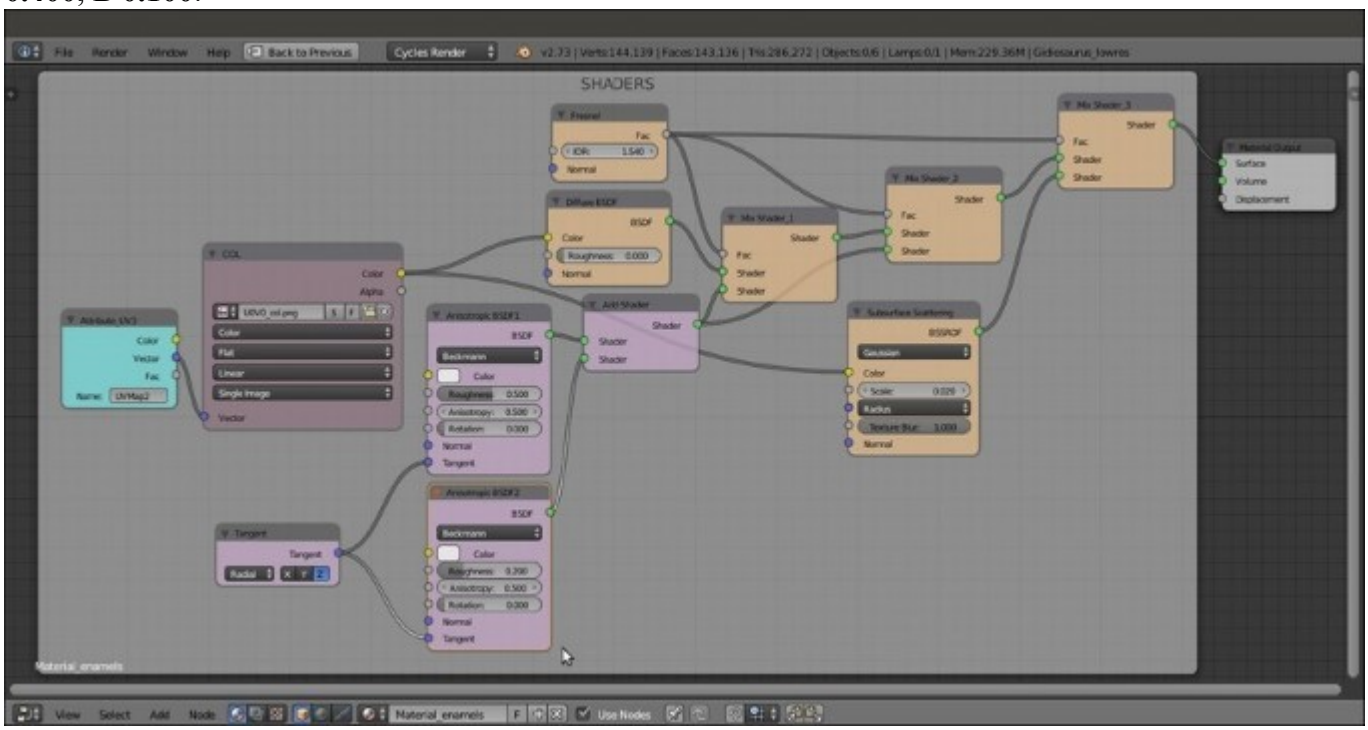
The rendered wet material

One more material we are going to create in this section of the recipe is the **Material_enamels** for **teeth** and **talons**; in this case, we just need mostly the **SHADERS** frame's nodes with the single contribution of the color image texture `U0V0_col.png`, here using the **UVMap2** coordinates layer to avoid having to create **5** different materials for the **talons** alone (originally distributed in different tiles). By the way, nothing is stopping you from creating several talon materials, if you prefer.

13. Again, select, copy and paste the skin material to the enamels material slot through the **Node Editor** window, as we have already done in steps 3, 4 and 5.
14. This time, just delete the unnecessary nodes, in short keeping only the **Attribute** node, the **COL** node and the **SHADERS** frame with its parented nodes.
15. Change the UV coordinates layer in the **Name** slot of the **Attribute** node to **UVMap2** (and the label to **Attribute_UV3**). Lower the **Roughness** value of the **Diffuse BSDF** node to **0.000**.
16. Go to the **SHADERS** frame; select and delete the **Col_Spec** and **Col_SSS** nodes, then connect the **Color** output of the **COL** node also to the **Color** input socket of the **Subsurface Scattering** node.
17. Select and delete the **Glossy BSDF1** and the **Glossy BSDF2** nodes.
18. Add 2 **Anisotropic BSDF** shader nodes (*Shift* + *A* | **Shader** | **Anisotropic BSDF**), a **Tangent** node (*Shift* + *A* | **Input** | **Tangent**) and detach the **Add Shader** node from the **Mix Shader3** node.
19. Label the two **Anisotropic BSDF** shader nodes as **Anisotropic BSDF1** and **Anisotropic BSDF2** and connect them to the two **Shader** input sockets of the **Add Shader** node. Connect

the output of the **Tangent** node to the **Tangent** input sockets of the two **Anisotropic** shader nodes.

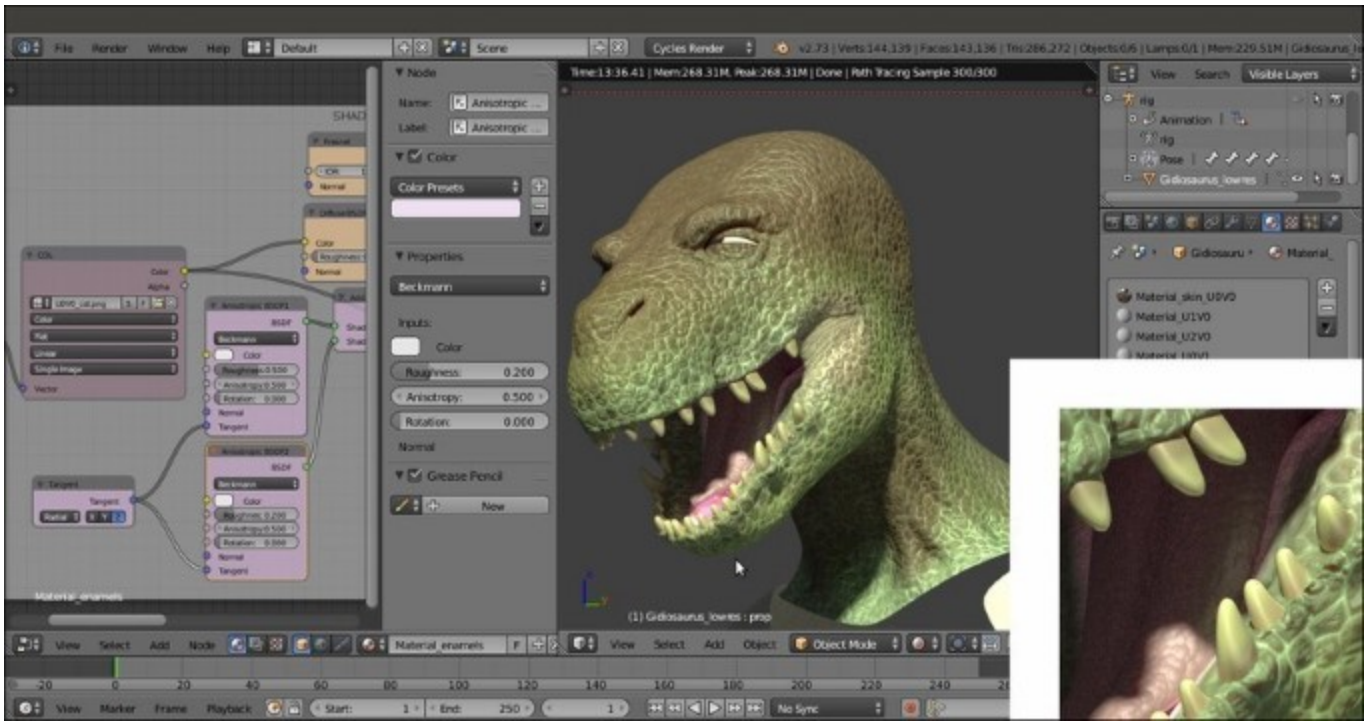
20. Set the **Tangent** of the **Tangent** node to **Z**. Set the **Anisotropy** of both the **Anisotropic** nodes to **0.500**, the **Roughness** of the **Anisotropic BSDF1** node to **0.500** and the **Roughness** of the **Anisotropic BSDF2** node to **0.200**.
21. Connect the **Add Shader** output to both the second **Shader** input sockets of the **Mix Shader1** and **Mix Shader2** nodes.
22. Set the **IOR** value of the **Fresnel** node to **1.540** and connect the **Fresnel** output to the **Fac** input sockets of the **Mix Shader1**, **Mix Shader2**, and **Mix Shader3** nodes.
23. Connect the output of the **Diffuse BSDF** shader node to the first **Shader** input socket of the **Mix Shader1** node, then connect the output of the **Mix Shader1** node to the first **Shader** input socket of the **Mix Shader2** node.
24. Connect the output of the **Subsurface Scattering** node to the second **Shader** input socket of the **Mix Shader3** node.
25. In the **Subsurface Scattering** node, change the **Scale** to **0.020** and the **Radius** to **R 1.000**, **G 0.400**, **B 0.100**.



The "Material_enamels" network

26. Save the file.

Thanks to the two **Anisotropic** shaders with their different roughness values, we obtained a nice specularity effect along the length of the **teeth** (and therefore also of the **talons**):



The rendered preview of the teeth (and talons) shader

See also

- Shameless self-promotion—one other cookbook, published by Packt Publishing, explaining the logic behind Cycles materials and textures and with several material recipes (https://www.packtpub.com/hardware-and-creative/blender-cycles-materials-and-textures-cookbook-third-edition)
- The online documentation (http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes)

Making a node group of the skin shader to reuse it

Once we are satisfied with the reptile skin shader created for the character's **head**, we can copy it to the other parts of the **body**, that is to the other material slots, and then apply the necessary modifications. Those, in this case, just consist of different color and scales image textures.

This means that all the other shader parts can be reused as they are. In this recipe, in fact, we are going to make a node group of these parts so as to easily re-use the shader for the other materials slots.

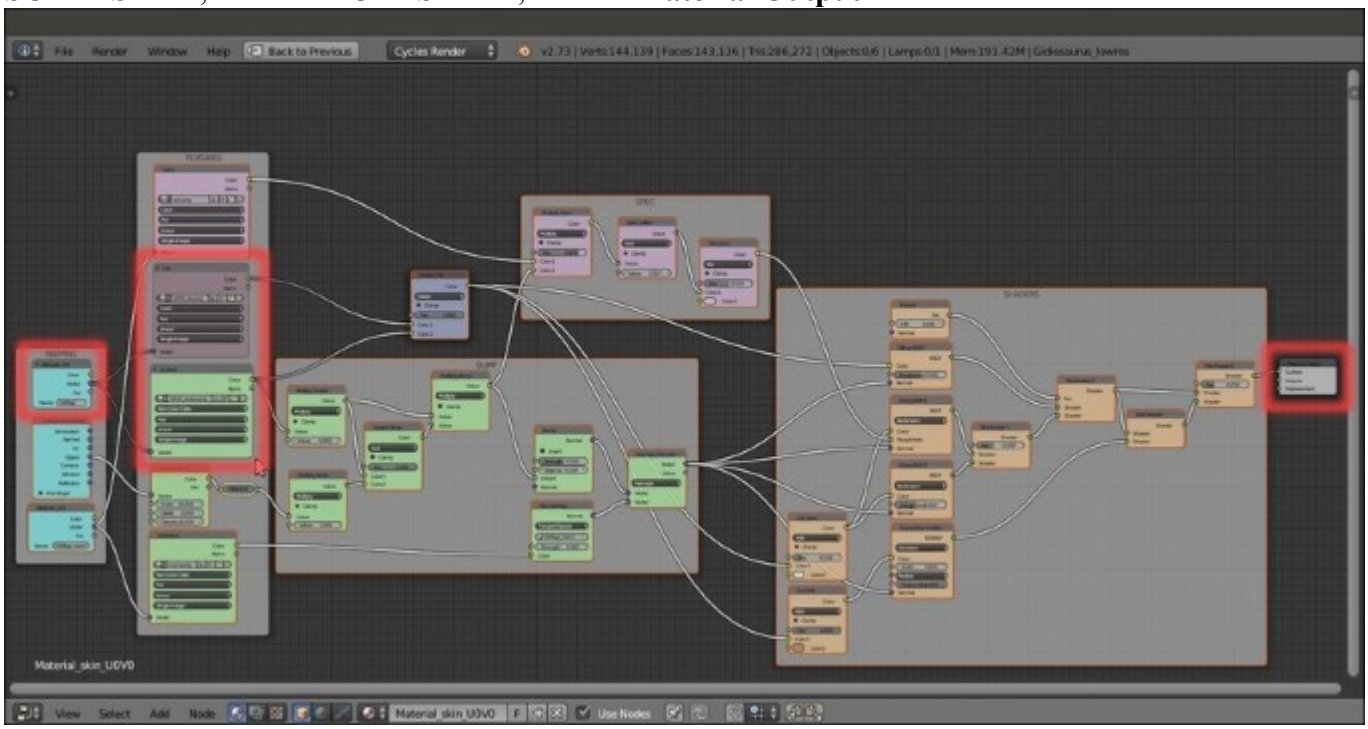
Getting ready

Just start Blender and re-open the previously saved `Gidiosaurus_skin_Cycles_01.blend` file.

How to do it…

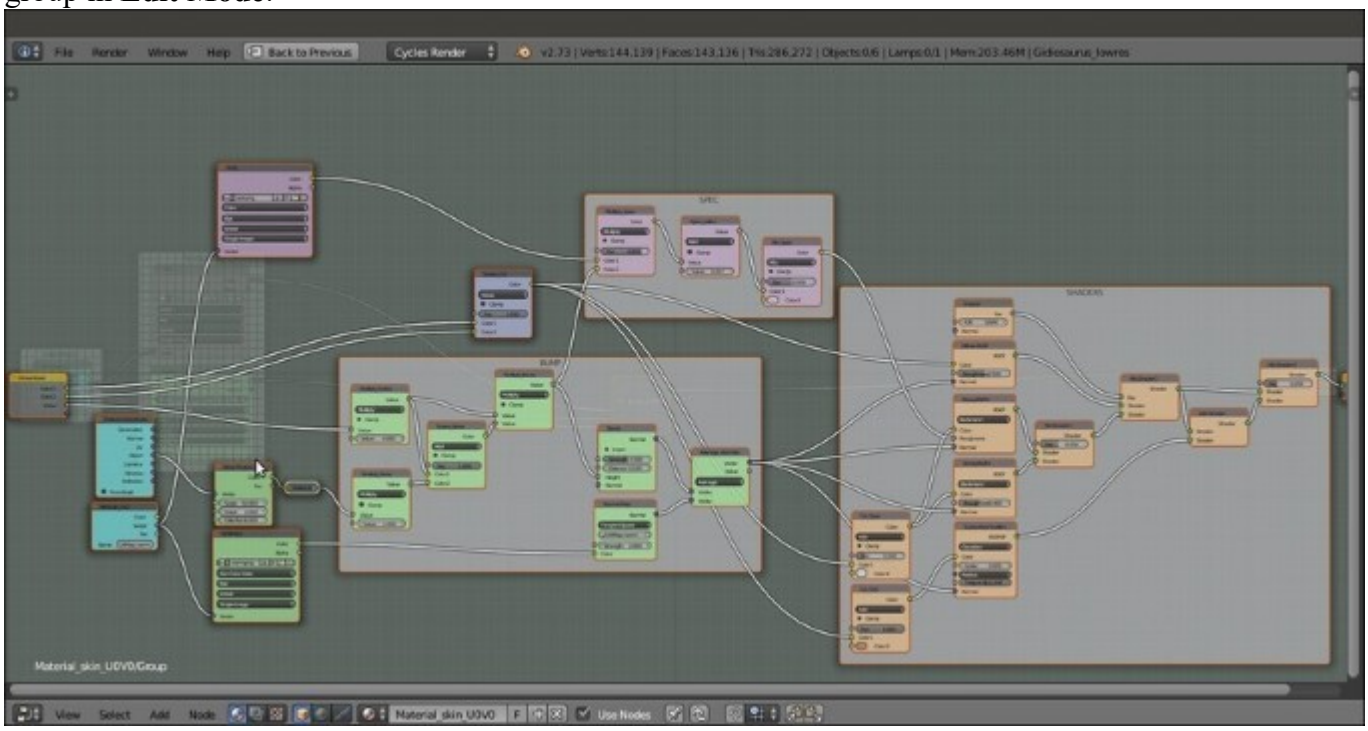
Let's start to create our skin node group:

1. In the **Material** window, select the slot of the `Material_skin_U0V0`.
2. Put the mouse pointer in the **Node Editor** window, press the *B* key and left-click to box-select all the nodes with their respective frames. Then, press the *Shift* key and right-click (twice for each one) to deselect the **Attribute_UV1** node, the **MAPPING** frame, the **COL** node, the **SCALES** node, the **TEXTURES** frame, and the **Material Output** node:



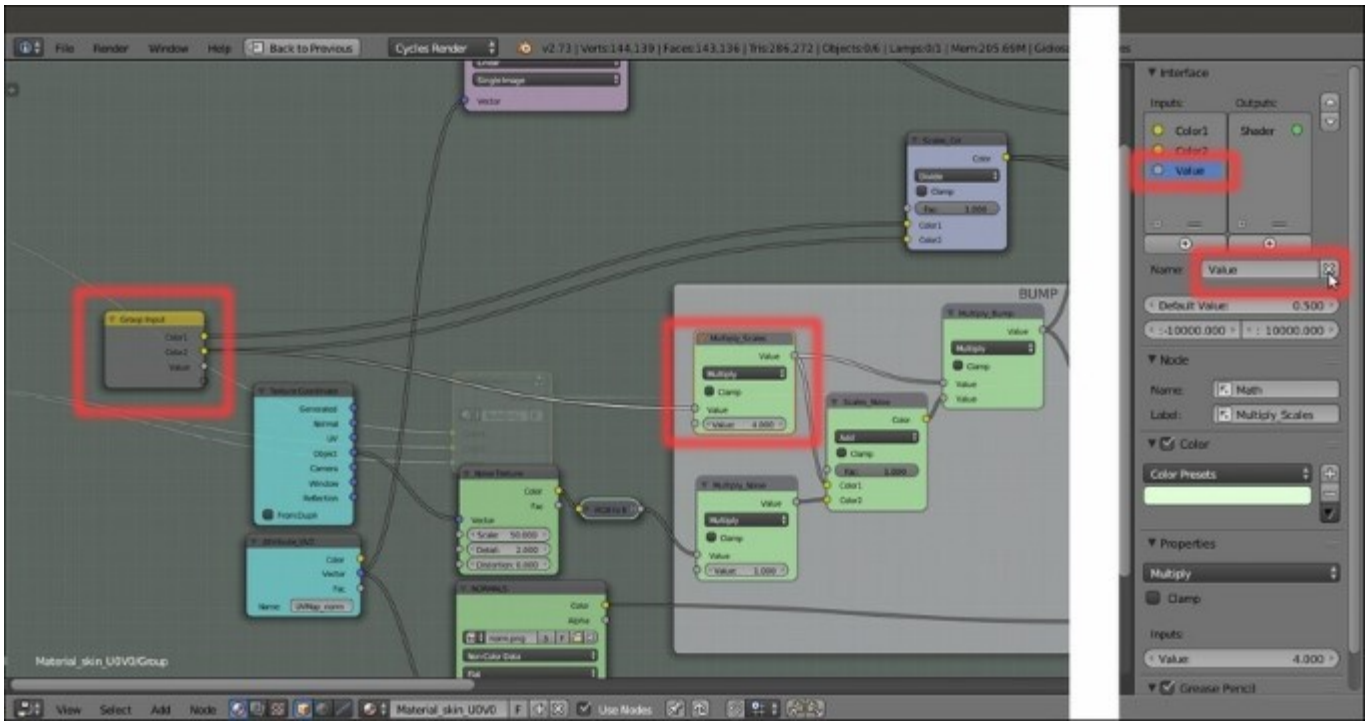
The box-selected nodes and the highlighted deselected ones

3. Press *Ctrl* + *G* to make a node group of the selected nodes; automatically you are inside the group in **Edit Mode**:



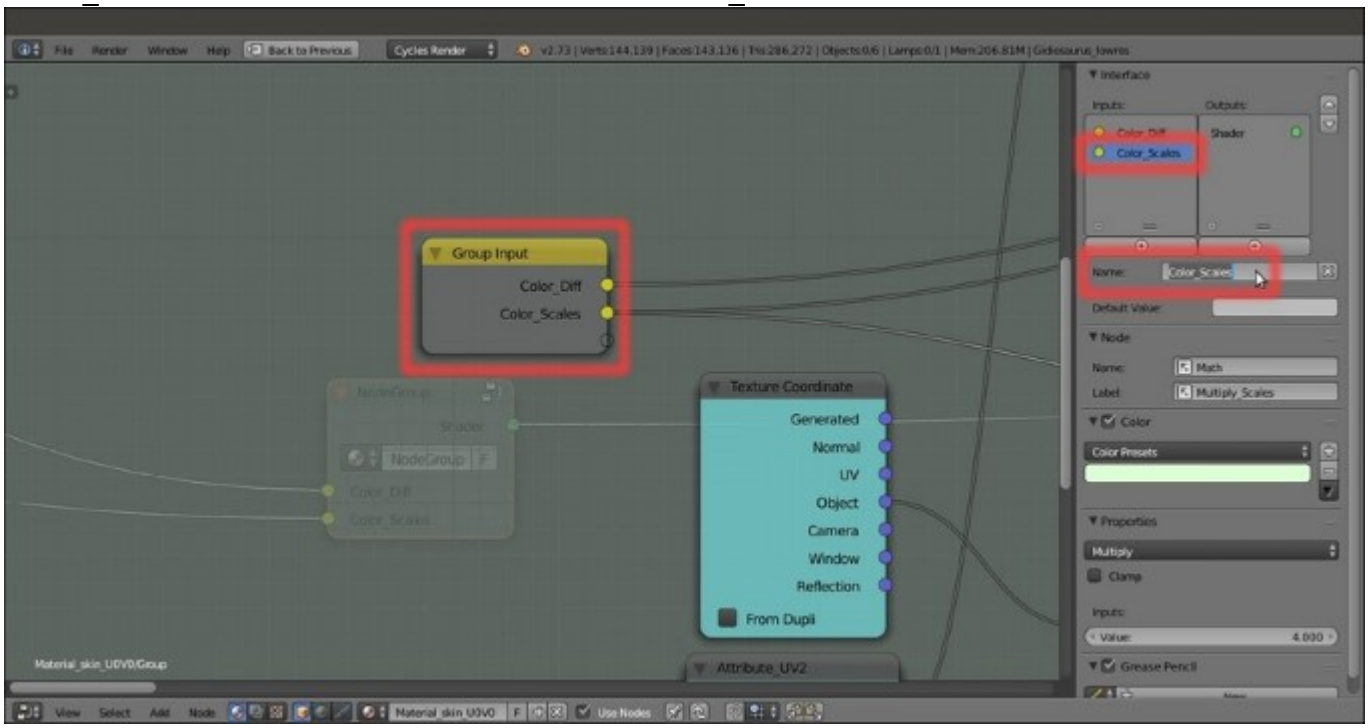
Inside the node group in Edit Mode

4. Click on the **Group Input** node to select it and zoom in on it, then press *N* to call the **Properties** sidepanel. Connect the **Color2** output socket to the first **Value** input socket of the **Multiply_Scales** node, replacing the connection coming from the **Value** output.
5. Go to the **Properties** sidepanel and in the **Interface** subpanel, click on the **Value** item inside the little **Inputs** window; then go down to the **Name** slot and click on the **X** icon button to delete the socket from the **Group Input** node:



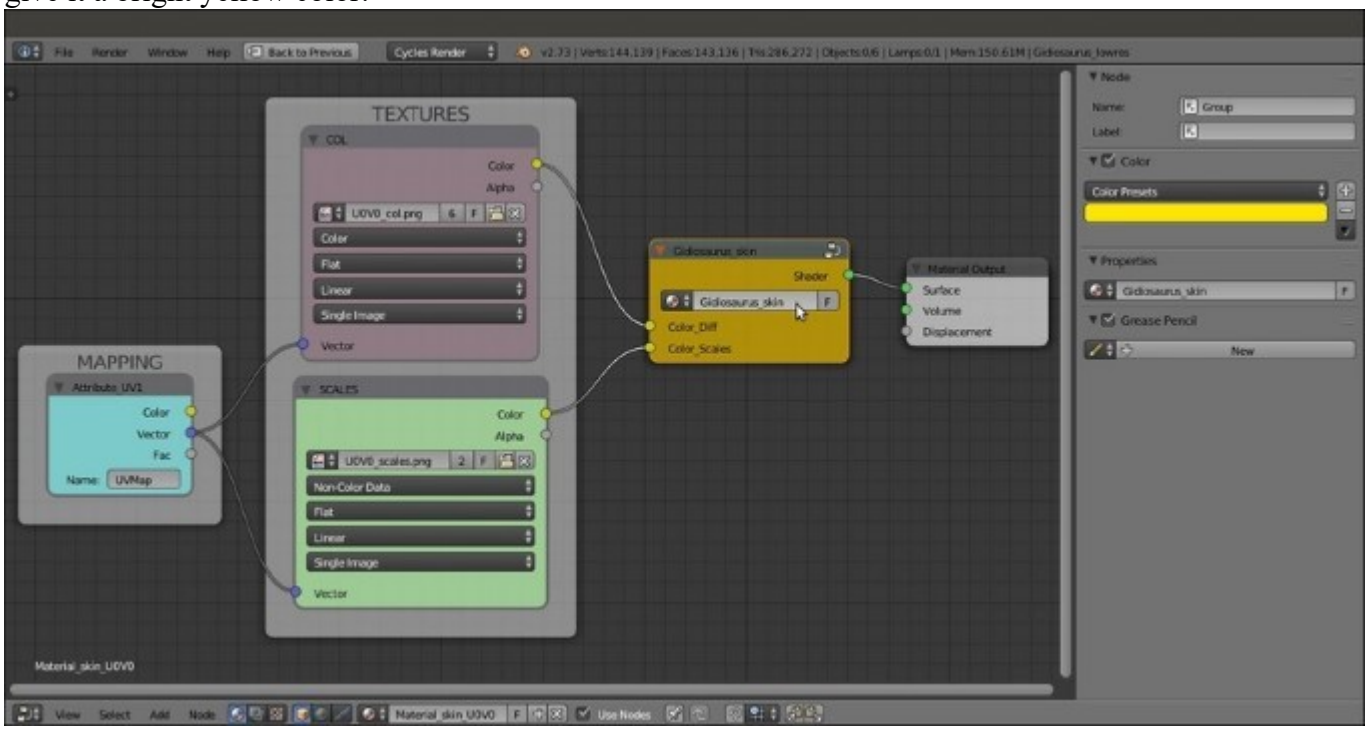
Tweaking the node group input socket connections

6. Still in the **Name** slot in the **Interface** subpanel, select the **Color1** item and rename it **Color_Diff**. Select the **Color2** item and rename it **Color_Scales**:



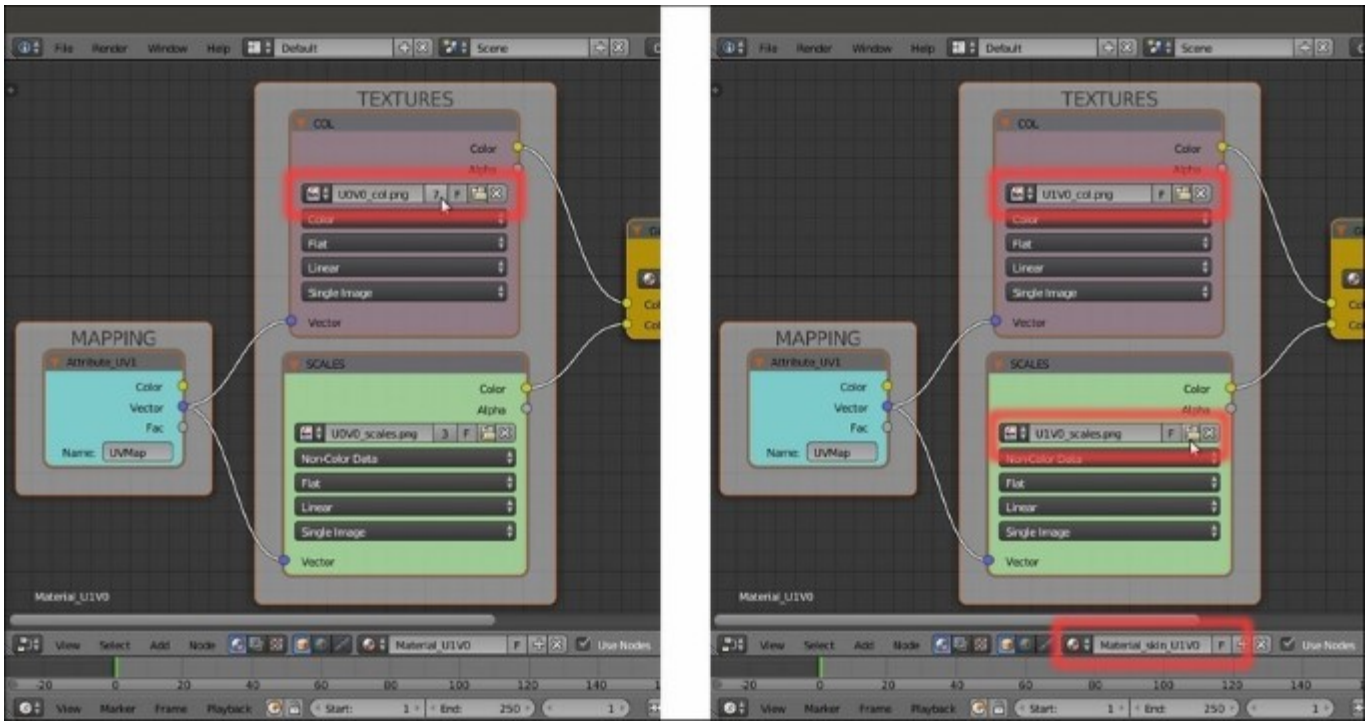
Renaming the node group input sockets

7. Press *Tab* to exit out of **Edit Mode** and close the node group; rename it **Gidiosaurus_skin** and give it a bright yellow color:



The "Gidiosaurus_skin" node group

8. Now select everything by pressing the *A* key twice and then press *Ctrl* + *C*.
9. In the **Material** window, select the `Material_U1V0`, then click on the **Use Nodes** button in the **Surface** subpanel.
10. Put the mouse pointer in the **Node Editor** window and delete the already selected default nodes (the **Diffuse BSDF** connected to the **Material Output** nodes), then press *Ctrl* + *V* to paste all the copied nodes inside the window.
11. Zoom on the **COL** and **SCALES** nodes. Click on the numbered button to the right side of the texture name to make them single users, then click on the folder icon buttons to browse to the `textures` folder and load the proper images according to the material name, that is, the `U1V0_col.png` and `U1V0_scales.png` image textures.
12. Rename the material as `Material_skin_U1V0`:



Making the copied textures single users and loading the right image textures for the "Material_skin_U1V0"

13. Repeat step 8 to step 12 for the other remaining **3** material slots and then save the file as `Gidiosaurus_skin_Cycles_02.blend`.

The "Material_wet_U0V0" network and the completed Gidiosaurus shading in the rendered preview

How it works…

Of course, it wasn't mandatory to make a group of the skin shader to reuse it for the other material slots; we could just have selected, copied and pasted all the nodes and frames as they were at the end of the previous recipe.

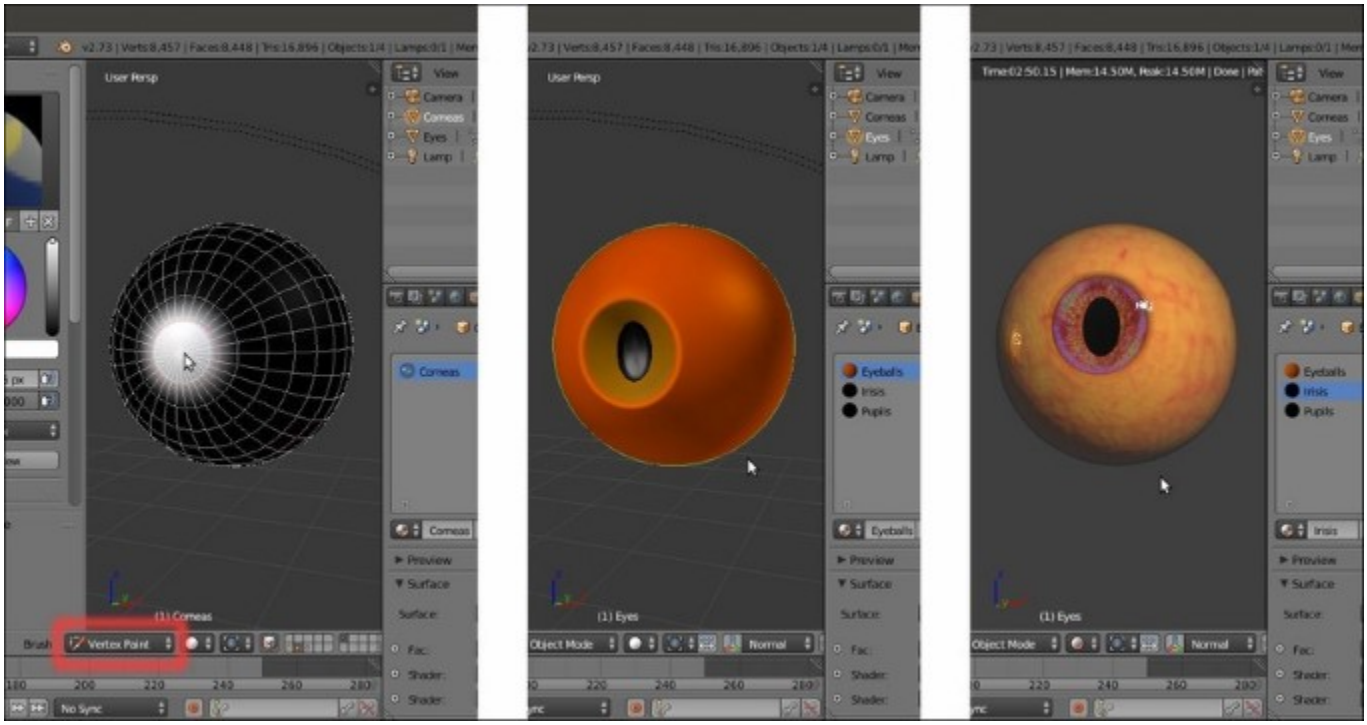
The (quite big) advantage in having a node group instanced in different materials is that if you need to change something in the internal network, you don't have to repeat the modifications in the node group of each material. It's enough to do it in **Edit Mode** in one of the instances, and all the internal modifications will be reflected in all the instances of the node group used by the other materials.

Building the eyes' shaders in Cycles

The character's **eyes** are made up of two **UV Spheres**, the **Corneas** and the **Eyes** objects: the bigger **Corneas** one enveloping a smaller **Eyes** sphere, which in turn is made up of **three** parts: the **eyeballs**, the **irises**, and the **pupils**.

The **Corneas** sphere was first painted with a totally black **Vertex Color** layer, then painted with a white color only to the vertices corresponding to the front crystalline lens.

The **Eyes** sphere has **three** different materials assigned to the **three** different parts:



The Corneas object in Vertex Paint mode, the Eyes object with its three materials and the Rendered preview of the textured objects together

Getting ready

Start Blender and open the `Gidiosaurus_skin_Cycle_02.blend` file; save it as `Gidiosaurus_shaders_Cycles.blend`.

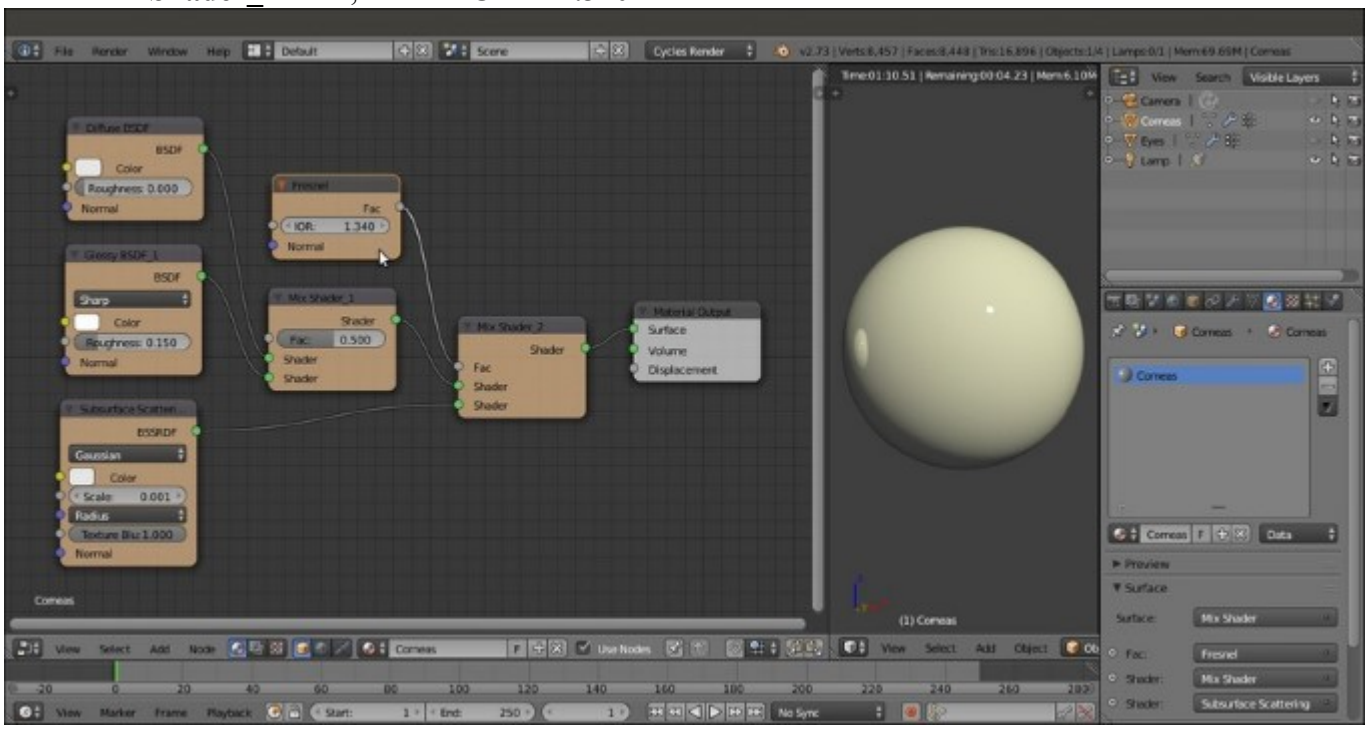
1. Enable only the **6th** and the **12th** scene layer, in order to have visible only the **Corneas**, the **Eyes** and the **Lamp** objects (actually the **Camera** is also on the **6th** scene layer, but it's hidden and at the moment we don't need it).
2. Zoom the 3D view onto the **Corneas** and **Eyes** objects, and press *Shift* + *Z* to start the **Rendered** preview.

3. In the **Outliner**, disable the *Restrict view-port visibility* button of the **Eyes** object to hide it.
4. Select the **Corneas** object and go to the **Material** window.

How to do it…

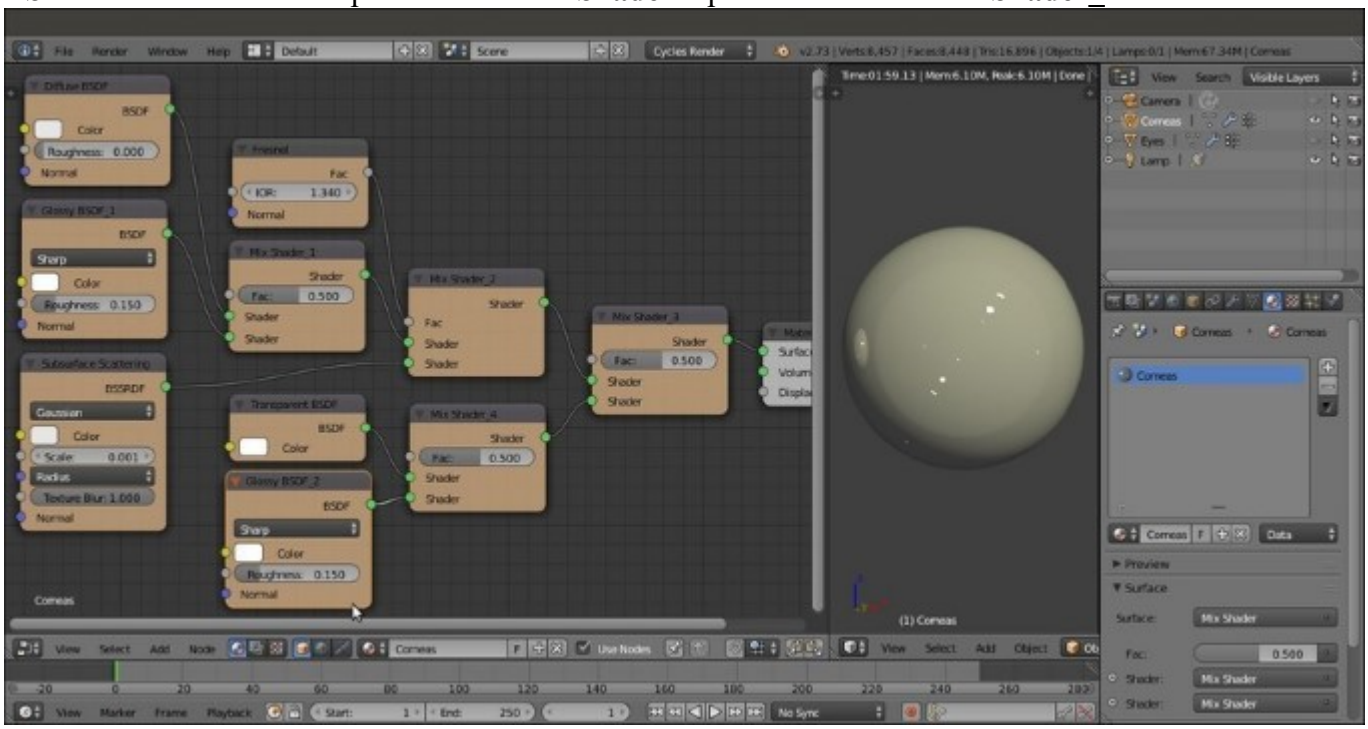
So, let's start with the **Corneas** material, first:

1. In the **Material** window, click on the **New** button in the **Surface** subpanel. Rename the material as `Corneas`.
2. In the **Material** window, switch the **Diffuse BSDF** shader node with a **Mix Shader** node (label it as **Mix Shader_1**). In the first **Shader** slot, select a **Diffuse BSDF** shader node and in the second one select a **Glossy BSDF** shader (label it as **Glossy BSDF1**).
3. Go to the **Node Editor** window and set the **Roughness** of the **Glossy BSDF1** shader to **0.150**, the **Color** to pure white and the **Distribution** to **Sharp**.
4. Select the **Mix Shader_1** node and press *Shift* + *D* to duplicate it. Label the duplicate as **Mix Shader_2**, then add a **Subsurface Scattering** node (*Shift* + *A* | **Shader** | **Subsurface Scattering**). Connect the output of the **Mix Shader_1** node to the first **Shader** input socket of the **Mix Shader_2** node and the output of the **Subsurface Scattering** shader node to the second **Shader** input socket.
5. Change the **Subsurface Scattering** falloff from **Cubic** to **Gaussian**, set the **Scale** to **0.001** and the **Radius** to **R 9.436**, **G 3.348**, **B 1.790**.
6. Add a **Fresnel** node (*Shift* + *A* | **Input** | **Fresnel**) and connect its output to the **Fac** input socket of the **Mix Shader_2** node; set the **IOR** to **1.340**:



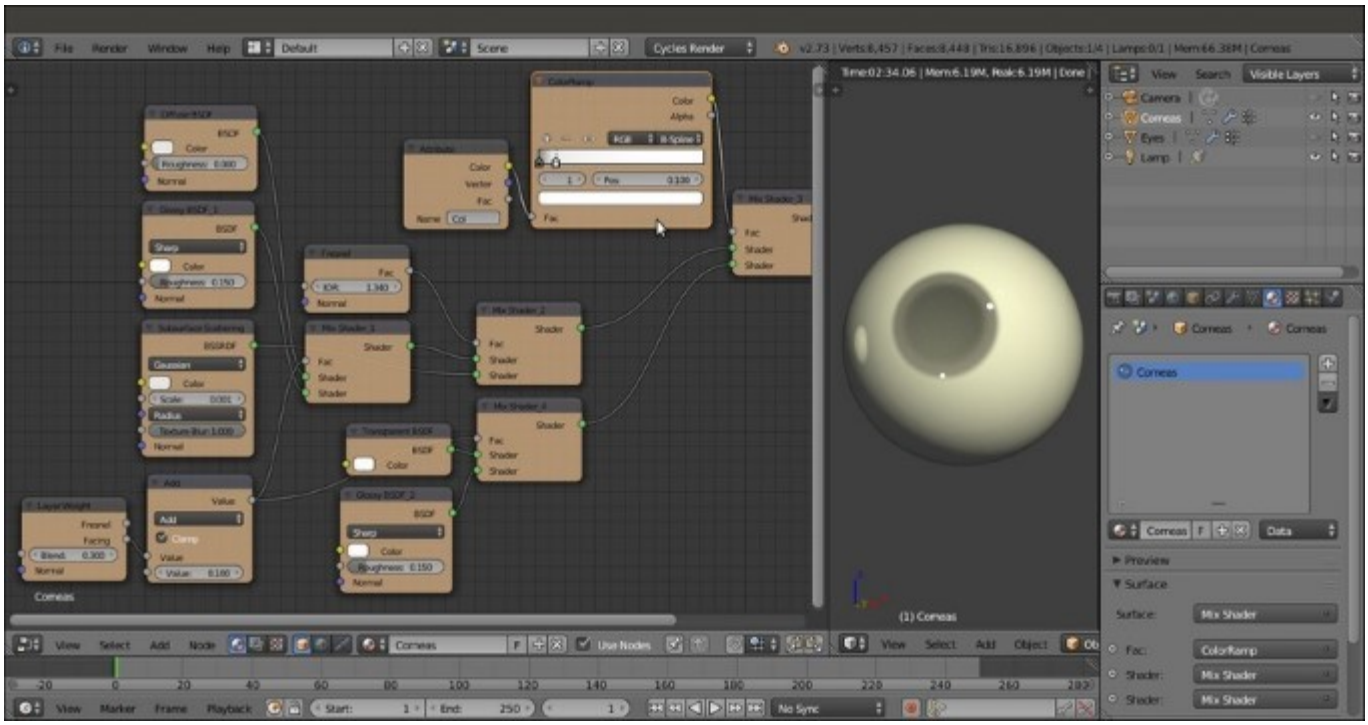
The basic starting "Corneas" shader

7. Add a new **Mix Shader** node (*Shift* + *A* | **Shader** | **Mix Shader**), label it as **Mix Shader_3** and paste it between the **Mix Shader_2** and the **Material Output** node.
8. Add a new **Mix Shader** node (*Shift* + *A* | **Shader** | **Mix Shader**), label it as **Mix Shader_4** and connect its output to the second **Shader** input socket of the **Mix Shader_3** node.
9. Add a **Transparent BSDF** shader (*Shift* + *A* | **Shader** | **Transparent BSDF**) and connect it to the first **Shader** input socket of the **Mix Shader_4** node.
10. Select and press *Shift* + *D* to duplicate the **Glossy BSDF1** node; label the duplicate as **Glossy BSDF2** and connect its output to the second **Shader** input socket of the **Mix Shader_4** node:



The "Corneas" shader with the added transparency nodes

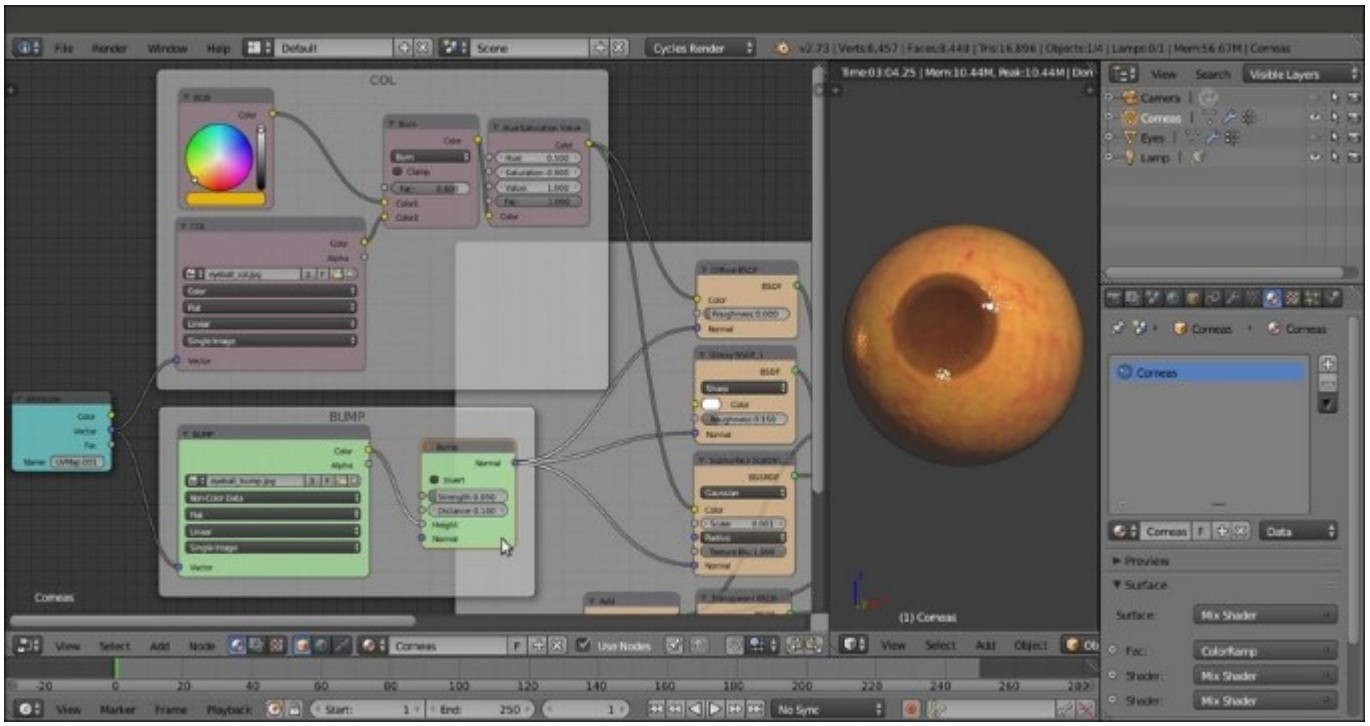
11. Add a **Layer Weight** node (*Shift* + *A* | **Input** | **Layer Weight**) and a **Math** node (*Shift* + *A* | **Converter** | **Math**); set the **Blend** factor of the **Layer Weight** to **0.300** and connect its **Facing** output to the first **Value** input socket of the **Math** node, then set the second **Value** to **0.100** and check the **Clamp** item.
12. Connect the **Math** (labeled as **Add**) output to the **Fac** input sockets of the **Mix Shader_1** and **Mix Shader_4** nodes.
13. Add an **Attribute** node (*Shift* + *A* | **Input** | **Attribute**) and a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**). In the **Name** slot of the **Attribute** node, type **Col**, then connect its **Color** output to the **Fac** input socket of the **ColorRamp** node.
14. In the **ColorRamp** node, set the **Interpolation** to **B-Spline** and move the white color stop to position **0.100**. Connect its **Color** output to the **Fac** input socket of the **Mix Shader_3** node.



The "Corneas" shader with the transparency area located by the Vertex Color layer

15. Add two **Image Textures** nodes (*Shift* + *A* | **Texture** | **Image Texture**) and label them respectively as **COL** and **BUMP**.
16. Add an **Attribute** node (*Shift* + *A* | **Input** | **Attribute**); in the **Name** slot type **UVMap.001** and connect its **Vector** output to the **Vector** input sockets of the two image texture nodes.
17. Add an **RGB** node (*Shift* + *A* | **Input** | **RGB**), a **MixRGB** node (*Shift* + *A* | **Input** | **MixRGB**) and a **Hue Saturation Value** node (*Shift* + *A* | **Input** | **Hue/Saturation**).
18. Click on the **Open** button of the **COL** node to browse to the `textures` folder and load the image `eyeball_col.jpg`.
19. Connect the **Color** output of the **COL** node to the **Color2** input socket of the **MixRGB** node and the output of the **RGB** node to the **Color1** input socket; set the **Blend Type** to **Burn** and the **Fac** value to **0.800**.
20. Connect the **Color** output of the **MixRGB** node to the **Color** input socket of the **Hue Saturation Value** node, and the output of this latter node to the **Color** input sockets of the **Diffuse BSDF** and **Subsurface Scattering** nodes.
21. Set the **RGB** node color to **R 0.800**, **G 0.466**, **B 0.000**; set the **Saturation** value of the **Hue Saturation Value** node to **0.900**.
22. Click on the **Open** button of the **BUMP** node to browse to the `textures` folder and load the image `eyeball_bump.jpg`; set **Color Space** to **Non-Color Data**.
23. Add a **Bump** node (*Shift* + *A* | **Vector** | **Bump**) and connect the **Color** output of the **BUMP** node to the **Height** input socket of the **Bump** node. Connect the **Normal** output of this latter node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF1**, and **Subsurface Scattering** nodes, and set the **Strength** to **0.050**.

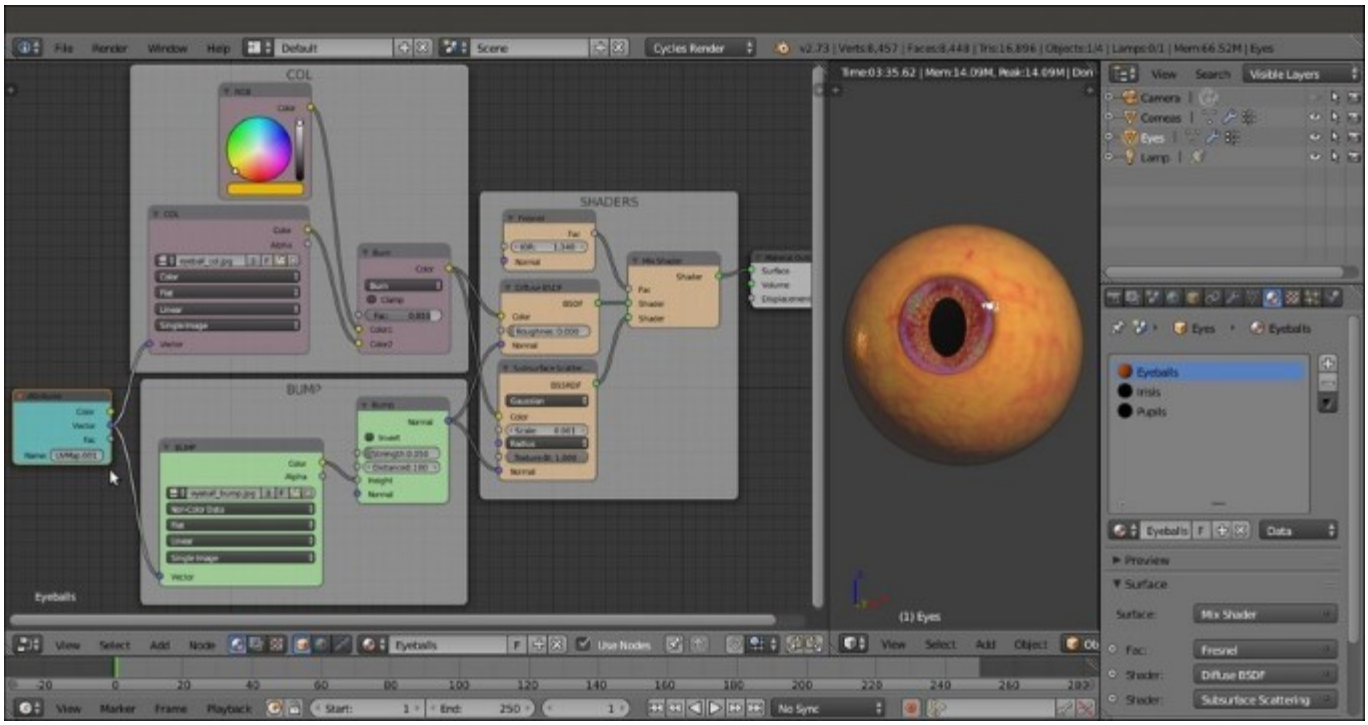
24. If you wish, add frames and colors to the different components to make the shader more easily readable:



The textured "Corneas" material

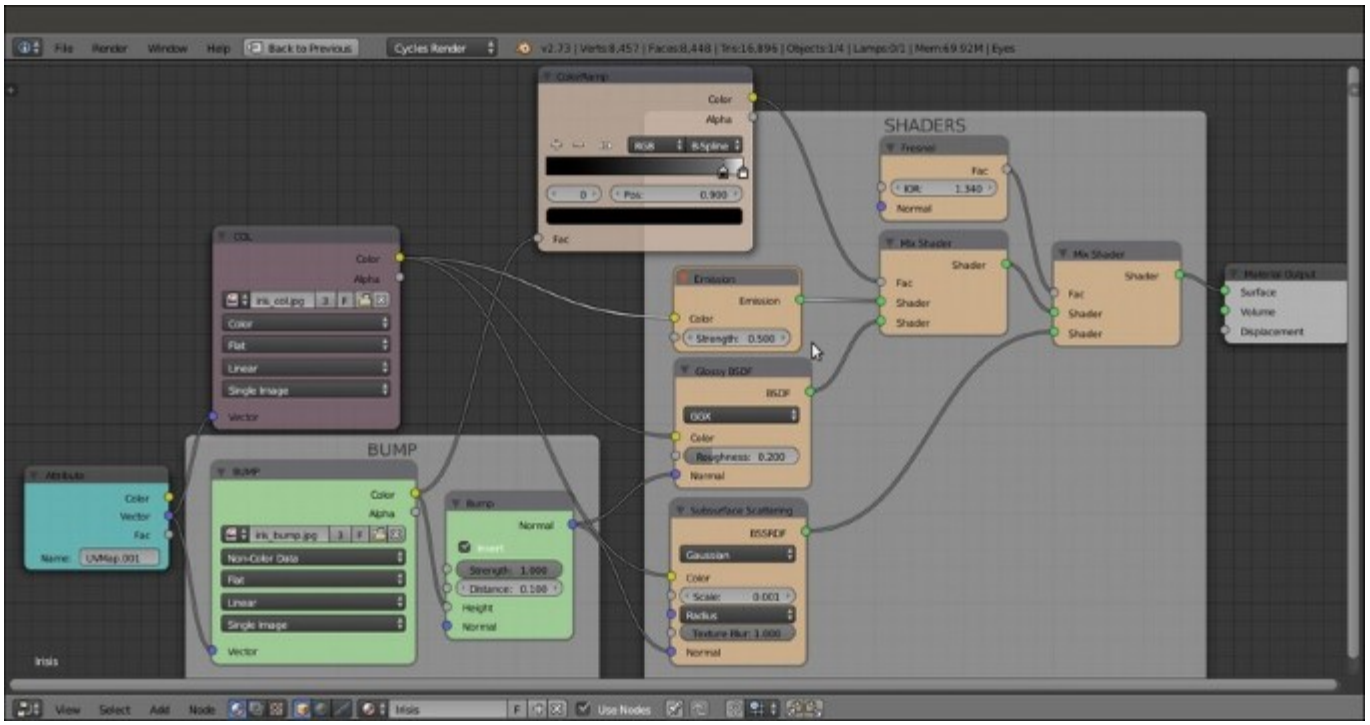
Now let's quickly see the materials for the **Eyes** object:

- As you can see in the following screenshot, the `Eyeballs` material is essentially the same as we just made for the **Corneas** except for the transparent part; this material, by the way, is obsolete because it's hidden behind the **Corneas'** opaque surface, so can be safely omitted (but I left it in place in case you want to try the totally transparent **Cornea**):



The "Eyeballs" material and the completed rendered eye

- The `Irises` material follows the same scheme; the only differences are in the fact that it uses different image textures (`iris_col.jpg` and `iris_bump.jpg`) and that a contrasted (by a **ColorRamp** node) version of the bump image is used as a factor for the mixing of an **Emission** shader; note that the color map is also connected to this **Emission** shader:



The "Irises" material network

- The Pupils are a simple, basic, black diffuse material.

To have a look at these materials, open the `Gidiosaurus_shaders_Cycles.blend` file and select the **Corneas** and **Eyes** objects in the **Outliner**.

How it works…

These shaders are quite simple; the more complex one is the shader for the **Corneas**, essentially because it's made up of **two** materials, one with a slight bump effect and one totally smooth, mixed on the ground of the black and white **Vertex Color** layer that takes care also of the distribution of the transparent and opaque materials on the **Corneas** object itself.

If you are wondering why we didn't use the `Eyeball` material on the underlying **Eyes** sphere, leaving the **Corneas** object totally transparent, the reason is simple: in Cycles, to have a material transparent but also reflecting the environment, you need to use a **Transparent** shader mixed with a **Glass** or a **Glossy** shader node, that inevitably will make whatever material is behind appear darker; sometimes this can look right, in this case I preferred to use a different approach.

Note

Note that the transparent part in front of the iris of the cornea, to be anatomically correct, should be a convex, bulging half sphere; instead, we modeled the cornea as a simple spherical sheath around the eyeball to avoid complications with the open/closed movements of the eyelids.

Building the armor shaders in Cycles

The last thing to do, for this chapter, is to create the shaders for the **Armor** object, made up of metallic **plates** and leather **tiers**.

Getting ready

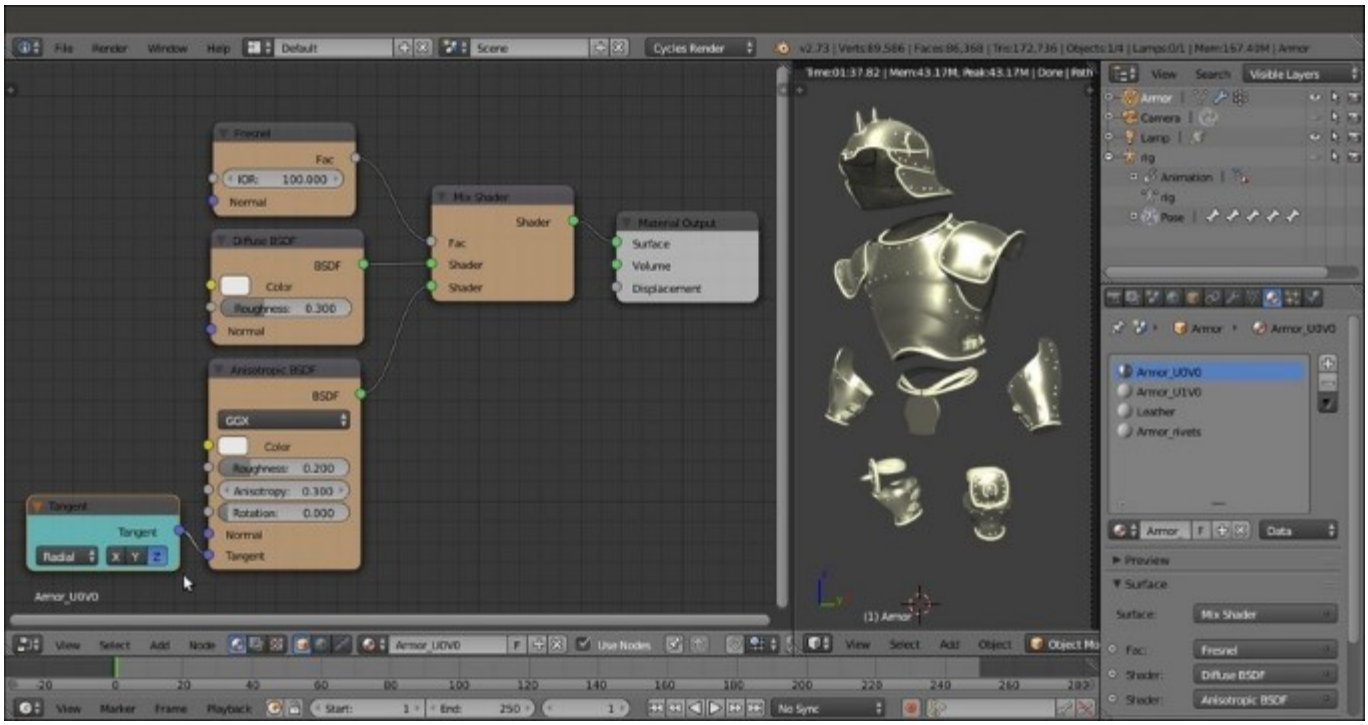
Continuing from the previously saved blend file:

1. Enable the **6th** and the **13th** scene layer and select the **Armor** object in the **Outliner**.
2. Put the mouse pointer in the 3D viewport and press the *0* key on the numpad to go into **Camera** view; fit the window into the field of view.
3. Go to the **Material** window and press the + icon button to the right side to add **four** empty material slots to the **armor**. Select the first material slot and click on the **New** button in the **Surface** subpanel, and rename the material `Armor_U0V0`.
4. Select the **second** material slot, click on the **New** button and rename the material as `Armor_U1V0`; repeat for the **third** slot and rename the material as `Leather` and repeat also for the **fourth** slot and rename the material `Armor_rivets`.
5. Switch the **Node Editor** window temporarily with a **UV/Image Editor** window, then press *Tab* to go into **Edit Mode**; go to the **UV Maps** subpanel under the **Object Data** window to be sure you have the **UVMap** coordinates layer (the first one) as the active one, then enable the *Keep UV and edit mode mesh selection in sync* button on the **UV/Image Editor** toolbar.
6. In the **Node Editor** window, box-select the UV islands of the **U1V0** tile, then in the **Material** window, select the `Armor_U1V0` material and click on the **Assign** button.
7. Still in **Edit Mode**, select all the tiers vertices and then select the `Leather` material, click on the **Assign** button; repeat the operation by selecting all the rivets and assigning them to the `Armor_rivets` material.
8. Disable the *Keep UV and edit mode mesh selection in sync* button on the **UV/Image Editor** toolbar, go out of **Edit Mode** and switch the **UV/Image Editor** window back to the **Node Editor** window.
9. Put the mouse pointer inside the 3D viewport and press *Shift* + *Z* to start the **Rendered** preview.

How to do it…

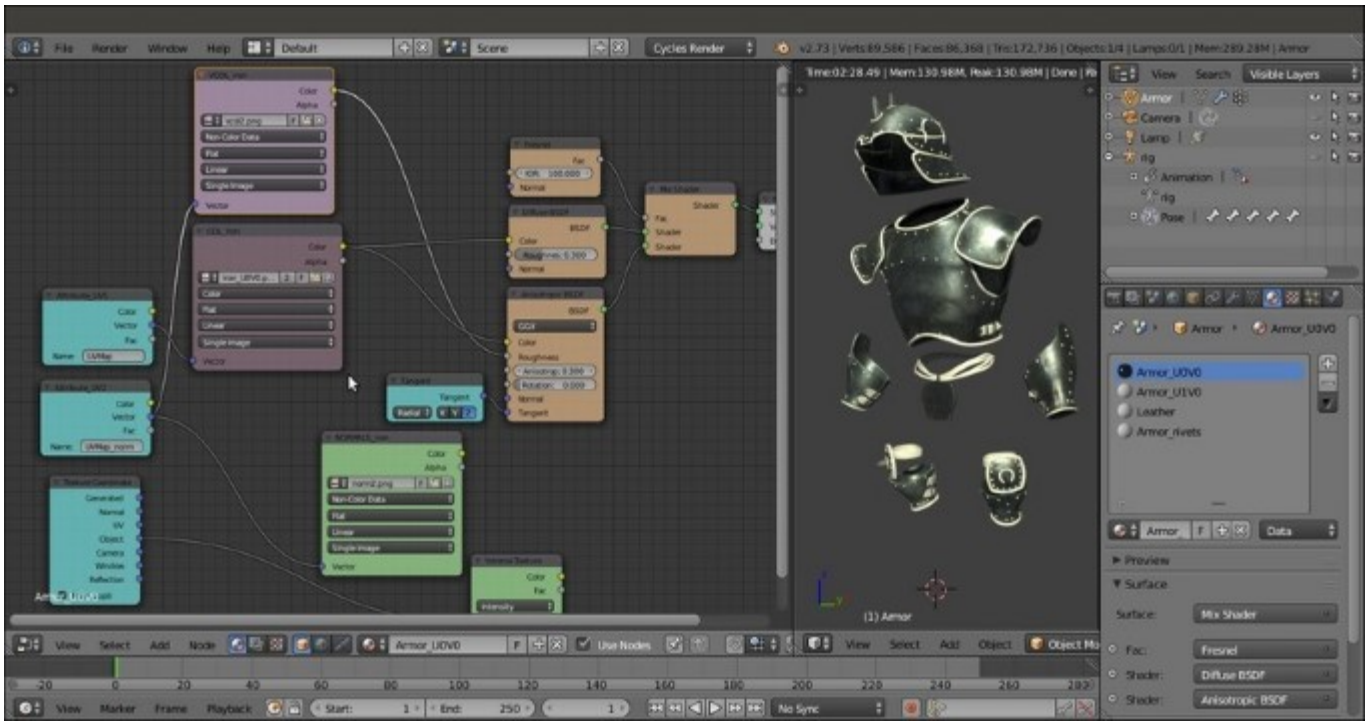
We are first going to create the shader for the metal **plates**:

1. In the **Material** window, select the `Armor_U0V0` material slot.
2. Go to the **Node Editor** window and switch the **Diffuse BSDF** shader node with a **Mix Shader** node; in the first **Shader** slot, select a **Diffuse BSDF** shader node and in the second one, select an **Anisotropic BSDF** shader.
3. Go to the **Node Editor** window and set the **Roughness** of the **Diffuse BSDF** shader to **0.300** and the **Anisotropy** of the **Anisotropic BSDF** shader to **0.300**.
4. Add a **Fresnel** node (*Shift* + *A* | **Input** | **Fresnel**) and connect its output to the **Fac** input socket of the **Mix Shader** node; set the **IOR** to **100.000**.
5. Add a **Tangent** node (*Shift* + *A* | **Input** | **Tangent**) and connect its output to the **Tangent** input socket of the **Anisotropic BSDF** shader node; set the **Tangent** to **Z**.



Starting to build the metal shader for the armor

6. Add a **Frame** (*Shift* + *A* | **Layout** | **Frame**) and parent the nodes, except the **Material Output**, to it, then label it as **SHADERS**.
7. Add three **Image Textures** nodes (*Shift* + *A* | **Texture** | **Image Texture**) and a **Voronoi Texture** node (*Shift* + *A* | **Texture** | **Voronoi Texture**), then add two **Attribute** nodes (*Shift* + *A* | **Input** | **Attribute**) and a **Texture Coordinate** node (*Shift* + *A* | **Input** | **Texture Coordinate**).
8. Label the **Attribute** nodes as **Attribute_UV1** and **Attribute_UV2**. Label the **Image Texture** nodes as **COL_iron**, **NORMALS_iron**, and **VCOL_iron**.
9. Connect the **Vector** output of the **Attribute_UV1** node to the **Vector** input socket of the **COL_iron** node. Connect the **Vector** output of the **Attribute_UV2** to the **Vector** input sockets of both the **VCOL_iron** and **NORMALS_iron** nodes. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Voronoi Texture** node.
10. Click on the **Open** button of the **VCOL_iron** node, browse to the `textures` folder and load the image `vcol2.png`. Set the **Color Space** to **Non-Color Data**. Connect its **Color** output to the **Roughness** input socket of the **Anisotropic BSDF** shader node.
11. Click on the **Open** button of the **COL_iron** node, browse to the `textures` folder and load the image `iron_U0V0.png`. Connect its **Color** output to the **Color** input sockets of the **Diffuse BSDF** and **Anisotropic BSDF** shader nodes.
12. Click on the **Open** button of the **NORMALS_iron** node, browse to the `textures` folder and load the image `norm2.png`. Set the **Color Space** to **Non-Color Data**.
13. Set the **Scale** of the **Voronoi Texture** to **15.000**:

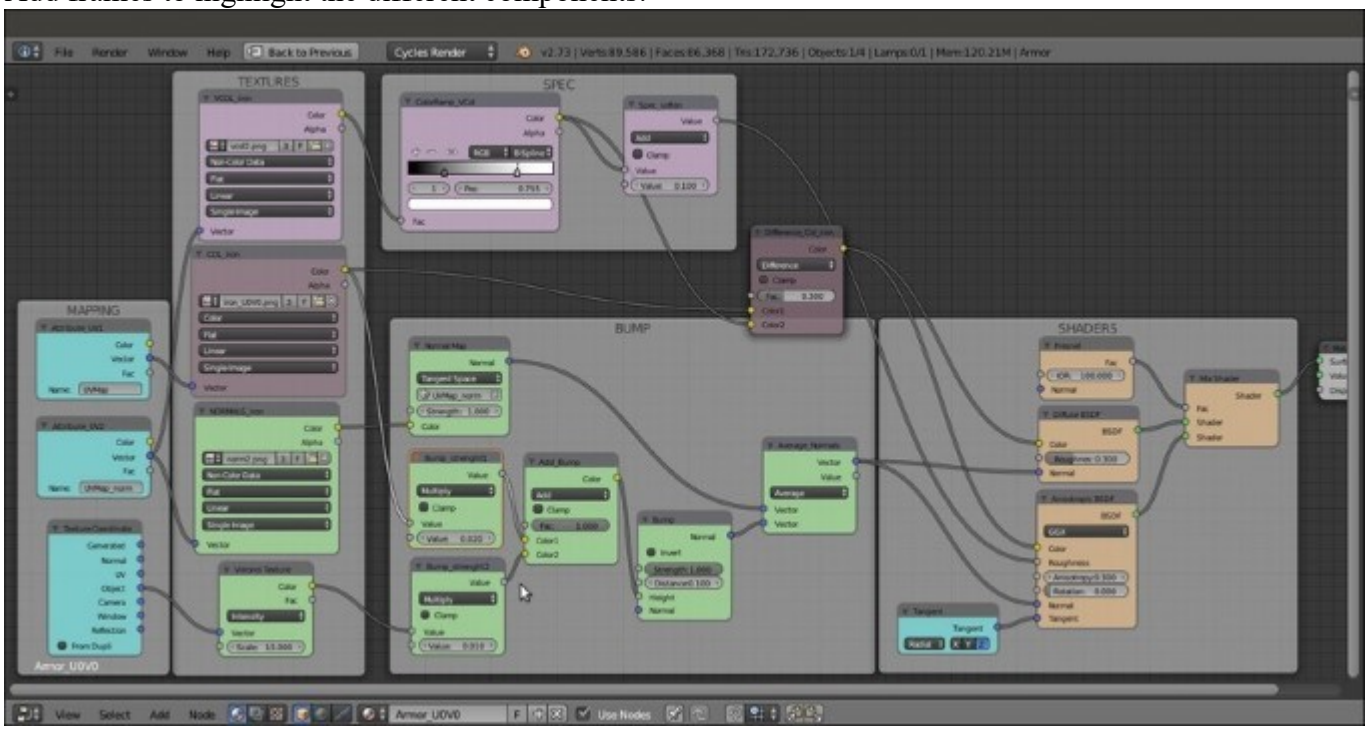


Adding the textures to the "Armor_U0V0" material

14. Add a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**) and a **Math** node (*Shift* + *A* | **Converter** | **Math**). Paste the **ColorRamp** node right after the **VCOL** node, and the **Math** node right after the **ColorRamp**.
15. Label the **ColorRamp** as **ColorRamp_Vcol** and set the **Interpolation** to **B-Spline**, then move the black color stop to position **0.245** and the white color stop to position **0.755**.
16. Label the **Math** node as **Spec_soften** and set the second **Value** to **0.100**.
17. Add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and label it as **Difference_Col_iron**; set the **Blend Type** to **Difference** and the **Fac** value to **0.300**.
18. Connect the **Color** output of the **COL** node to the **Color1** input socket and the **Color** output of the **ColorRamp_Vcol** node to the **Color2** input socket. Connect the **Color** output of the **Difference_Col_iron** node to the **Color** input sockets of the **Diffuse BSDF** and the **Anisotropic BSDF** shader nodes, replacing the old connections.
19. Add a **Normal Map** node (*Shift* + *A* | **Vector** | **Normal Map**), a **Bump** node (*Shift* + *A* | **Vector** | **Bump**), and a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**).
20. Connect the **Color** output of the **NORMALS_iron** node to the **Color** input socket of the **Normal Map** node; click on the empty slot (*UV Map for tangent space maps*) on this latter node to select the **UVMap_norm** item.
21. Connect the **Normal** output of the **Normal Map** node to the first **Vector** input socket of the **Vector Math** node; label this latter as **Average_Normals** and set the **Operation** to **Average**, then connect its **Vector** output to the **Normal** input sockets of the **Diffuse BSDF** and **Anisotropic BSDF** shader nodes.
22. Add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**), label it as **Add_Bump**, set the **Blend Type** to **Add** and the **Fac** value to **1.000**. Connect the **Color** output of the **COL** node to the **Color1**

input socket of the **Add_Bump** node also, and the **Color** output of the **Voronoi Texture** node to the **Color2** input socket.

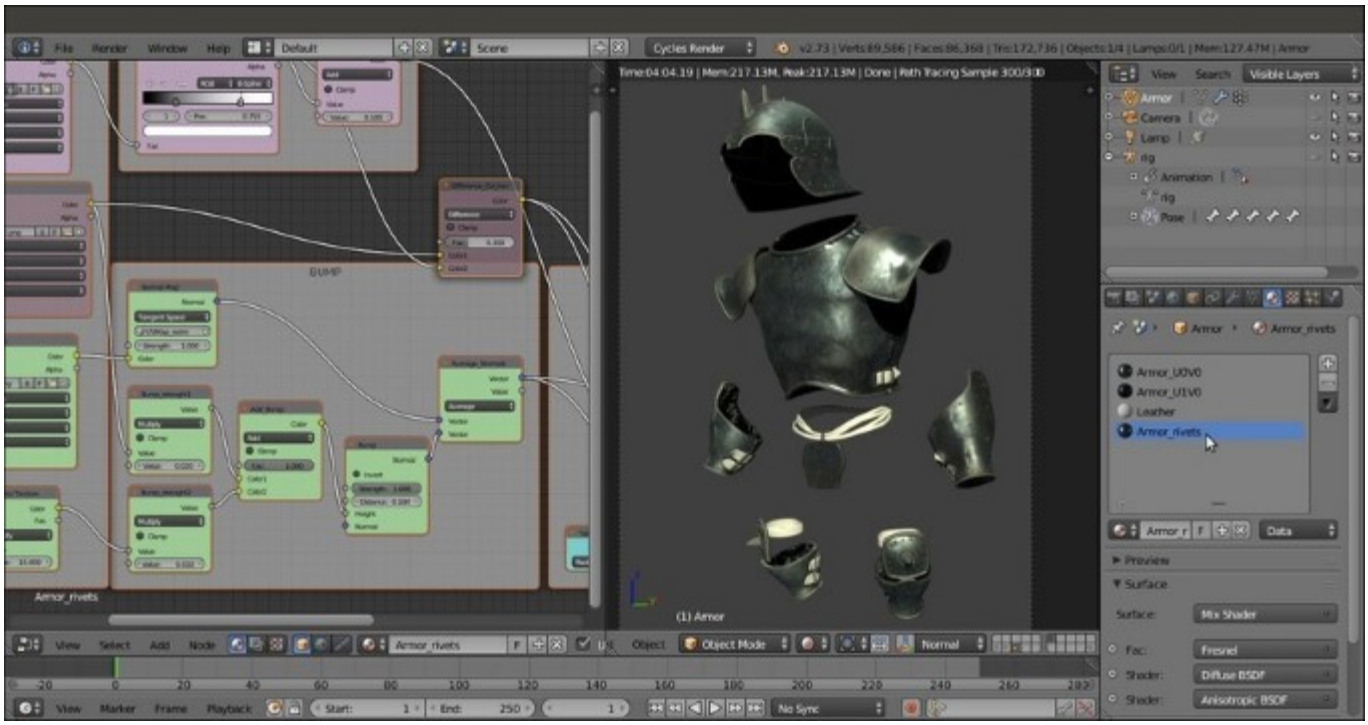
23. Connect the **Color** output of the **Add_Bump** node to the **Height** input socket of the **Bump** node, and the **Normal** output of this latter node to the second **Vector** input socket of the **Average_Normals** node. Set the **Strength** of the **Bump** node to **1.000**.
24. Add two **Math** nodes (*Shift* + *A* | **Converter** | **Math**), label them respectively as **Bump_strength1** and **Bump_strength2**; set the **Operation** to **Multiply** for both, then paste the **Bump_strength1** node between the **COL_iron** and the **Add_Bump** nodes and set the second **Value** to **0.020**. Paste the **Bump_strength2** node between the **Voronoi_Texture** and the **Add_Bump** nodes, and set the second **Value** to **0.010**.
25. Add frames to highlight the different components:



The completed "Armor_U0V0" material

The first **Armor** shader is ready! Now it's very easy to obtain the others:

26. Press *A* twice to select all the nodes, then press *Ctrl* + *C* to copy them.
27. In the **Material** window, select the `Armor_U1V0` material slot and in the **Node Editor** window, delete the default **Diffuse** and **Material Output** nodes; then press *Ctrl* + *V* to paste the nodes copied from the other material.
28. Zoom to the **COL** node and click on the numbered button to the right side of the texture name slot to make it single user, then click on the folder icon button to browse to the `texture` folder and load the image `iron_U1V0.png`.
29. Reselect the `Armor_U0V0` material slot and repeat the step 26 and 27, this time pasting the nodes inside the `Armor_rivets` material slot:



The "Armor_rivets" material and the rendered completed armor

30. Save the file.

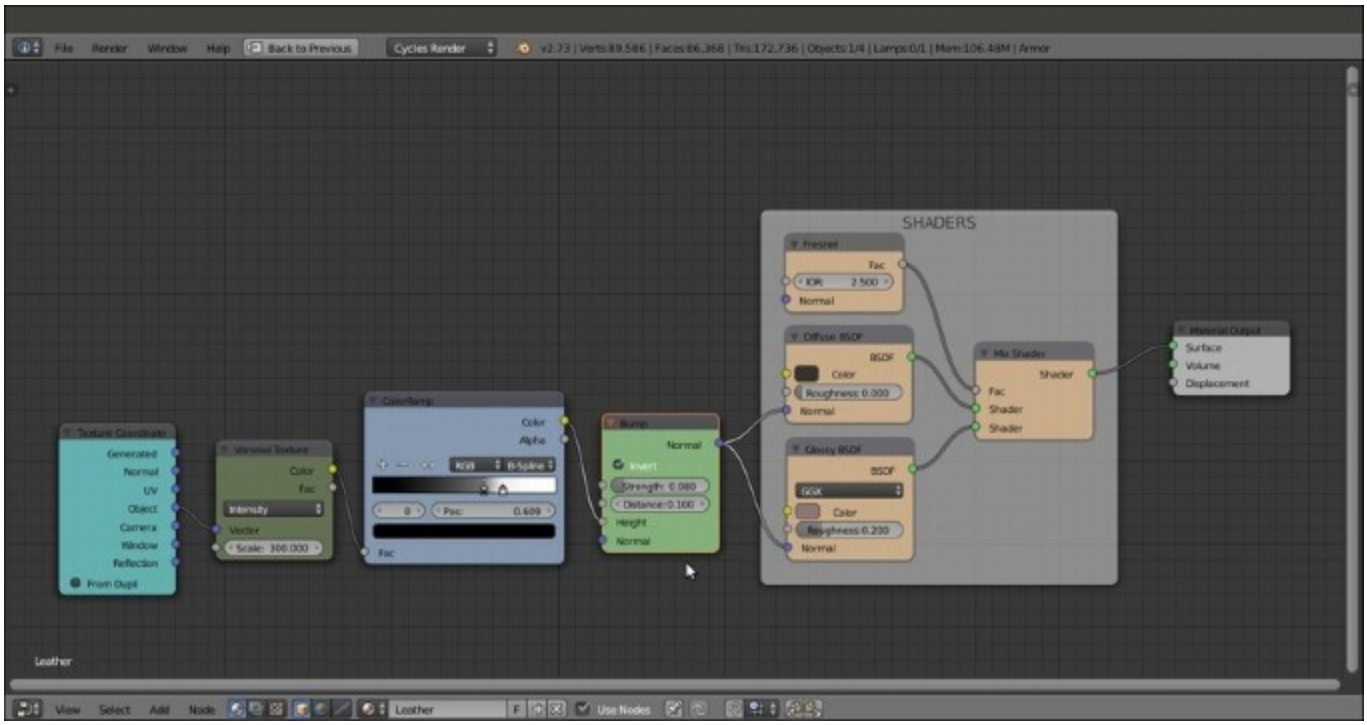
How it works…

The construction of the metallic **armor plates** material follows basically the same scheme we used for the other materials:

- First the shaders were produced, where the metallic look is mainly due to the **Anisotropic BSDF** shader mixed with the diffuse component with a quite high **IOR** value (metals can often have values from **20.000** to **200.000**; we used a midway value of **100.000**).
- The shininess of the metallic surface has been modulated through the output of the `vcol2.png` image, a **Dirty Vertex Color** layer we had previously baked to an image.
- The color of the **Armor** surface has been modulated as well through a **Difference** node with the same `vcol2.png` image.
- The bump pattern works by first adding the **Voronoi** and the **color map** output and then averaging the result with the **normal map** output.

There's more…

The last material created for our character is a very simple leather material made mainly from the output of a **Voronoi Texture** node, contrasted, inverted, and used as bump pattern:



The simple "Leather" material

This completes the creation of the **Gidiosaurus** shaders in **Cycles**:

The completed Gidiosaurus character in Cycles

Of course, reflecting materials, for example, the metallic **armor** surface or the **corneas** (but to some extent also the reptile **skin**), need something to reflect to show them at their best; we'll see this in the last chapter of this cookbook.

In the next chapter, which is the penultimate chapter, we'll see the creation of the same materials in **Blender Internal**.

Chapter 13. Creating the Materials in Blender Internal

In this chapter, we will cover the following recipes:

- Building the reptile skin shaders in Blender Internal
- Building the eyes' shaders in Blender Internal
- Building the armor shaders in Blender Internal

Introduction

In this chapter we'll see how to set up the materials for the **Gidiosaurus** and the **Armor** in the **Blender Render** engine; in fact, although not exactly of the same quality as in **Cycles**, it is also possible to obtain quite similar shader results in **Blender Internal**:

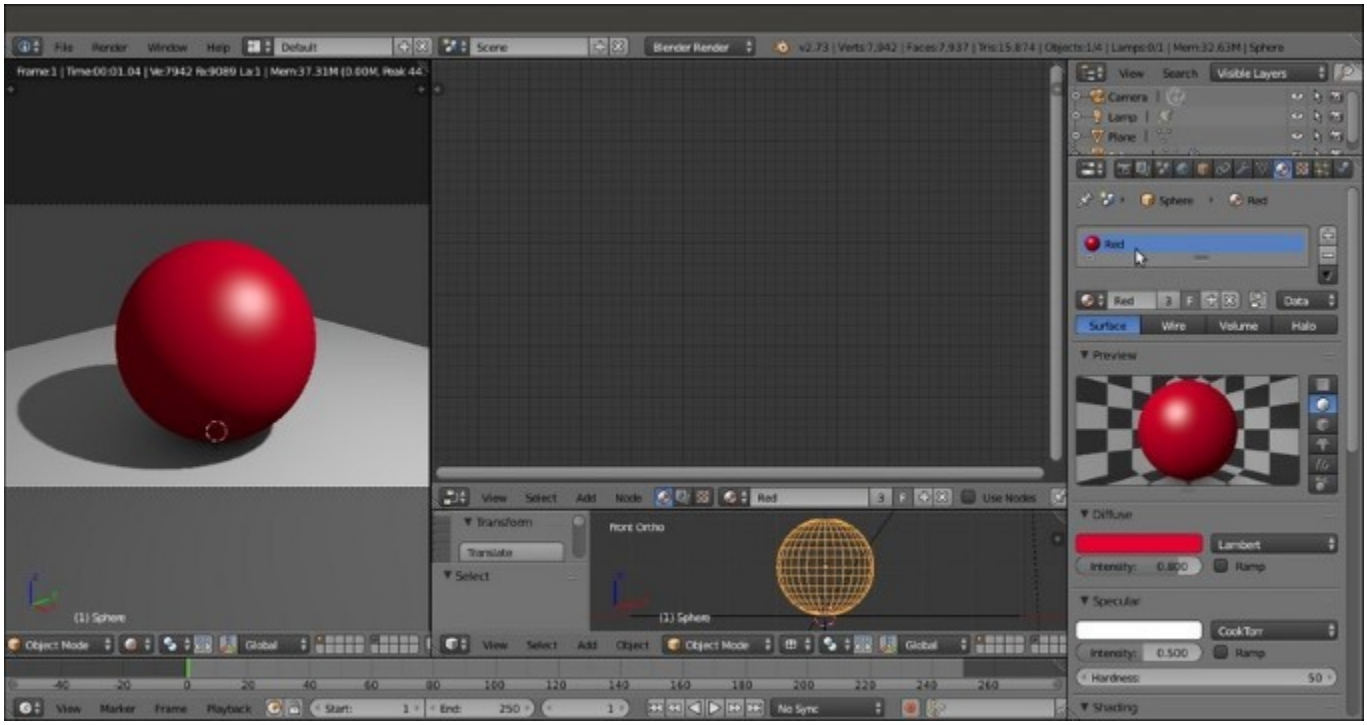
Comparison of the Gidiosaurus character rendered in Cycles (left) and Blender Internal (right)

If you are wondering why we should re-do in the **Blender Render** engine, which is quite old and no longer developed and/or supported, the same thing we have already done in **Cycles**, there are several possible reasons: for example, no doubt **Cycles** is superior in quality but, compared with the scanline **BI**, its rendering is (and, being a path-tracer, always will be) slower; even with the aid of a render-farm, rendering times are still a *money* issue in the production of animations.

The previous screenshot shows, for comparison, only the top parts of two full shot renderings of the **Gidiosaurus** character: the **Cycles** rendering to the left took around **1** hour and **20** minutes (**1920** × **1080** resolution CPU rendering with *Intel Core 2 Duo T6670 2.20 GHz* and **4** GB of RAM, in Ubuntu 12.04 64-bit); the **Blender Internal** rendering to the right took only **26** minutes.

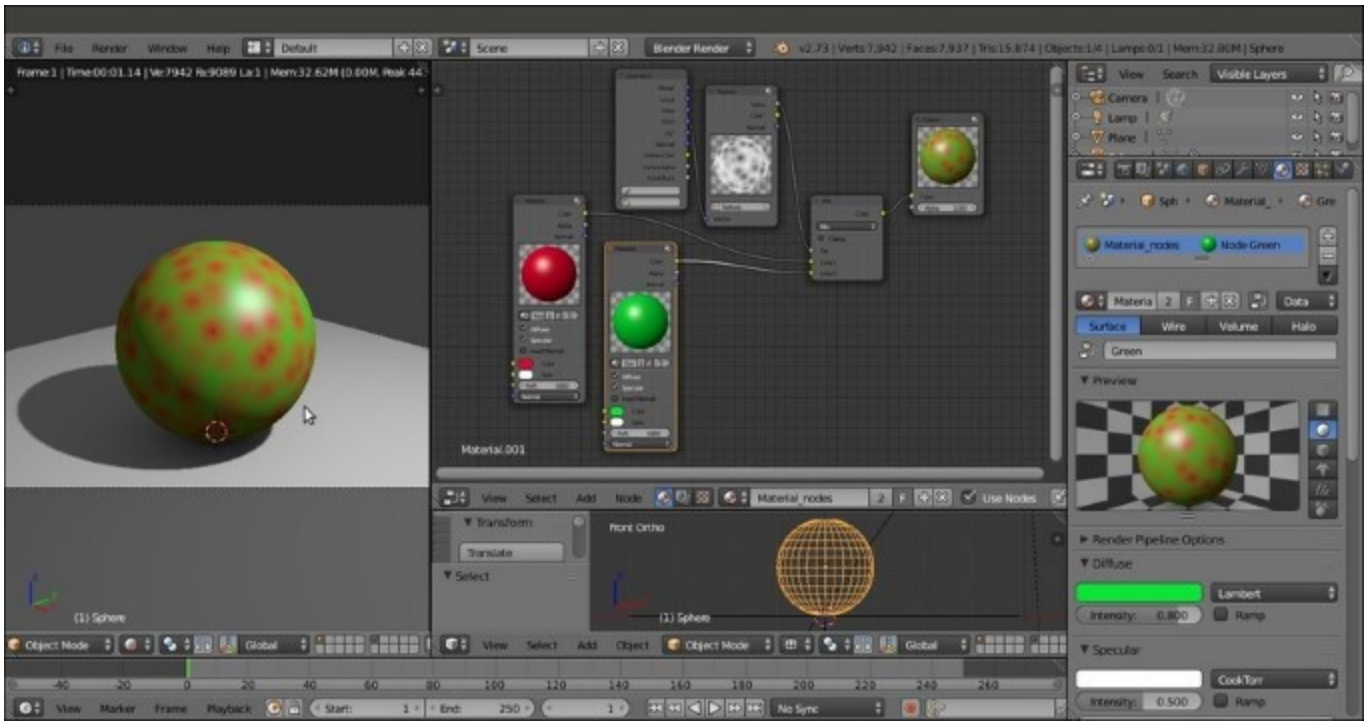
One other reason is that **Cycles'** normals baking capabilities are still not as good as in **Blender Internal** (at the moment, it bakes only the real geometry, contrary to **Blender Internal**, which can also bake the bump output of textures to normal maps), or that it's not as flexible for **Non-Photorealistic Rendering** (**NPR**) as the **Blender Render** engine.

Just a quick note: normally, materials under the **Blender Render** engine are created directly in the slots inside the **Material** window, often switching to the **Texture** window and back; in the following screenshot, you can see the **Rendered** preview of a generic `Red` *mono* material assigned to a **UV Sphere**:



A generic "mono" Blender Internal material

But, it's also possible to use node materials in **Blender Internal**, created and connected inside the **Node Editor** window; basically, let's say that *two or more materials can be mixed through nodes* to obtain more advanced results. In the following screenshot, for example, the mono `Red` material is mixed with a mono `Green` material through the output of a **Voronoi** texture connected to the **Fac** input socket of a **MixRGB** node:



Two mono materials mixed in the Node Editor window

This is the way we are going to create the **Blender Internal** shaders.

Building the reptile skin shaders in Blender Internal

Because we want to keep the materials we already created for **Cycles** in the same blend file (and the reason will be clear in the next chapter), before we start with the creation of the **Blender Internal** shaders, we must prepare the file a bit.

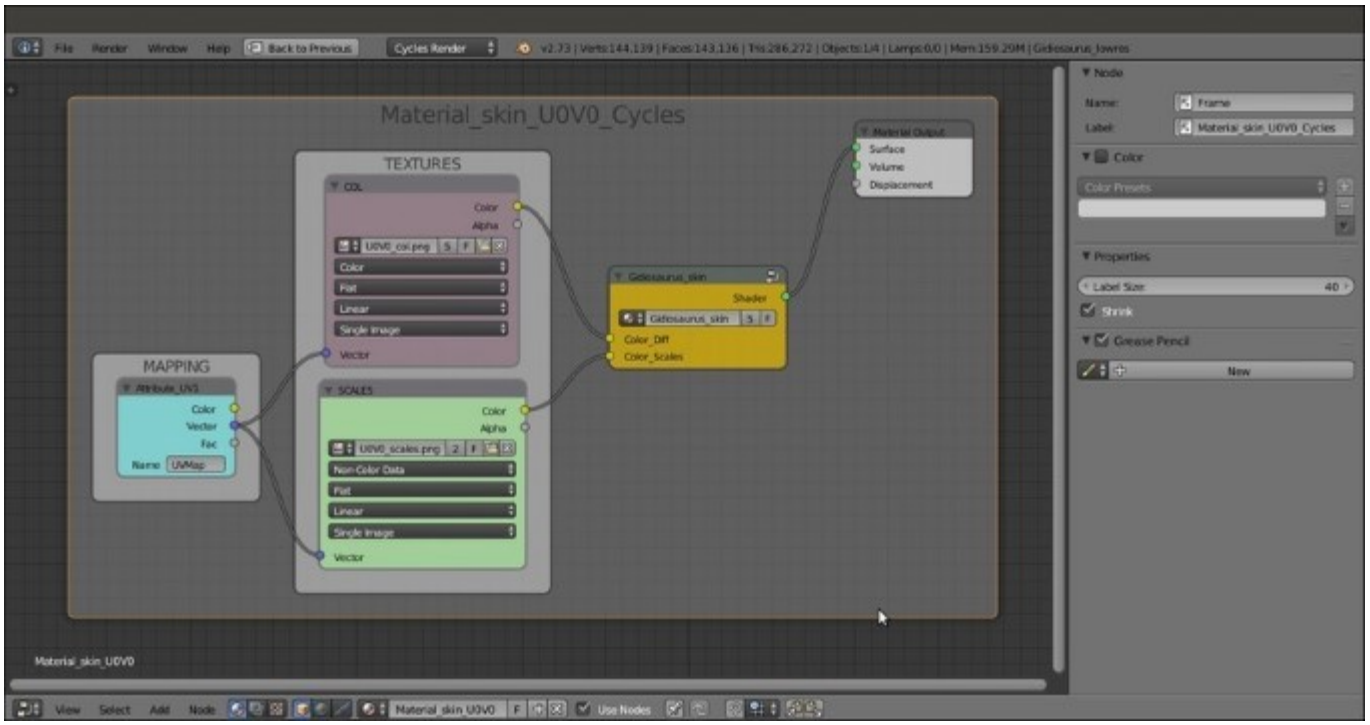
Getting ready

The first thing to do is to open the last saved blend file, add **Frames** to each material in the **Node Editor** window, and label them with the material name followed by the suffix **_Cycles**; this is to later distinguish them from the material we will build for **BI**.

Therefore:

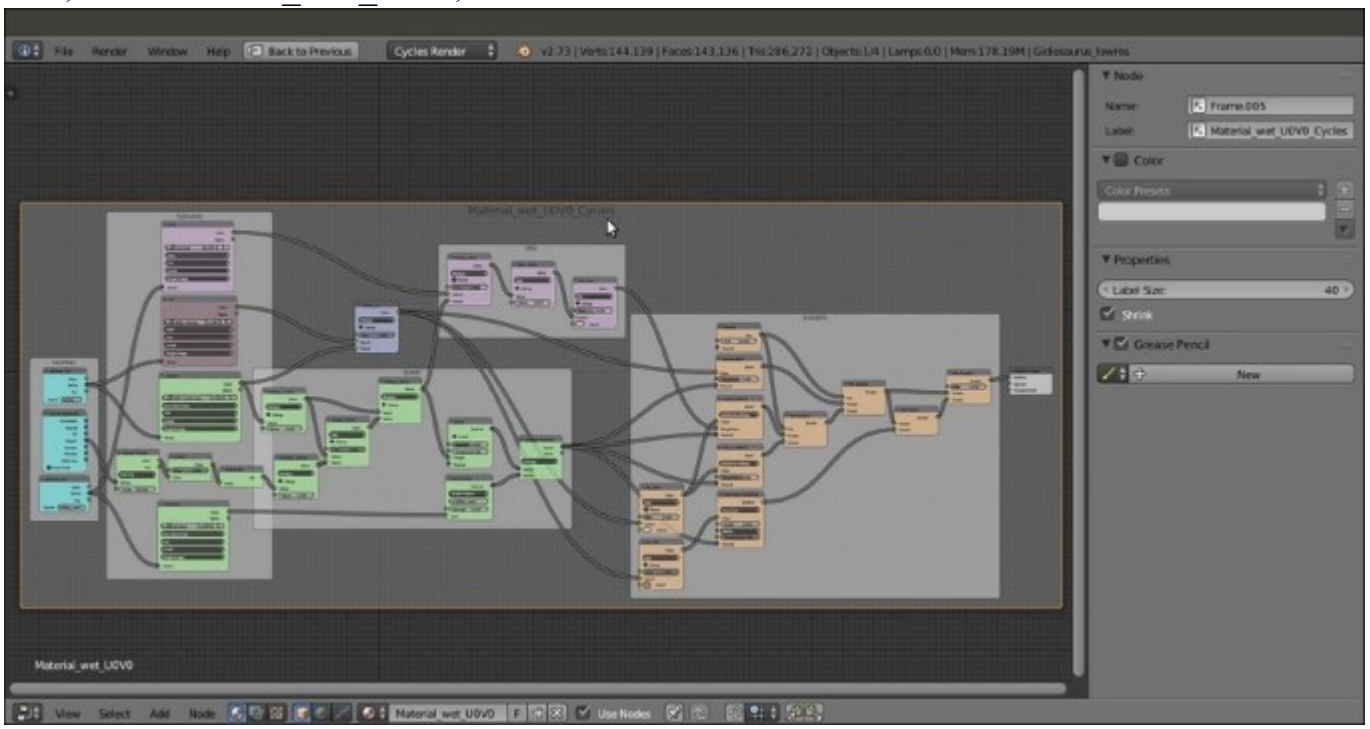
1. Start Blender and load the `Gidiosaurus_shaders_Cycles.blend` file.
2. In the **Outliner**, select the **Gidiosaurus_lowres** mesh, go to the **Material** window and click on the `Material_skin_U0V0` slot; put the mouse pointer inside the **Node Editor** window and press *Shift* + *A* to add a **Frame** (*Shift* + *A* | **Layout** | **Frame**).
3. Press *A* to select all the nodes (the added **Frame**, already selected, becomes the active one) and then press *Ctrl* + *P* to parent them to the active **Frame**.
4. Select only the **Frame** and press *N* to call the **Properties** sidepanel; in the **Label** slot under the **Name** subpanel, type **Material_skin_U0V0_Cycles**, then go down to the **Properties** subpanel and increase the **Label Size** to **40**.
5. Repeat the procedure for all the Cycles' **Gidiosaurus_lowres** materials, for the **Eyes** and **Corneas** and for the **Armor** materials.

   So, for example, the `Material_skin_U0V0`, in the **Node Editor** window, becomes this:



A "framed" Cycles material

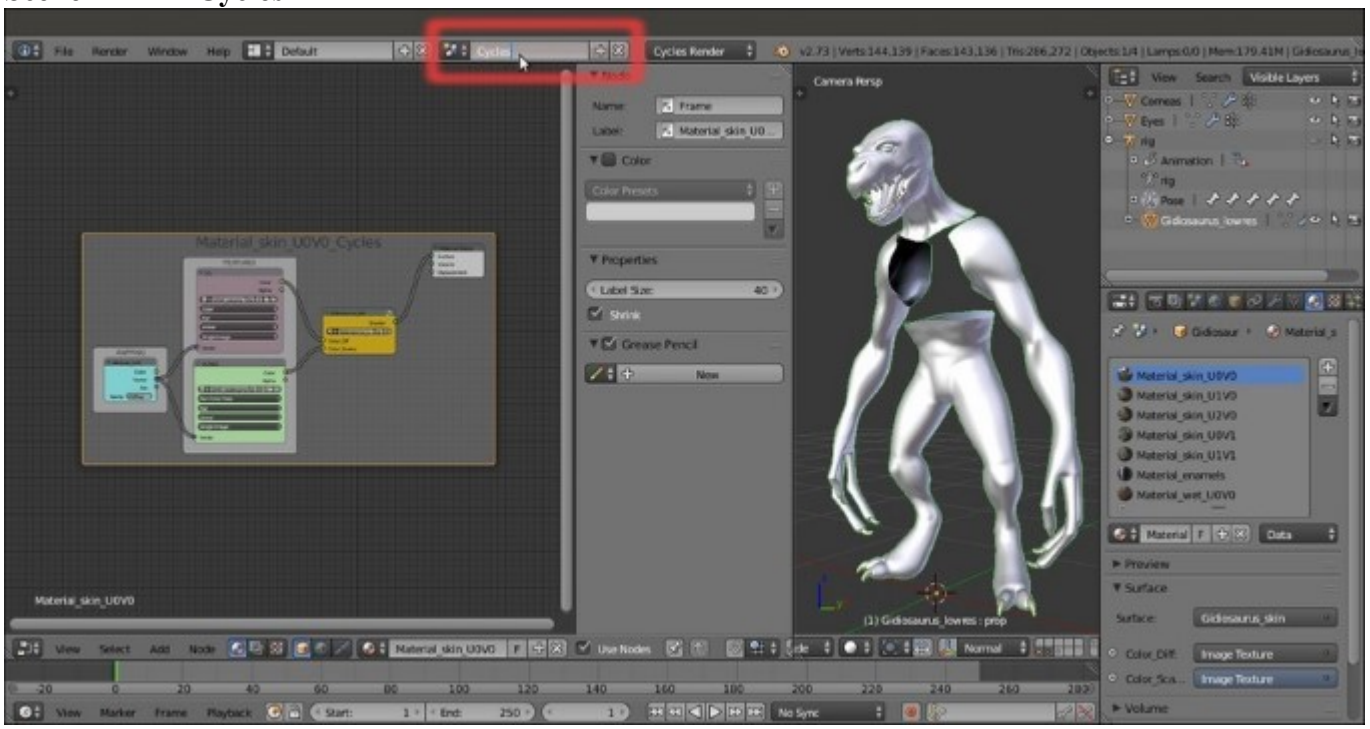
Also, the `Material_wet_U0V0`, becomes this:



Another "framed" Cycles material

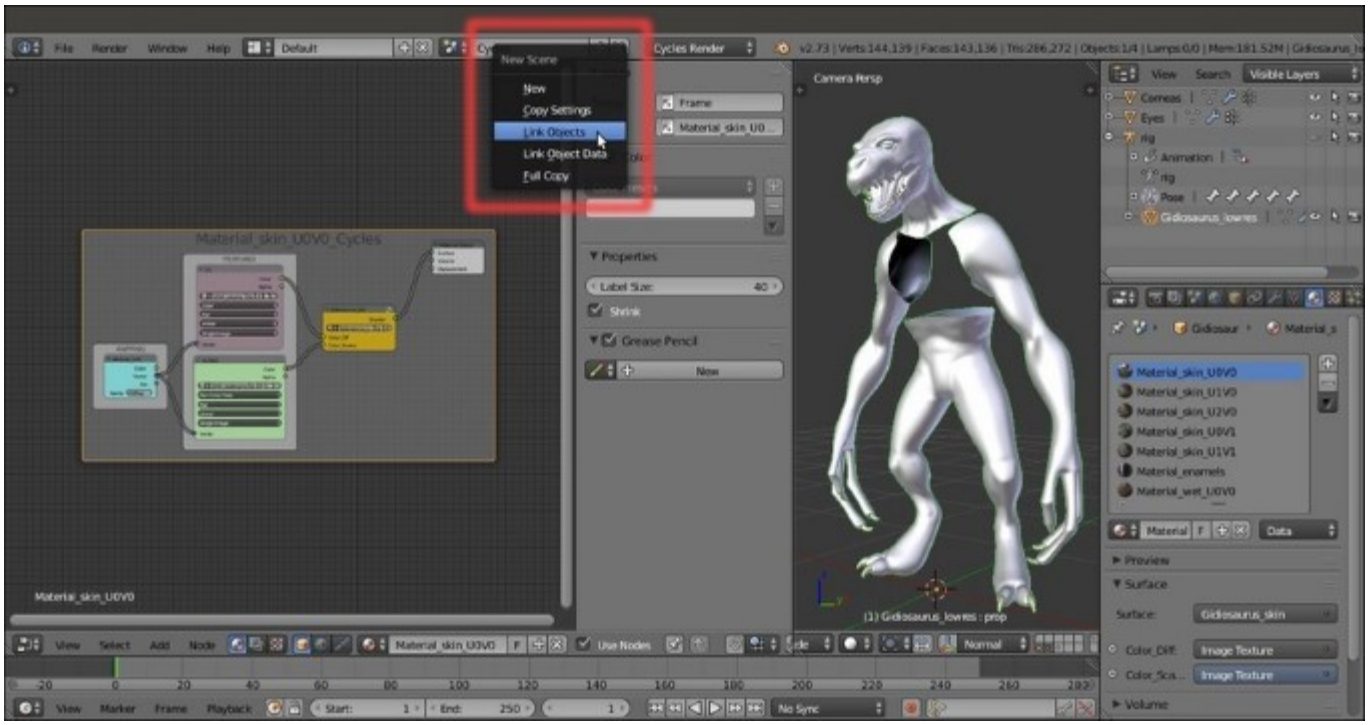
Note that the name of the material is the same as before, the only difference is that a **Frame** labeled with the **_Cycles** suffix has been added in the **Node Editor** window to visually group all the Cycles' nodes that are a constituent of the shader.

6. Now go to the *Scene data block* button on the main top header; left-click on it and rename the **Scene** label as **Cycles**:



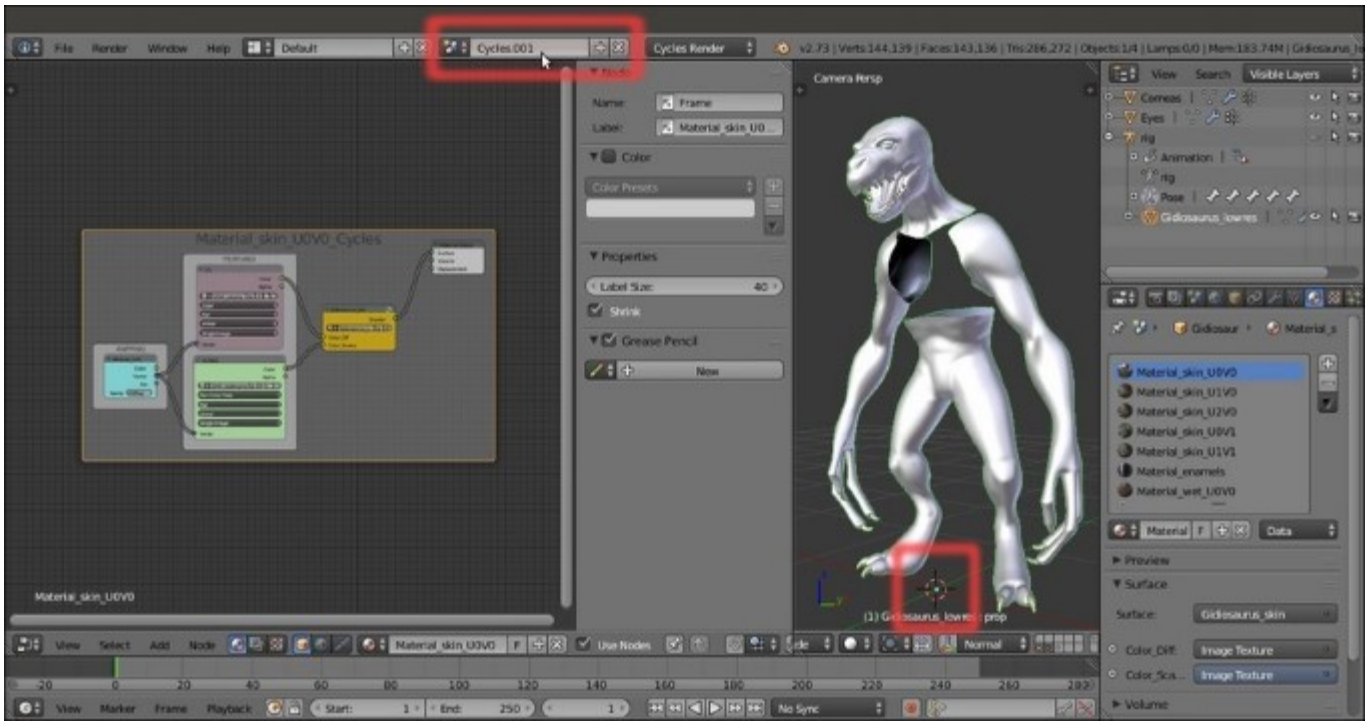
Renaming the Scene label

7. Click on the + icon button to the right of the datablock name; in the pop-up little **New Scene** panel, select the **Link Objects** item:



Adding a new scene with linked objects

At this point we have created a new scene (automatically labeled as **Cycles.001**) that is sharing the same objects of the other (**Cycles**) scene (be aware of this: the objects in one scene are not a copy of the others, *they are the same objects shared/linked between the two scenes*); you can say which objects are actually linked from one scene to another, by their blue pivot point (for example, look at the highlighted pivot point of the **Gidiosaurus_lowres** object in the following screenshot):

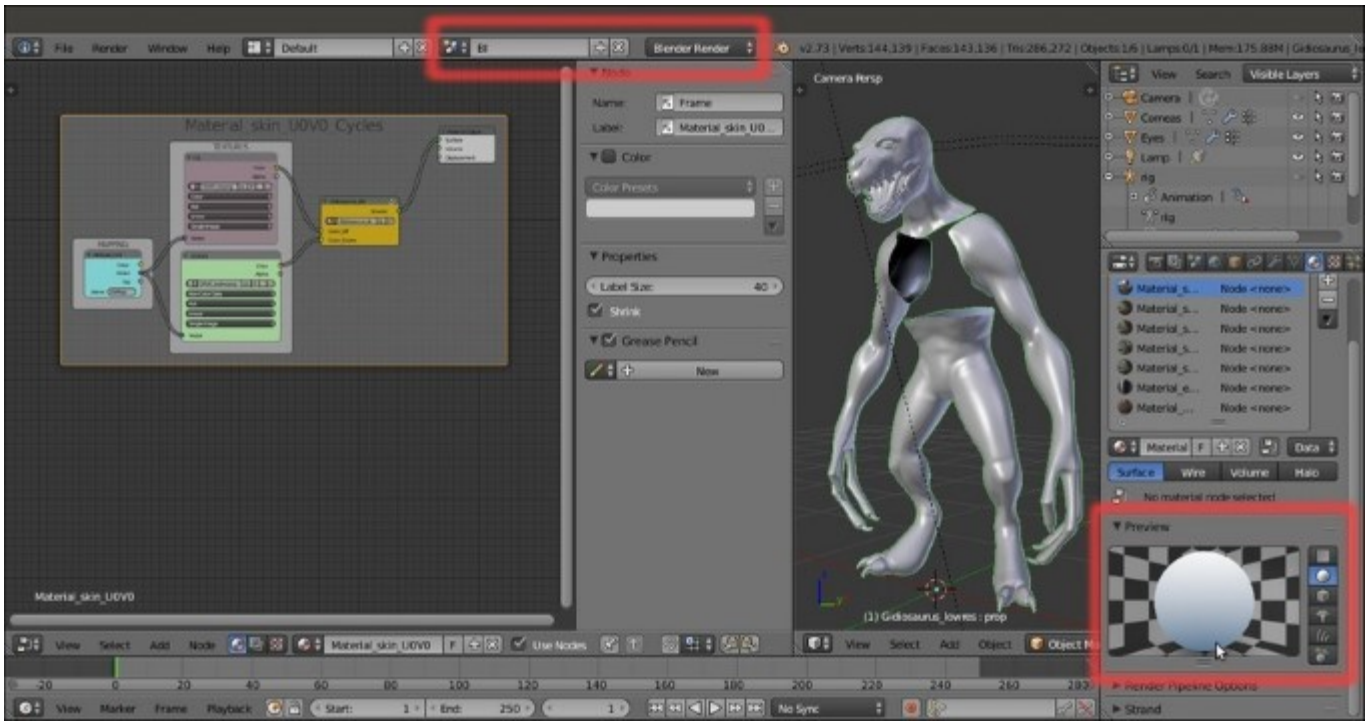


A new scene with linked objects

The advantages of creating new scenes with linked objects are obvious: we can have totally different rendering engines, or different worlds or lamps, in the different scenes and use the same objects and meshes data; so, for example, any modification to a linked object in one scene will automatically be transferred to the other scenes.

Furthermore, avoid duplicating the objects for each scene; this will help to keep a small file size.

8. Rename the scene from **Cycles.001** to **BI**, then move to the *Engine to use for rendering* button a bit to the right and switch from **Cycles Render** to **Blender Render**.



Switching to the Blender Render engine and the "empty material" preview

Note

Note that the **Preview** subpanel of the **Material** window shows an *empty material*, to point out that under the current **Blender Render** engine, the material slot, although filled with the **Cycles** material, doesn't have anything to render yet.

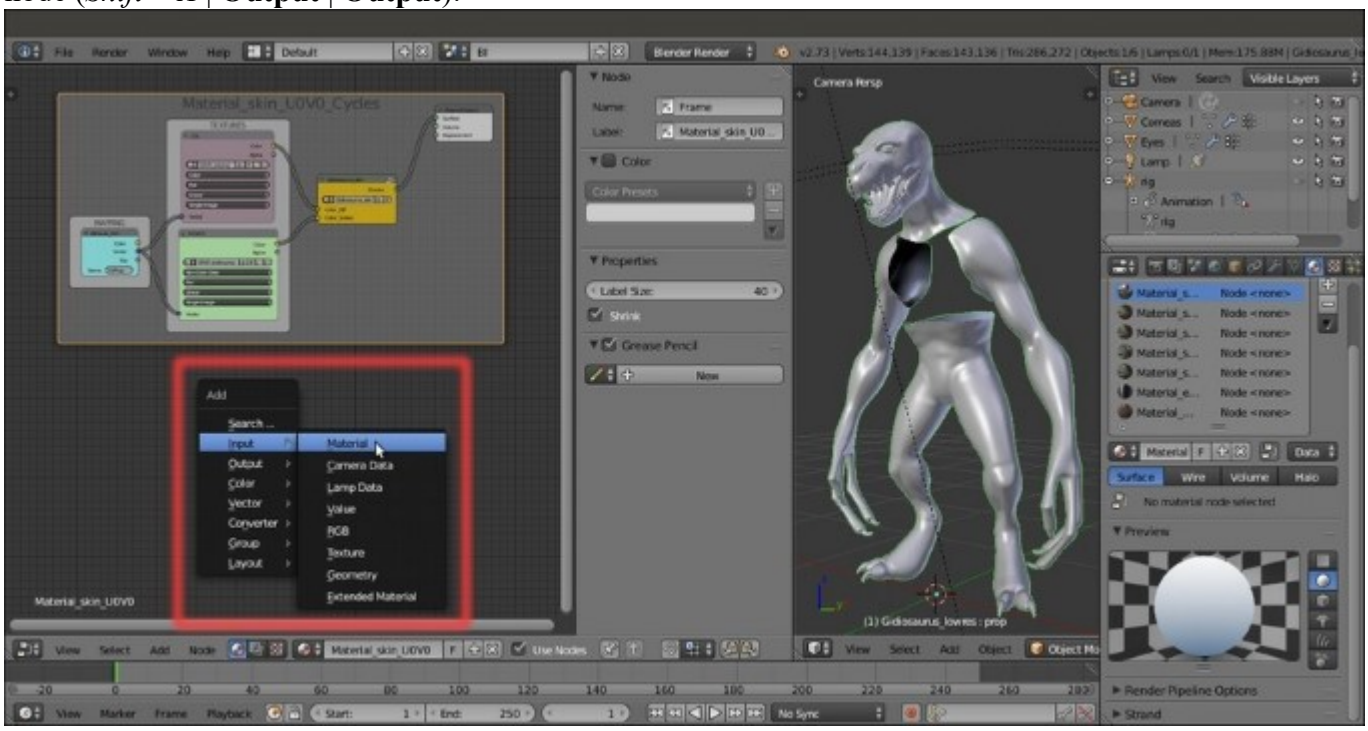
9. In the **Outliner**, select the **Lamp** (be sure to have enabled both the **11th** and the **6th** scene layers); go to the **Object Data** window, set the energy to **14.000** and the color to **R 1.000**, **G 1.000**, **B 0.650**; under the **Shadow** subpanel, enable the **Buffer Shadow** item, **Filter Type** to **Gauss**, **Soft** = **12.000**, **Size** = **4000**, and **Samples** = **16**. Set **Clip Start** = **9.000** and **Clip End** = **19.000**.
10. Go to the **World** window and enable the **Ambient Occlusion** by checking the item in the subpanel of the same name; leave the **Blend Mode** to **Add** and set the **Factor** to **0.35**.
11. Go further down to the **Gather** subpanel and click on the **Approximate** button: check the **Pixel Cache** item and then check also the **Falloff** checkbox under the **Attenuation** item; set the **Strength** to **0.900**.
12. Enable the **Indirect Lighting** item just above and set the **Factor** to **0.65**.

    These **World** settings are to obtain a sort of **Global Illumination** effect in the **Blender Render** engine; to learn more, have a look at http://www.blender.org/manual/render/blender_render/world/index.html.
13. Save the file as `Gidiosaurus_shaders_Blender_Internal.blend`.

How to do it…

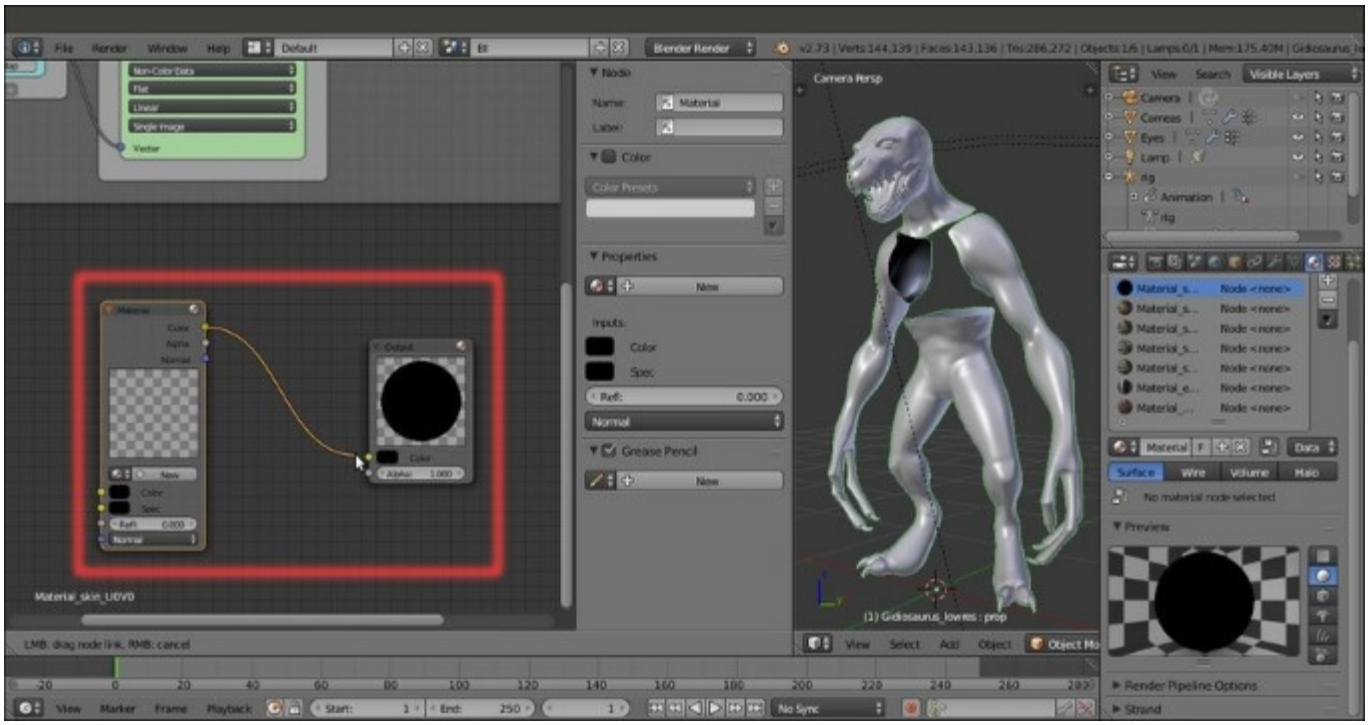
Let's start with the first top **Gidiosaurus** skin material, so:

1. Be sure to have the **Gidiosaurus_lowres** object selected and, back in the **Material** window, click on the `Material_skin_U0V0` slot.
2. Put the mouse pointer inside the **Node Editor** window and press *Shift* + *A* (*Shift* + *A* | **Input** | **Material**) to add a **Material** node to the window; then press again *Shift* + *A* and add an **Output** node (*Shift* + *A* | **Output** | **Output**):



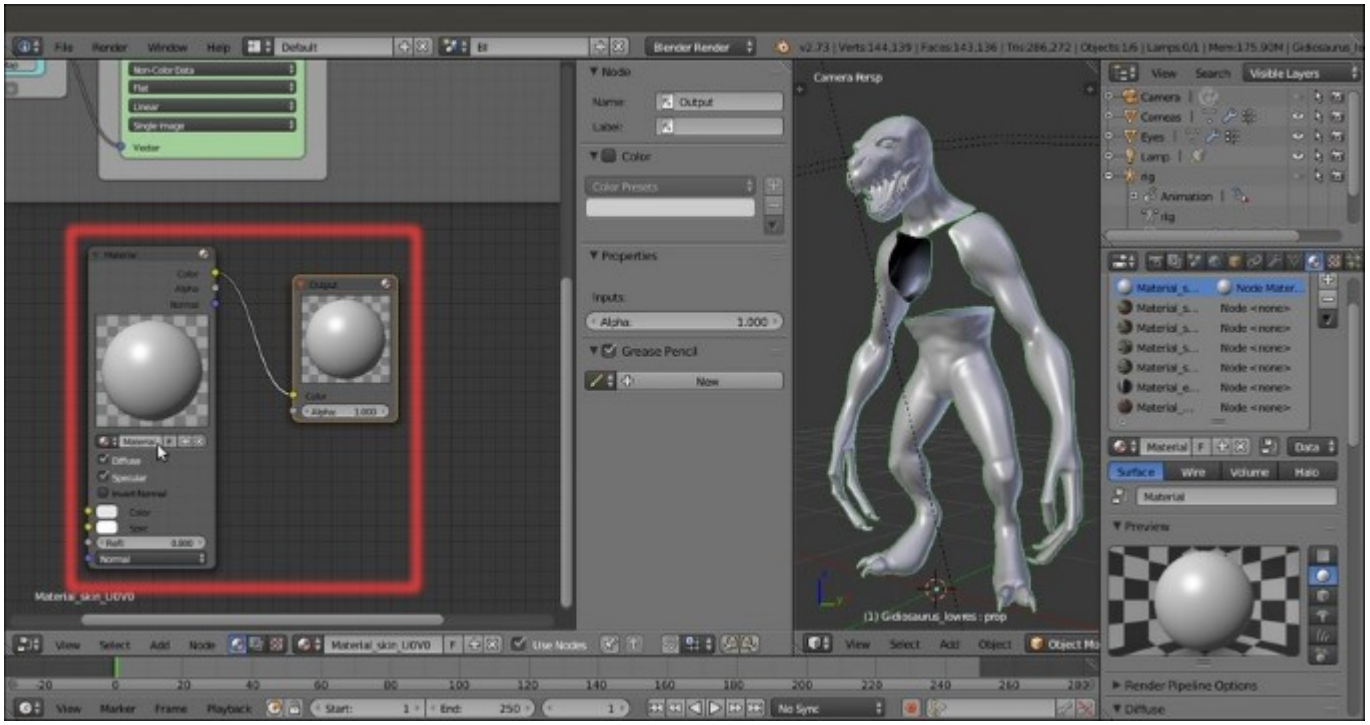
Adding a first material node in the Node Editor window

3. Connect the **Color** output of the **Material** node to the **Color** input socket of the **Output** node:



Connecting the material node to the output node

4. Now click on the **New** button on the **Material** node to create a new default **Blender Internal** material:



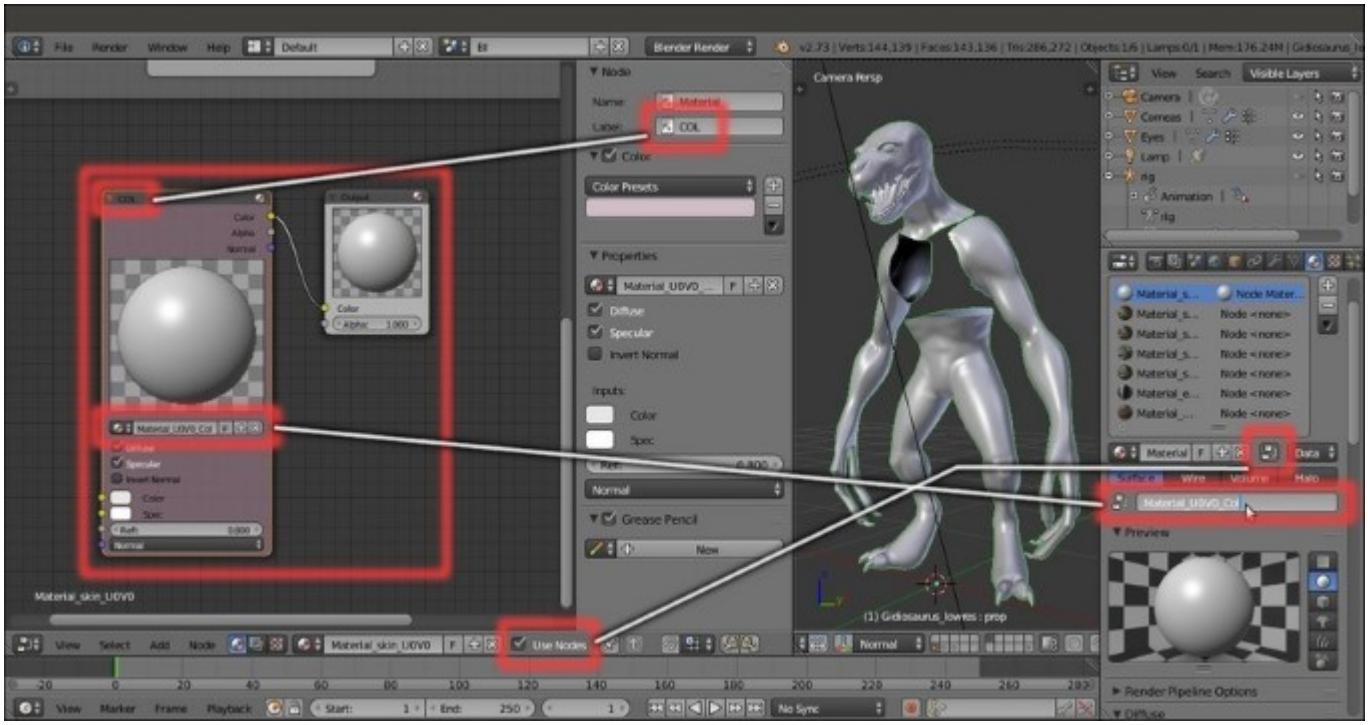
Creating a default "mono" material by clicking on the New button in the material node

5. In the **Properties** sidepanel of the **Node Editor** window (*N* key to call it) label the **Material** node as **COL** and assign a color.

   If you look now at the **Material** window, close to the right side of the material datablock (the name of the material), there is an already enabled and squared button with the symbol of the nodes.

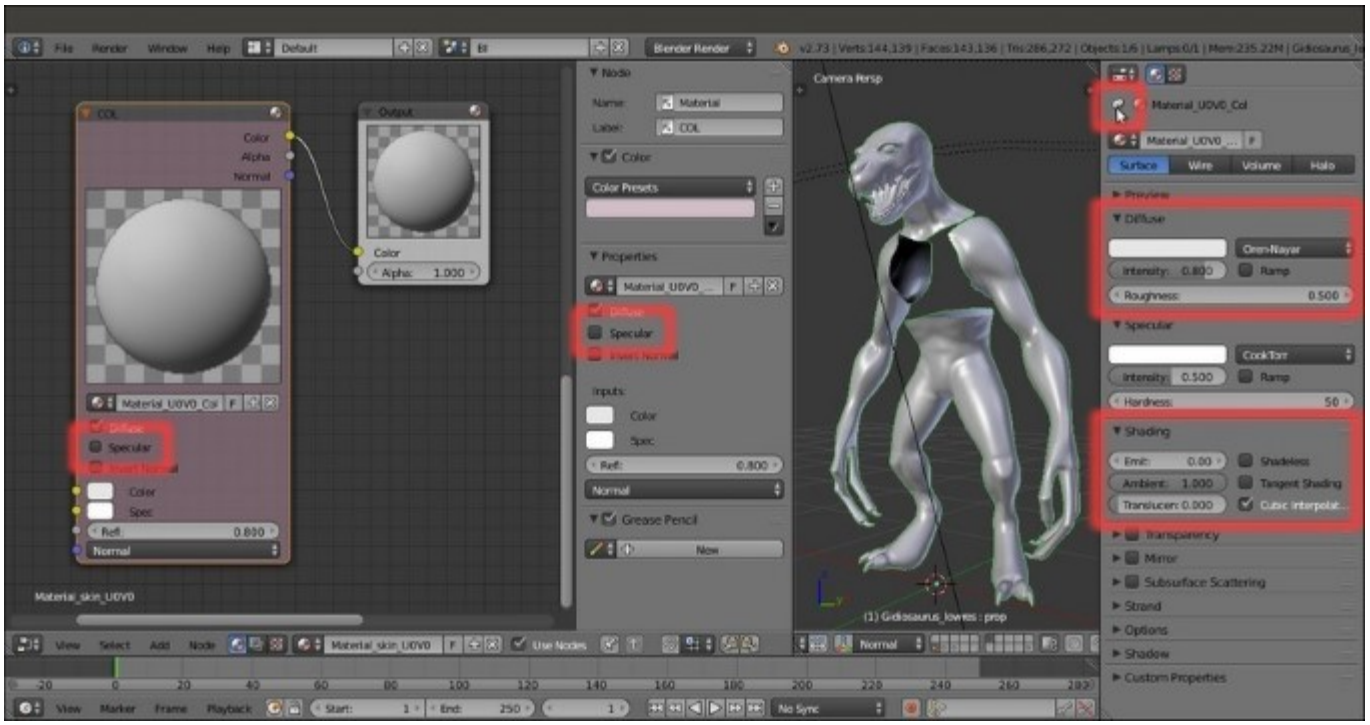
   In our case, that button is already enabled because we are already using material nodes; because it's enabled, a second material datablock slot has appeared just further down: that's the datablock slot for any node selected inside the **Node Editor** window and that is part of a material node.

   The purpose of this second datablock slot is to let us know which material is the selected one and we are therefore going to edit it by tweaking all the values in the subpanels below.
6. Go to the **Material** window to find the second material name slot: rename the material selected in the **Node Editor** window as `Material_U0V0_Col`; you can do the same thing by clicking on the name datablock on the **COL** node interface.



The corresponding datablock slots in the node interface and in the Material window

7. In the **Node Editor** window, or in the *N* **Properties** sidepanel, deselect the **Specular** item.
8. Go to the top of the **Material** window and click on the pin icon to the left of the contest; by doing this only the selected material is shown in the window.
9. Go to the **Diffuse** sidepanel and click on the **Diffuse Shader Model** button to select the **Oren-Nayar** item; then go down to the **Shading** subpanel and enable the **Cubic Interpolation** item:

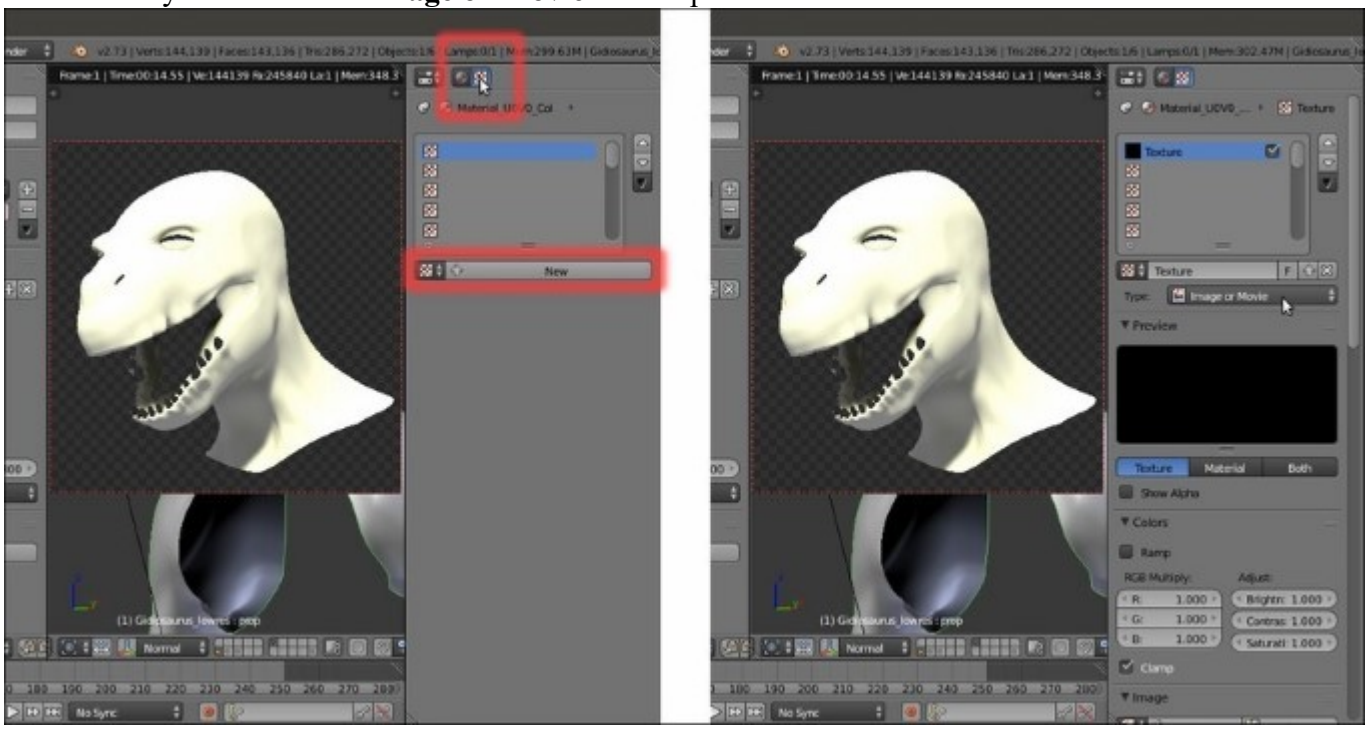


The Specular item to be disabled in the Node Editor window and the shader's parameters to be tweaked in the Material window

10. At this point, press *Shift* + *B* to draw a box around the character's head in the **Camera** view and then zoom to it. If your computer is powerful enough to allow you to work without slowing down, put the mouse pointer inside the 3D viewport and press *Shift* + *Z* to start the **Rendered** preview; in any case, you can easily enable or disable the preview every time you need it:

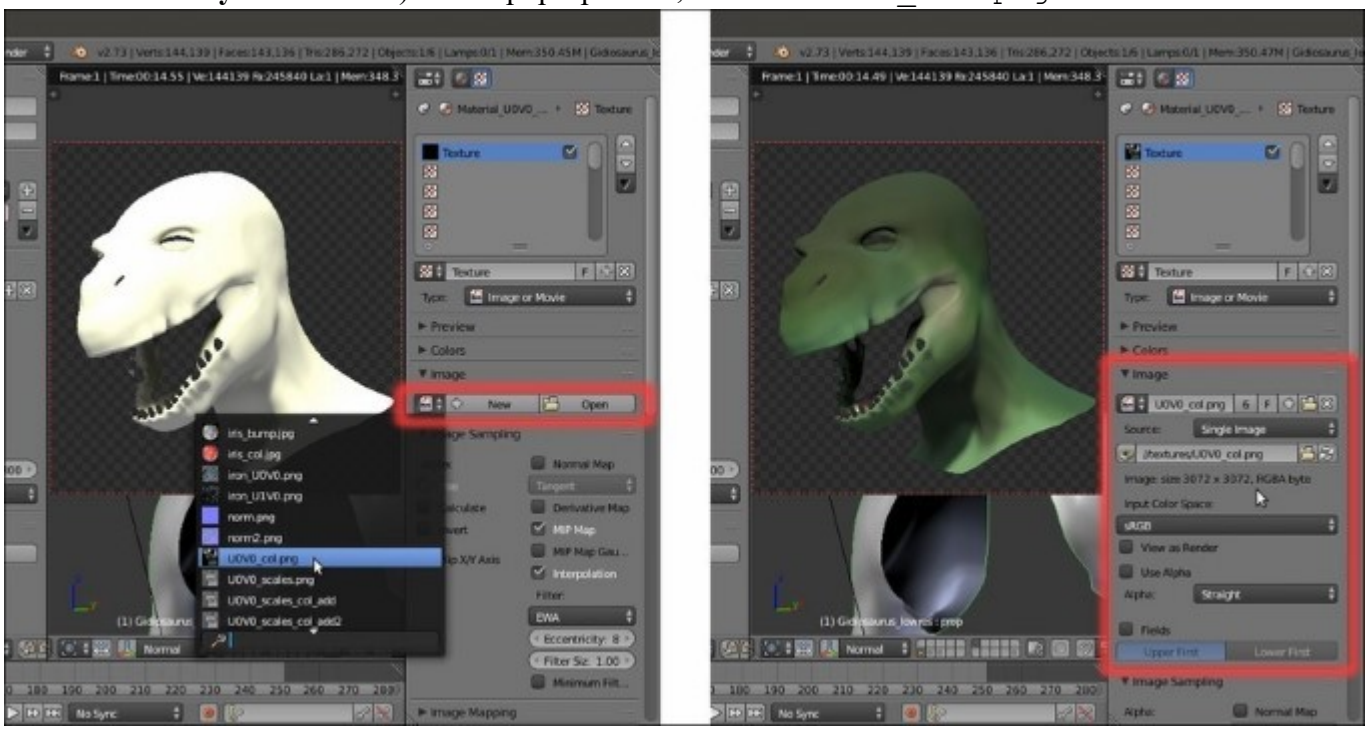
Cropping and starting the rendered preview

11. Click on the **Texture** window icon at the top right of the main **Properties** panel, just above the contest, be sure to have the **first** top texture slot selected and click on the **New** button to automatically load a default **Image or Movie** texture panel:



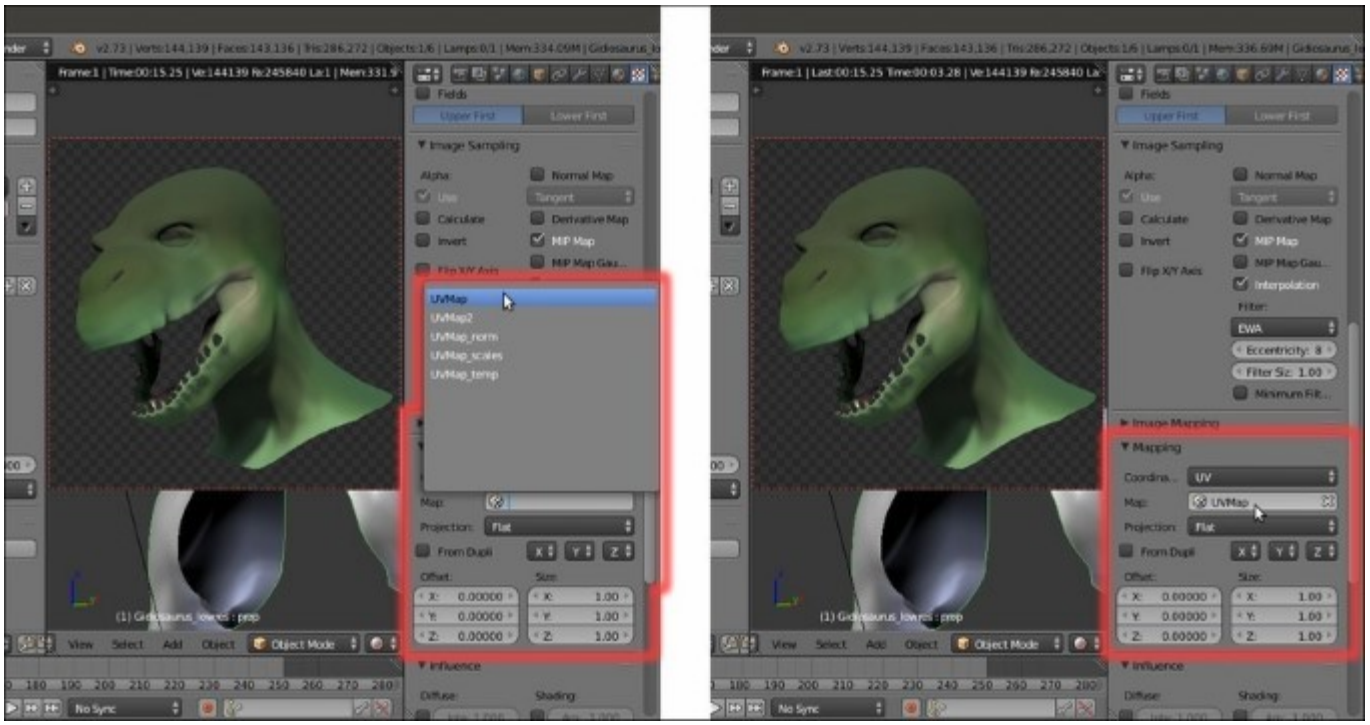
Adding a first texture slot to the material

12. Collapse the **Preview** and the **Colors** subpanels, which at this moment we don't need, and click on the double little arrows to the left side of the **New**/**Open** buttons in the **Image** subpanel (remember that we have already loaded inside the blend file all the image textures we need, because of the **Cycles** shaders!): in the pop-up menu, select the `U0V0_col.png` item:



Selecting the right image texture from the drop-down list

13. Go further down to find the **Mapping** subpanel: be sure to have the **Coordinates** set to **UV**, the **Projection** to **Flat** (default settings) and click on the **Map** empty slot to select the **UVMap** item.



Selecting the right UV coordinates mapping

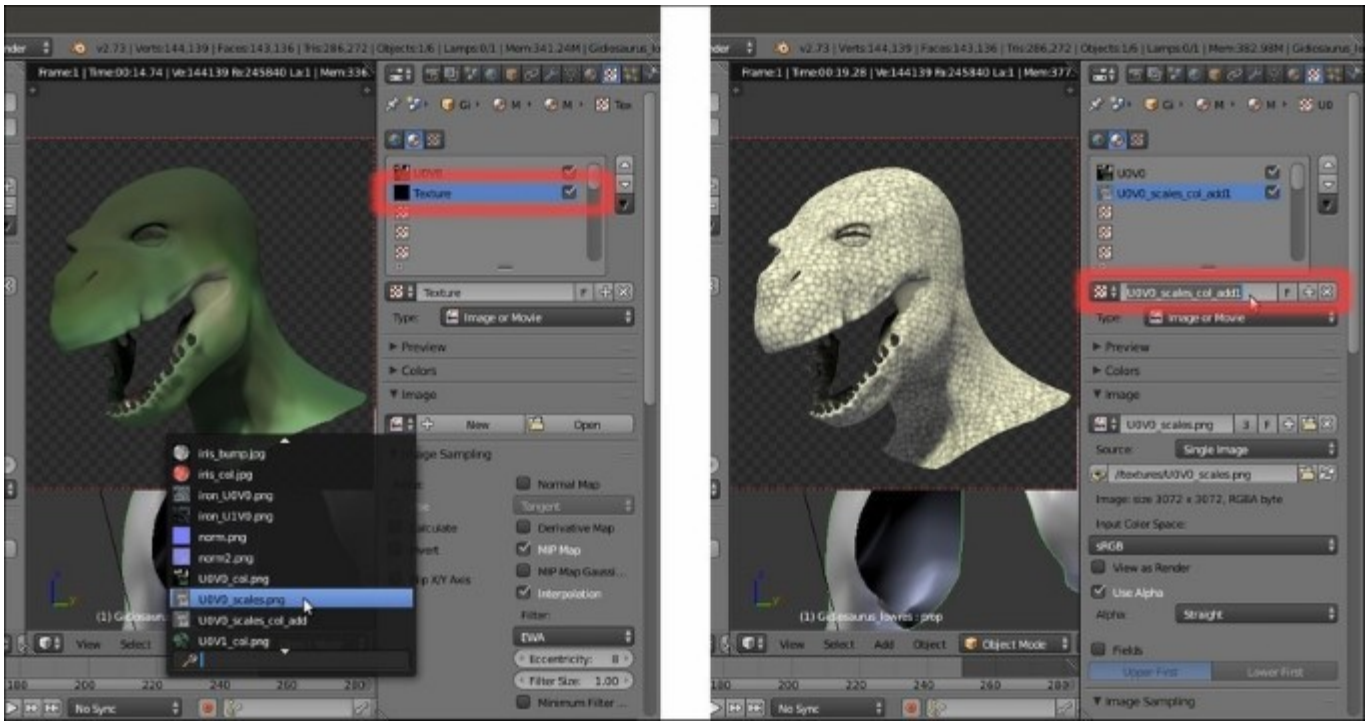
14. Go even further down to find the **Influence** subpanel: be sure that the diffuse **Color** channel is the one enabled and that the slider is set to **1.000** (again, default settings):

The Influence settings subpanel for the texture

15. Scroll back to the top of the **Texture** window and click on the *Unique datablock ID name* slot, where the generic **Texture** name is written; rename it as `U0V0` (as you can see in the following screenshot, this is the name that also appears in the textures list window):

Renaming the texture datablock

16. Now click on the empty **second** slot: again, click on the **New** button, click on the double arrows and this time load the image `U0V0_scales.png`.
17. In the *Unique datablock ID name* slot, rename it as `U0V0_scales_col_add1`.

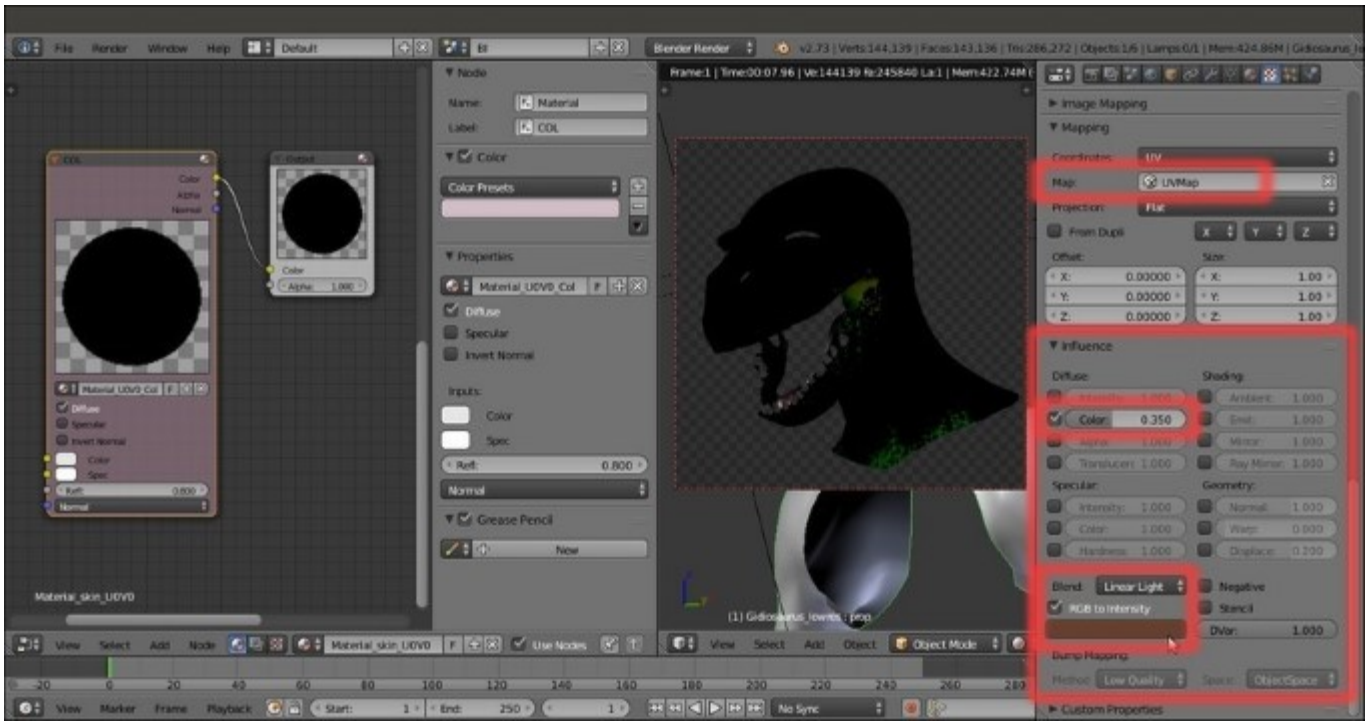


Adding a new texture slot, loading a new image texture and renaming it accordingly

Note

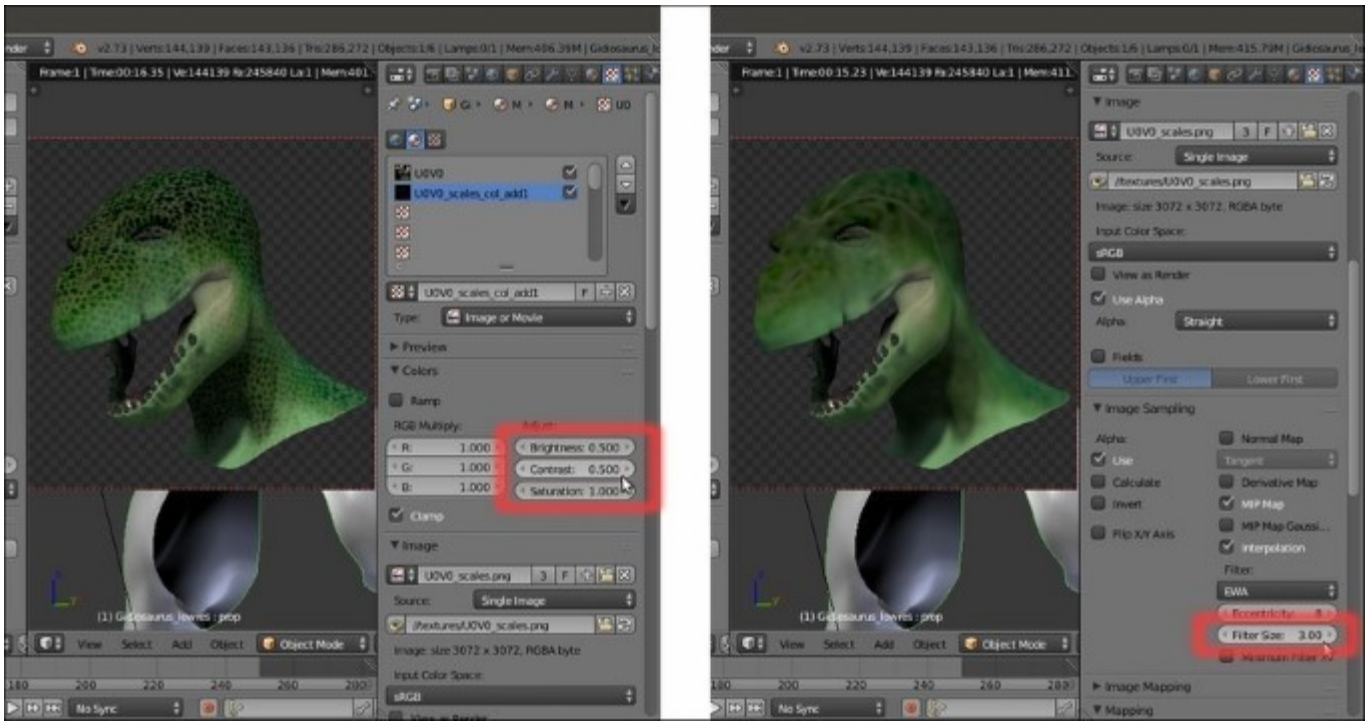
Note that as we load the `U0V0_scales.png` image in the **second** texture slot, the **Rendered** preview changes to show the grayscale image mapped on the model; this is because, by default, the **Influence** of any new added texture is set to the **Color** channel with a value **1.000** and **Blend Type** to **Mix**.

18. In the **Input Color Space** slot under the **Image** subpanel, change the default **sRGB** to **Non-Color**; then, scroll down to the **Mapping** panel to set the **UVMap** coordinates item and then go to the **Influence** subpanel: leave the **Color** channel enabled but move the slider to the lower value of **0.350**, then change the **Blend Type** to **Linear Light** (for the **Blender Internal** materials, the **Blend Type** works as the layer system of a 2D image editor such as **Photoshop** or **Gimp**); enable the **RGB to Intensity** item and change the pink color to **R 0.130**, **G 0.051**, **B 0.030**.



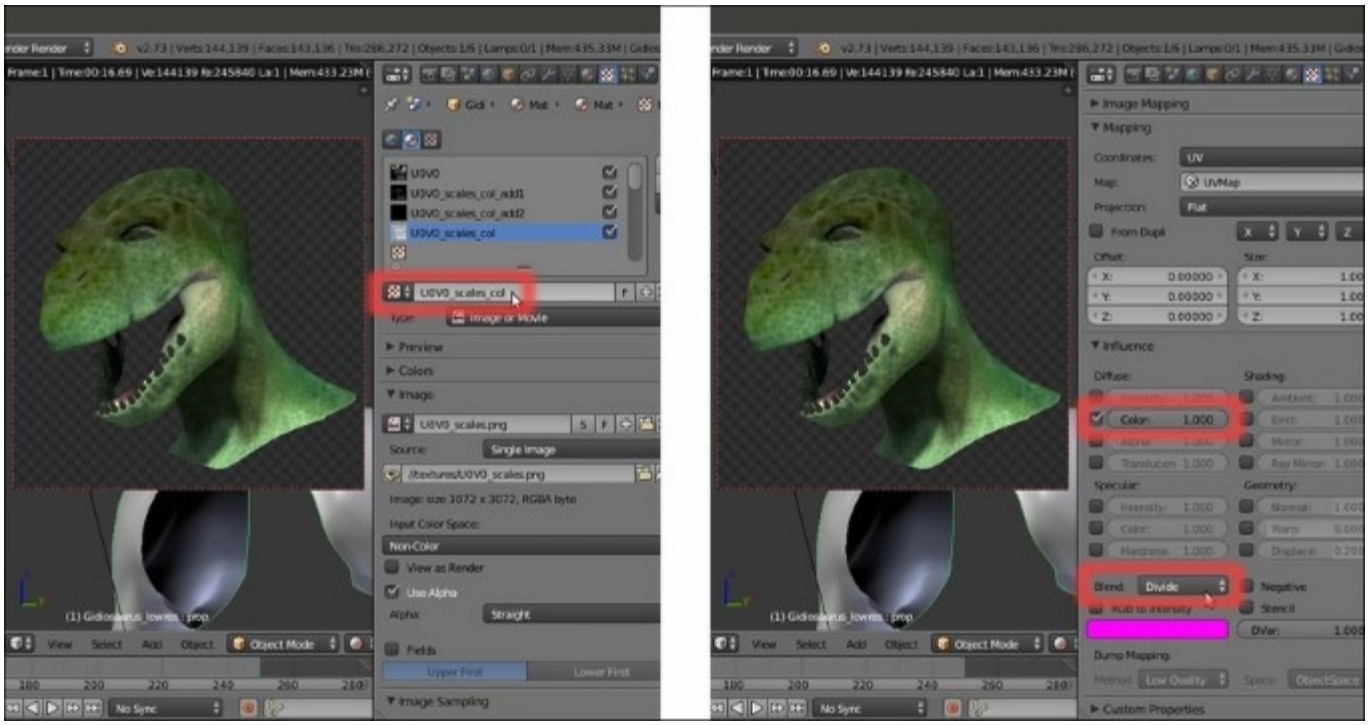
Tweaking the Influence settings for the second texture

19. Go up to expand the **Colors** subpanel: set the **Brightness** and the **Contrast** to **0.500**, to make the texture less bright and less contrasted. Go down to the **Image Sampling** subpanel and set the **Filter Size** to **3.00**, to blur the image (values beyond **1.00** start to blur the image more and more):



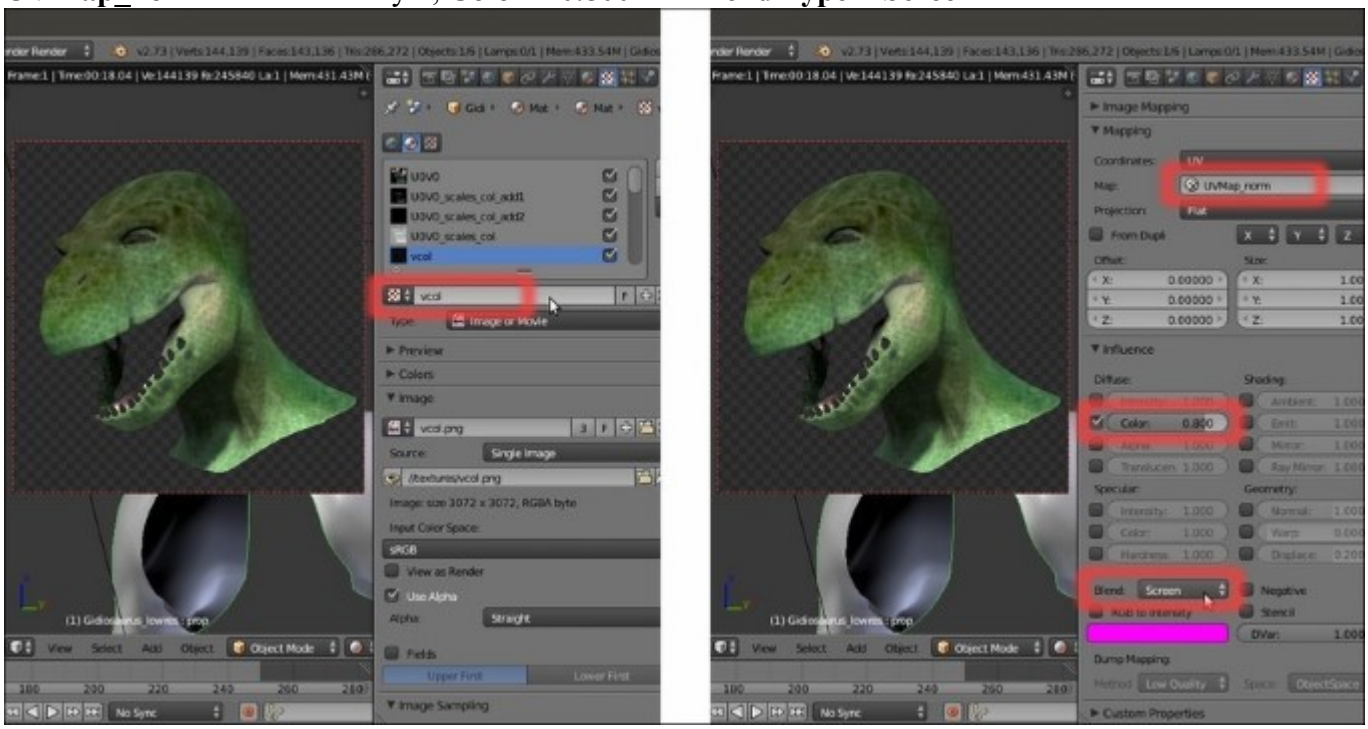
Modifying the appearance of the second image texture

20. Select the **third** empty texture slot and repeat the procedure, again loading the `U0V0_scales.png` image; in the *Unique datablock ID name* slot, rename it as `U0V0_scales_col_add2`.
21. Scroll down to the **Mapping** panel to set the **UVMap** coordinates item and then in the **Influence** subpanel, leave the **Color** channel enabled at value **1.000** but change the **Blend Type** to **Subtract**. Set the **Brightness** to **0.100** and the **Contrast** to **1.500**. Again, set the **Filter Size** to **3.00**.
22. Select the **fourth** texture slot, load again the `U0V0_scales.png` image, rename it `U0V0_scales_col`, set the **UVMap** coordinates layer, **Color** = **1.000** and **Blend Type** = **Divide**:



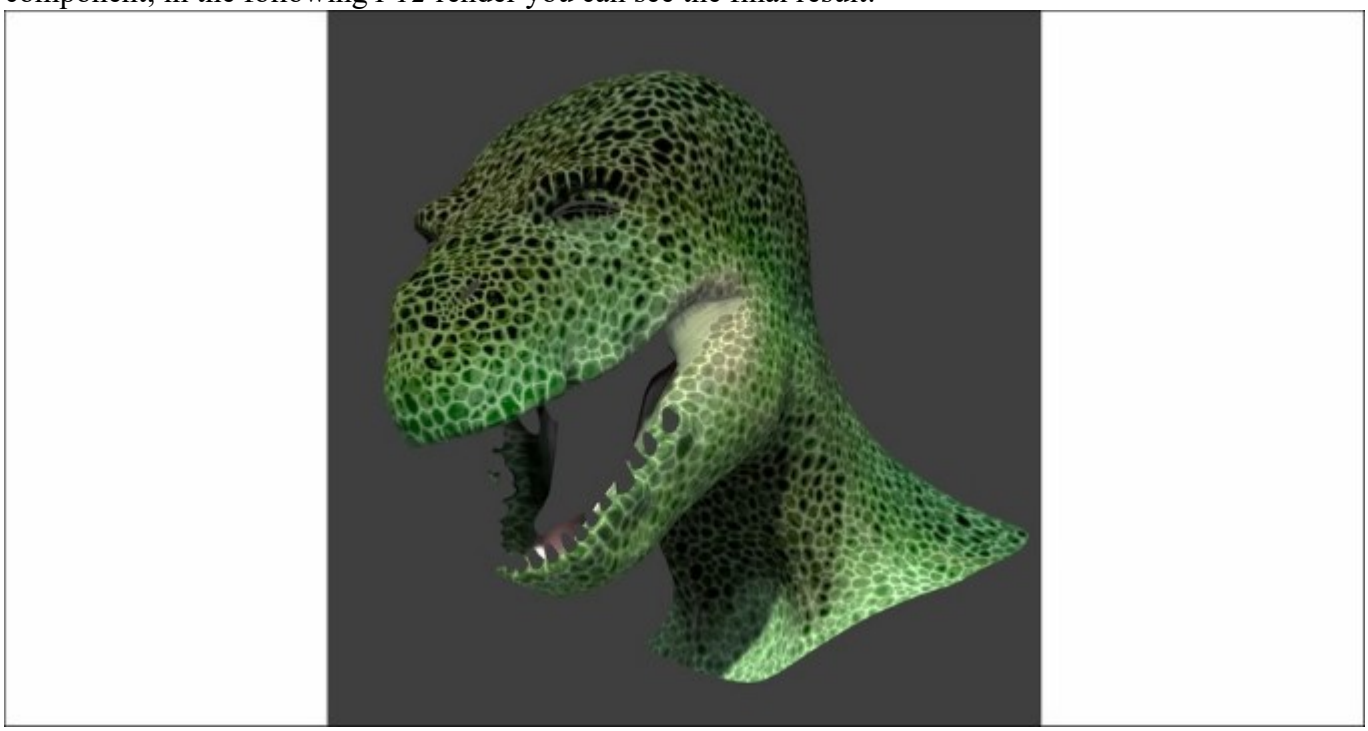
Adding more texture slots with different settings

23. Select the **fifth** texture slot, load the `vcol.png` image again, rename it `vcol`, set the **UVMap_norm** coordinates layer, **Color** = **0.800** and **Blend Type** = **Screen**:



Adding the baked Vertex Color image texture as well

At this point we have completed the **first** component of the skin shader, that is, the **diffuse color** component; in the following *F12* render you can see the final result:



The completed diffuse color component of the Blender Internal skin material

Note that this *F12* render result is quite different from the **Rendered** real-time preview; this is probably due to the complexity of using several textures inside a node material system with the (sadly) bad real-time viewport performances of Blender.

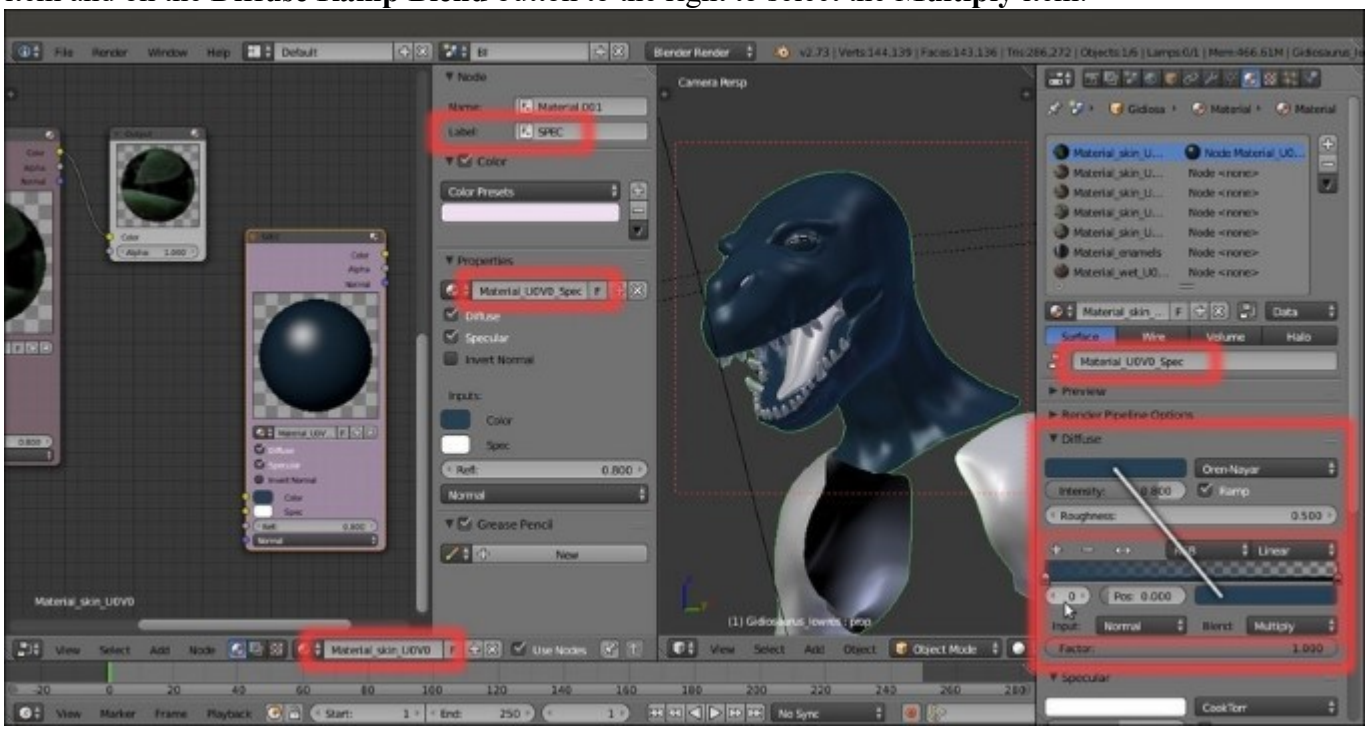
Note

Also note that the only parts of the **Gidiosaurus** mesh that appear in the rendered image are actually the parts we assigned a **Blender Internal** material to; in fact, the **teeth** and the **tongue** are rendered as blank shapes (even working as a mask).

Now, we can carry on with building the second component of the shader, the **glossy** component.

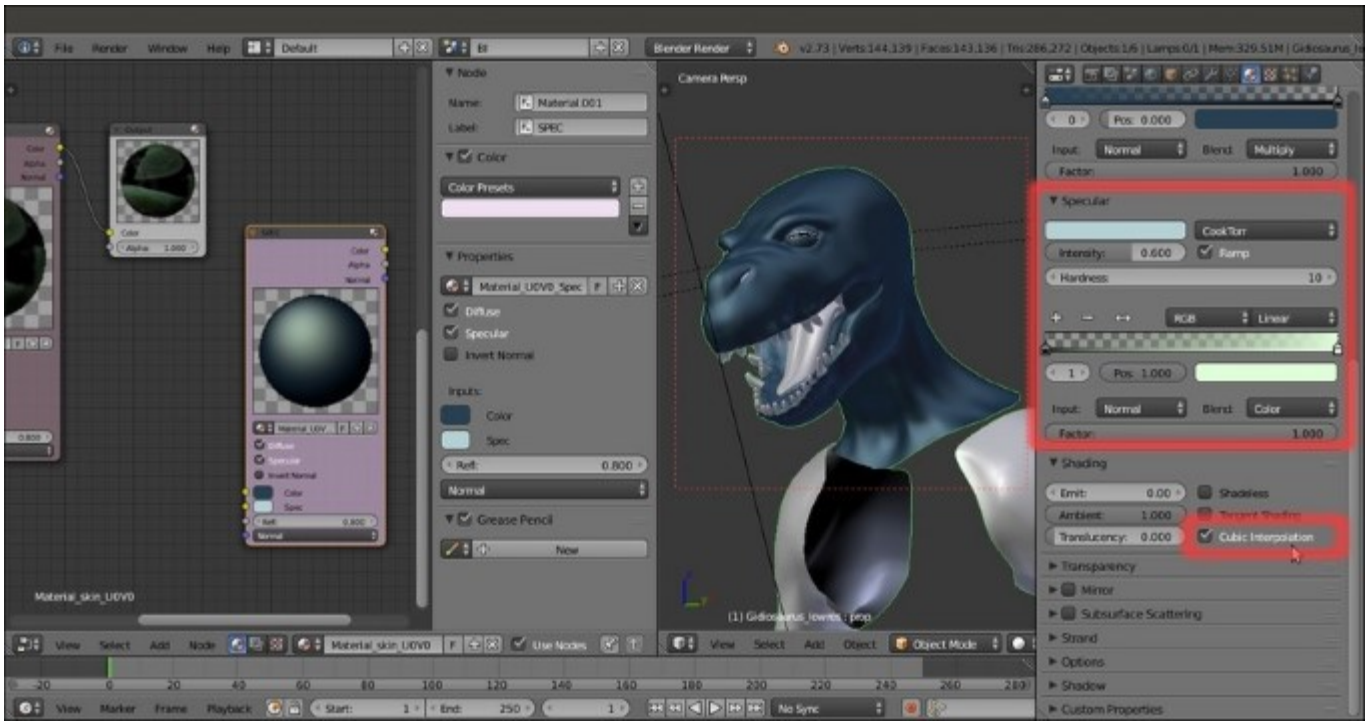
24. Put the mouse pointer inside the **Node Editor** window and add a new **Material** node (*Shift* + *A* | **Input** | **Material**); label it as **SPEC** and then click on the **New** button to create a new material: rename it `Material_U0V0_Spec`.
25. Go to the **Material** window; in the **Diffuse** sidepanel, change the shader model to **Oren-Nayar**, then change the color to a deep blue **R 0.020**, **G 0.051**, **B 0.089**.

26. Enable the **Ramp** item: in the slider, switch the positions of the two color stops (that is: white color stop to position **0.000** and black color stop to position **1.000**), then select the white color stop; put the mouse on the deep blue color slot of the **Diffuse** subpanel and press *Ctrl + C* to copy it; put the mouse pointer on the color slot of the selected color stop and press *Ctrl + V* to paste the deep blue color.
27. Click on the **Diffuse Ramp Input** button at the bottom of the subpanel to select the **Normal** item and on the **Diffuse Ramp Blend** button to the right to select the **Multiply** item:



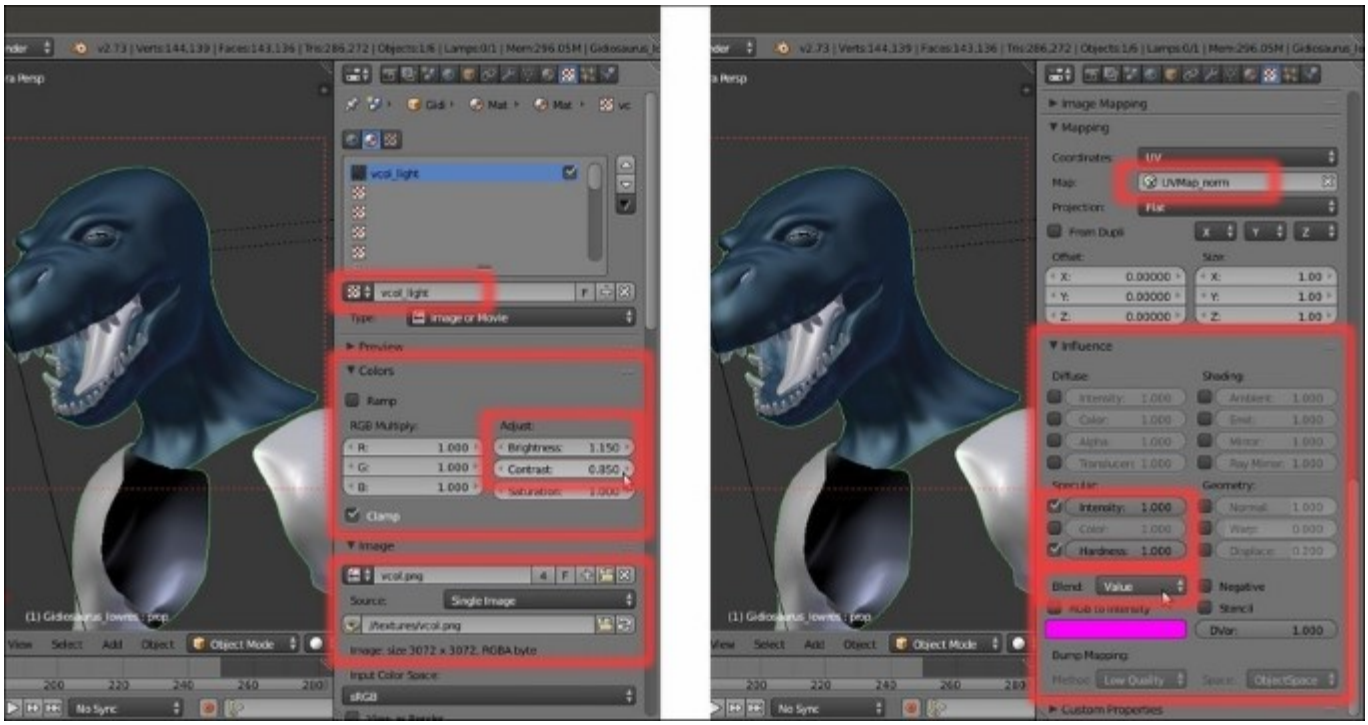
The "Material_U0V0_Spec", to be used inside the "Material_skin_U0V0" node material

28. Scroll down to the **Specular** subpanel: change the color to a light blue **R 0.474**, **G 0.642**, **B 0.683**; set the **Intensity** to **0.600** and the **Hardness** to **10**.
29. Enable the **Ramp** item: select the white color stop and change the color to **R 0.761**, **G 1.000**, **B 0.708**, then set the **Diffuse Ramp Input** button to **Normal** and the **Diffuse Ramp Blend** to **Color**.
30. Go to the **Shading** subpanel and enable the **Cubic Interpolation** item:



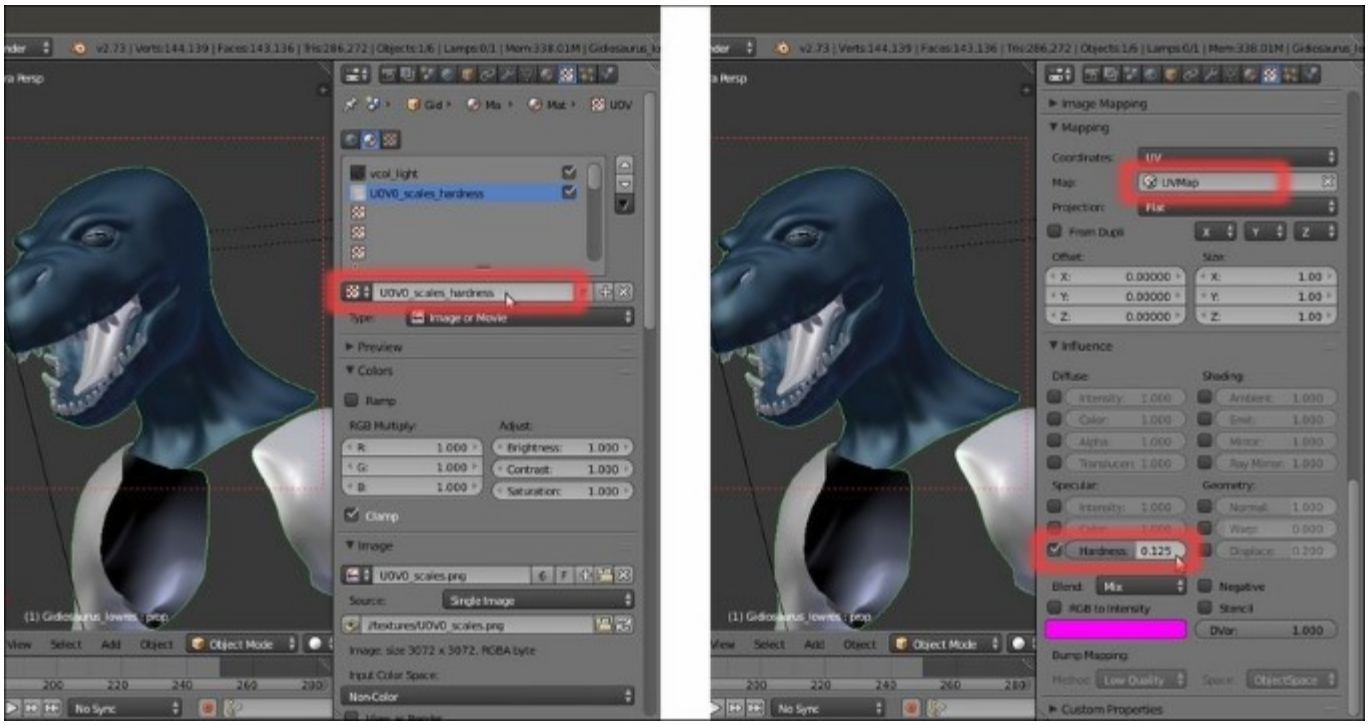
Setting the parameters of the specularity component

31. Go to the **Textures** window; select the top **first** empty texture slot and click on the **New** button. Load the image `vcol.png`, rename the ID datablock as `vcol_light` and go to the **Colors** subpanel: set the **Brightness** to **1.150** and the **Contrast** to **0.850**. Go down to the **Mapping** subpanel and set the **UVMap_norm** coordinates layer, then in the **Influence** subpanel disable the diffuse **Color** channel and enable both the **Intensity** and the **Hardness** channels under **Specular**; set the **Blend Type** to **Value**:



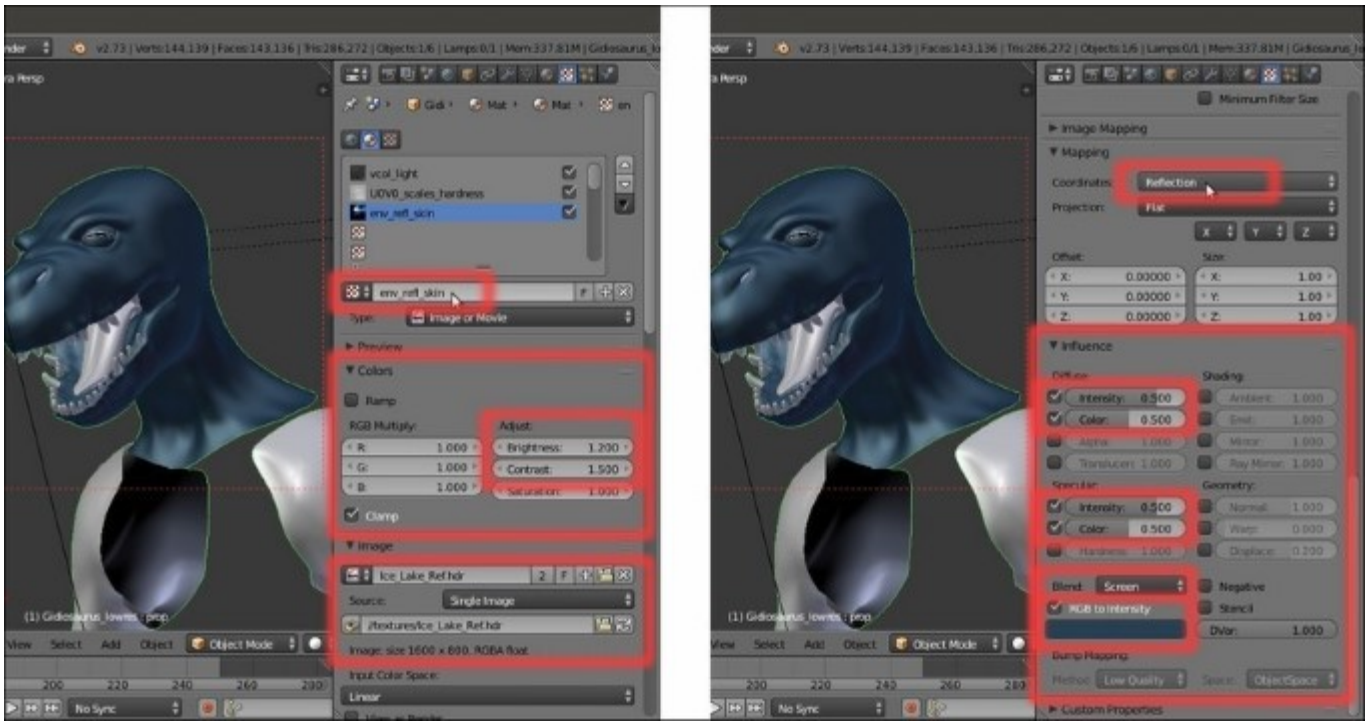
The settings for the specularity first texture

32. Go to the **second** slot and load the image `U0V0_scales.png`; rename it as `U0V0_scales_hardness`, in the **Mapping** subpanel, set the **UVMap** coordinates layer, in the **Influence** subpanel disable the diffuse **Color** and enable the **Hardness** channel under **Specular** to **0.125**. In the **Image Sampling** subpanel set the **Filter Size** to **5.00**.



Re-using the "U0V0_scales.png" image texture for the specularity hardness

33. In the **third** slot load the image `Ice_Lake_Ref.hdr`, a free high dynamic range image licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 License from the **sIBL Archive** (http://www.hdrlabs.com/sibl/archive.html); there is a reason we are now using the **hdr** image, and it's explained in the *How it works…* section.
34. Rename the image ID datablock as `env_refl_skin` and in the **Colors** subpanel, set the **Brightness** to **1.200** and the **Contrast** to **1.500**; go to the **Mapping** subpanel and set the **Texture Coordinates** to **Reflection**. Down in the **Influence** subpanel, enable both the **Intensity** channel under **Diffuse** and **Specular** and set their sliders to **0.500**; enable also, the **Color** channel under **Specular** and set the sliders of both the **Color** channels to **0.500** as well. Set the **Blend Type** to **Screen**, enable the **RGB to Intensity** item and set the color to the same deep blue of the diffuse color (**R 0.020**, **G 0.051**, **B 0.089**):



Using the environment hdr image as reflection map

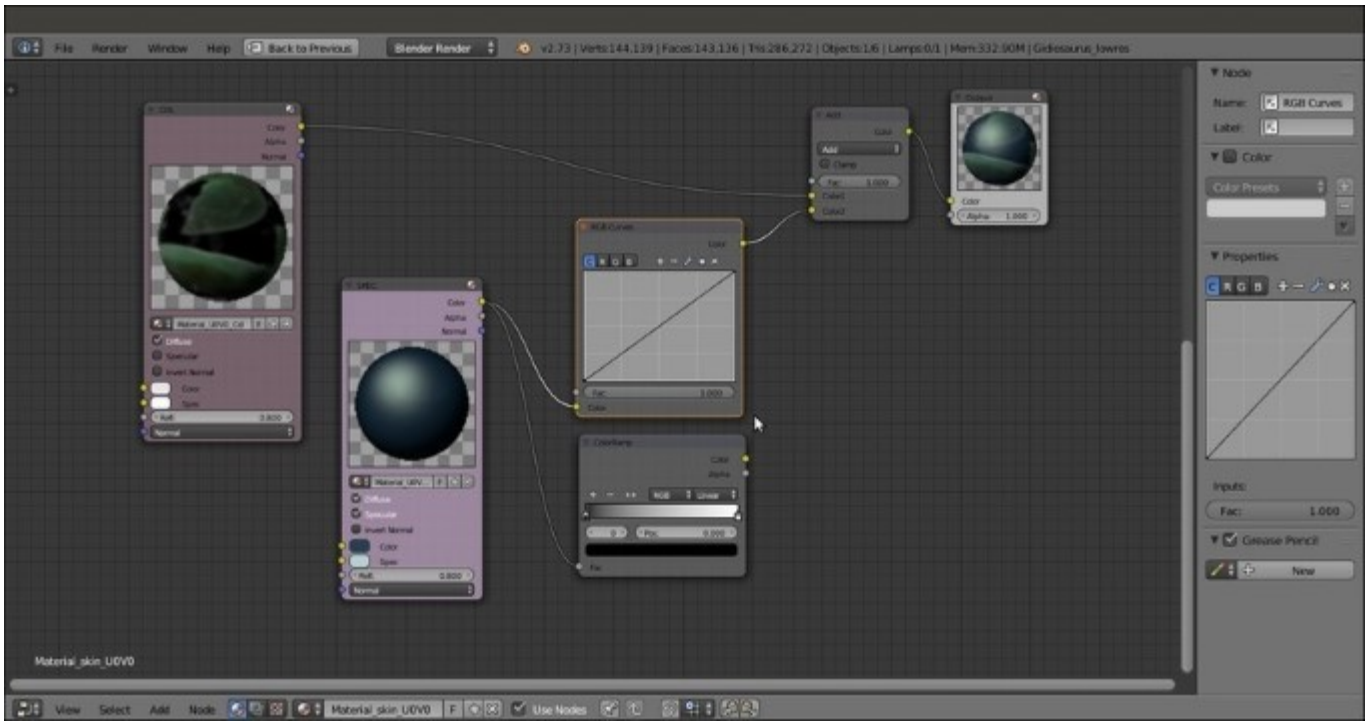
If you want to see the effect of the single components in the **Rendered** preview as we build the shader, just temporarily disconnect the **COL** node link to the **Output** node and replace it with the **Color** output of the **SPEC** node (in this case):

Testing the specularity component material in the rendered preview

35. At this point, add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and move it on the link connecting the **COL** node to the **Output** node, to automatically paste it between them; then connect the **Color** output of the **SPEC** node to the **Color2** input socket of the **MixRGB** node, set the **Blend Type** of this latter node to **Add** and its **Fac** value to **1.000**:

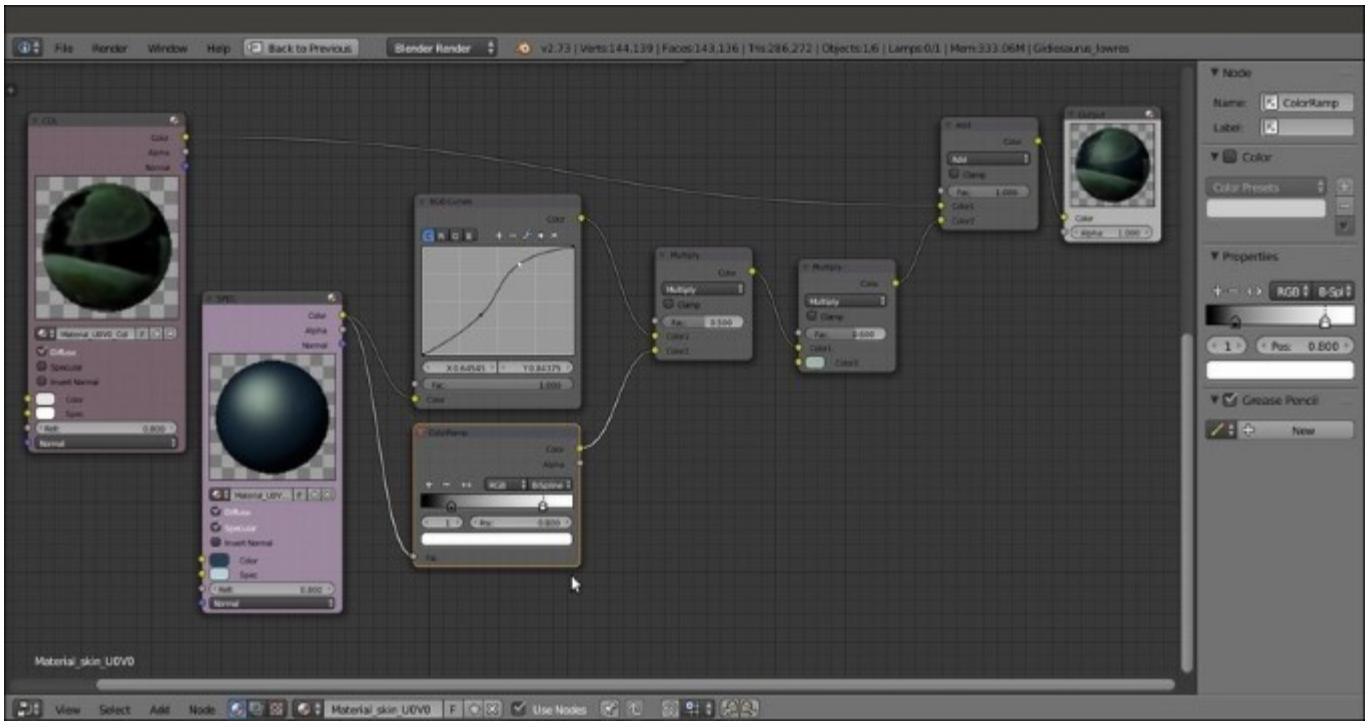
Finally adding the specularity component to the diffuse component

36. Add a **RGB Curves** node (*Shift* + *A* | **Color** | **RGB Curves**) and a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**); paste the **RGB Curves** node between the **SPEC** and the **MixRGB** node, then connect the **Color** output of the **SPEC** node also to the **ColorRamp** input socket:



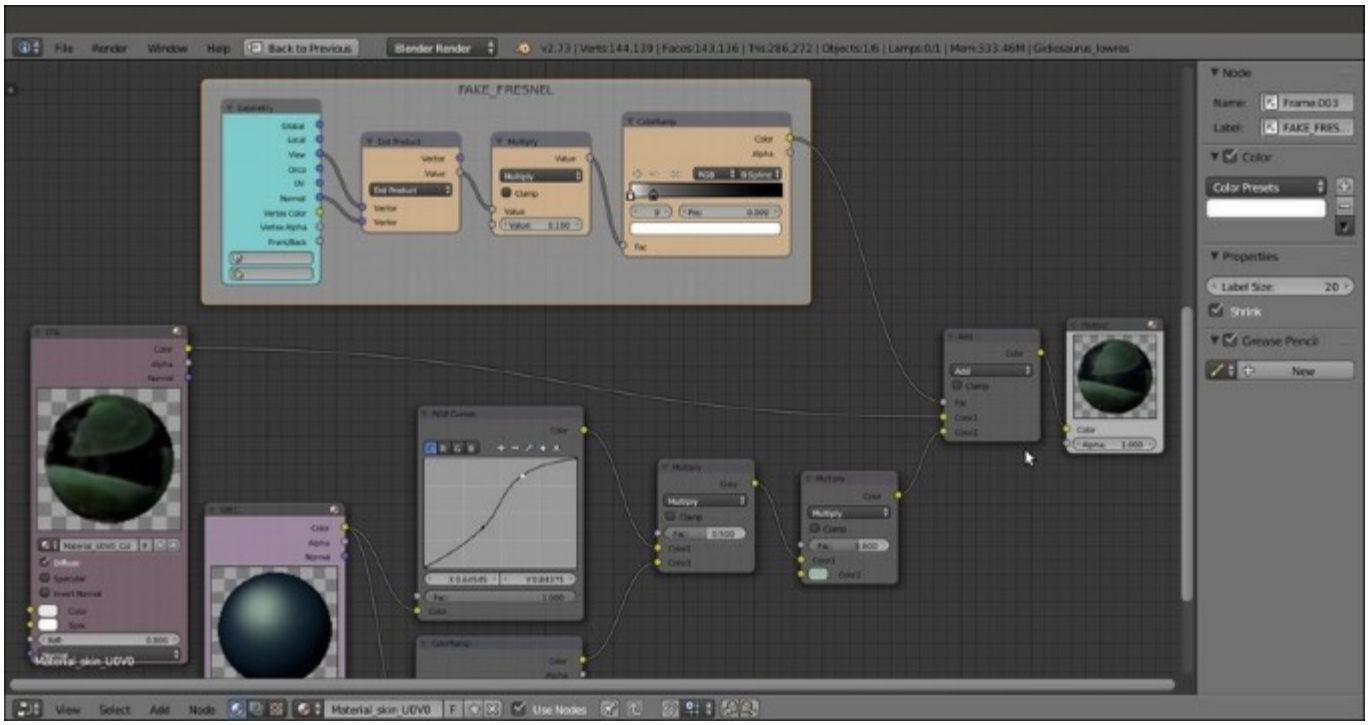
Adding new nodes

37. Press *Shift* + *D* to duplicate the **MixRGB** node, change the **Blend Type** of the duplicate to **Multiply** and paste it between the **RGB Curves** and the first **Add-MixRGB** nodes: set the **Fac** value to **0.500**. Connect the **Color** output of the **ColorRamp** node to the **Color2** input socket of the **Multiply-MixRGB** node.
38. Press *Shift* + *D* to duplicate the **Multiply-MixRGB** node and paste the duplicate between the first **Multiply-MixRGB** node and the **Add-MixRGB** node. Set the **Fac** value of the last **Multiply-MixRGB** node to **0.600** and the **Color2** to **R 0.347**, **G 0.462**, **B 0.386**.
39. Go to the **RGB Curves** node and left-click inside the interface window to add a point; set its coordinates to **X** = **0.38636** and **Y** = **0.36875**. Add a second point and set its coordinates to **X** = **0.64545** and **Y** = **0.84375**.
40. Go to the **ColorRamp** node and set the **Interpolation** to **B-Spline**, then move the black color stop to position **0.195** and the white color stop to position **0.800**:



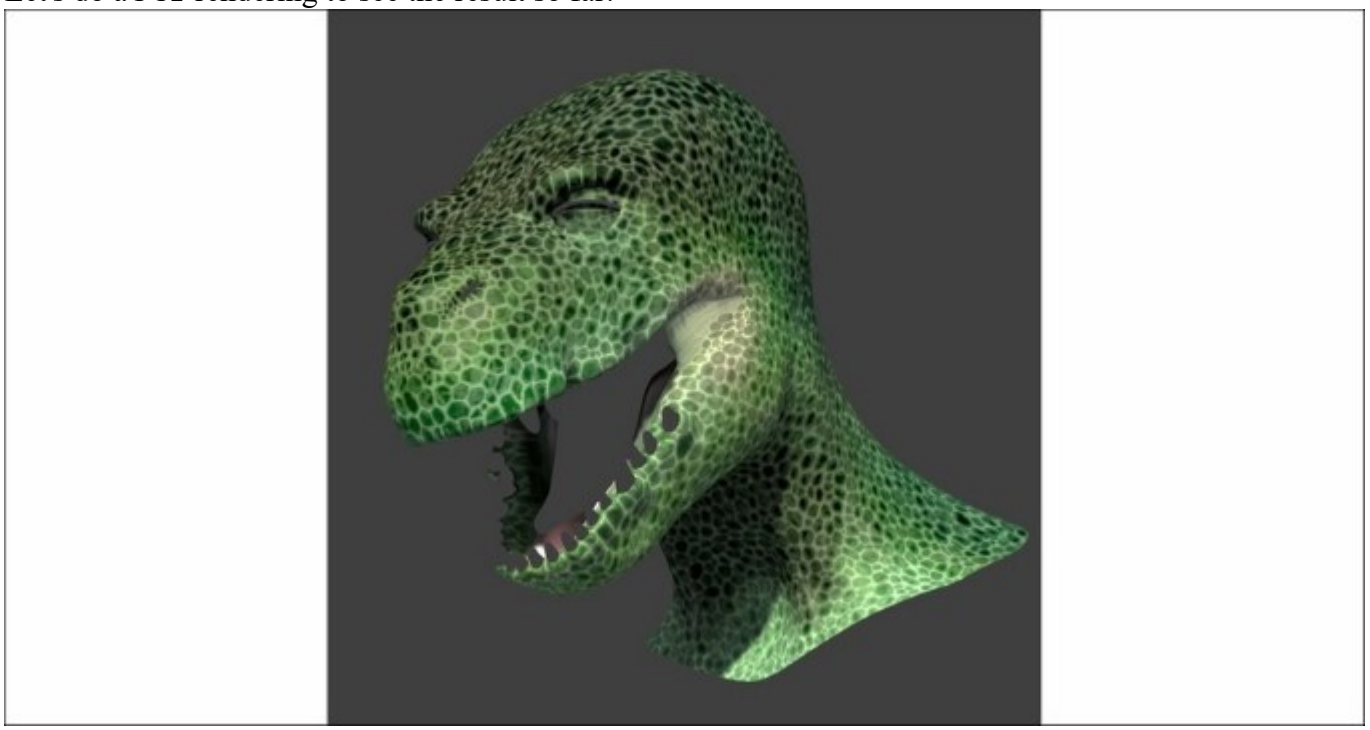
Tweaking the specularity component through the new nodes

41. Add a **Geometry** node (*Shift* + *A* | **Input** | **Geometry**), a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**), a **Math** node (*Shift* + *A* | **Converter** | **Math**), and a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**).
42. Add a **Frame** (*Shift* + *A* | **Layout** | **Frame**), label it as **FAKE_FRESNEL** and parent the last **four** added nodes to it.
43. Set the **Operation** of the **Vector Math** node to **Dot Product**, then connect the **View** output of the **Geometry** node to the first **Vector** input socket of the **Vector Math** node, and the **Normal** output of the **Geometry** node to the second **Vector** input socket of the **Vector Math** node.
44. Connect the **Value** output of the **Dot Product** node to the first **Value** input socket of the **Math** node; set the **Operation** of this latter node to **Multiply** and the second **Value** to **0.100**.
45. Connect the output of the **Math** node to the **Fac** input socket of the **ColorRamp** node; set the **Interpolation** of this latter node to **B-Spline**, then move the black color stop to position **0.150** and the white color stop to position **0.000**.
46. Connect the **Color** output of the **ColorRamp** node to the **Fac** input socket of the **Add-MixRGB** node:



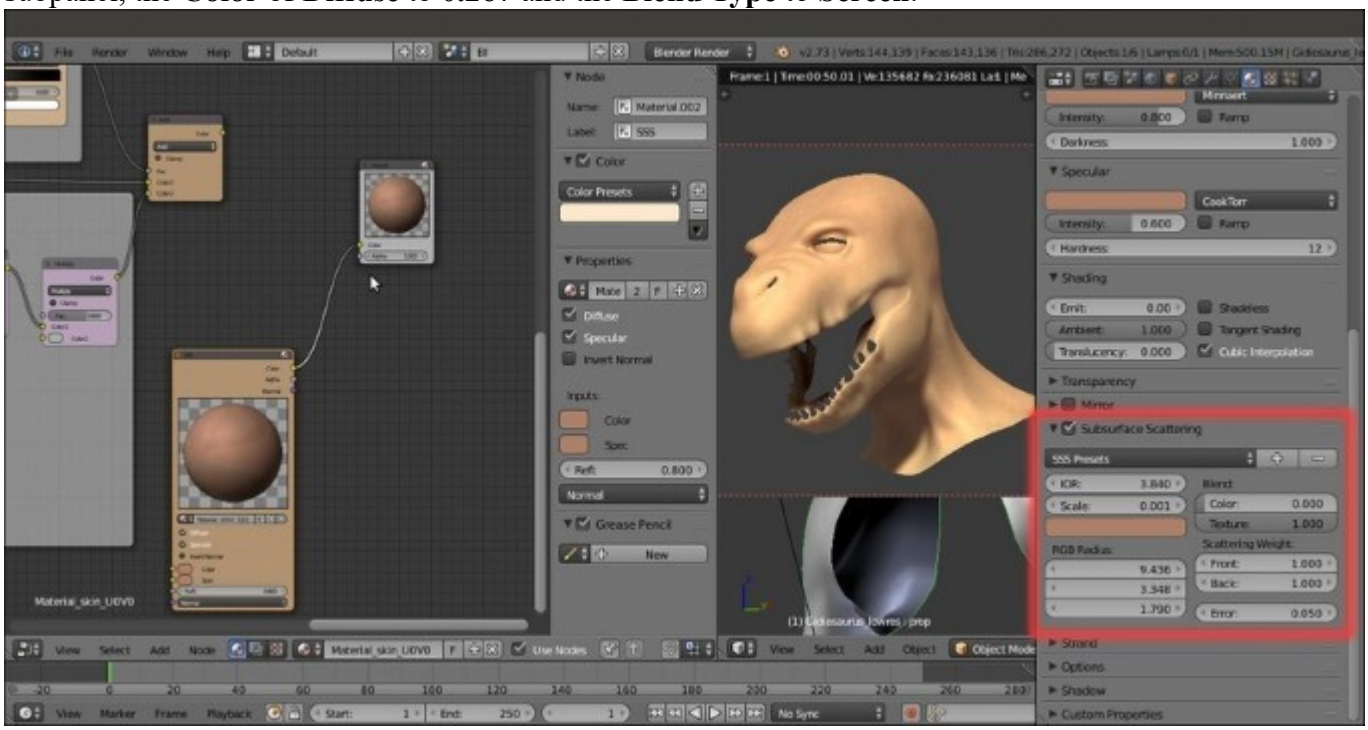
Adding the output of a fake Fresnel as factor for the blending of the two components

Let's do a *F12* rendering to see the result so far:



The F12 rendered result so far

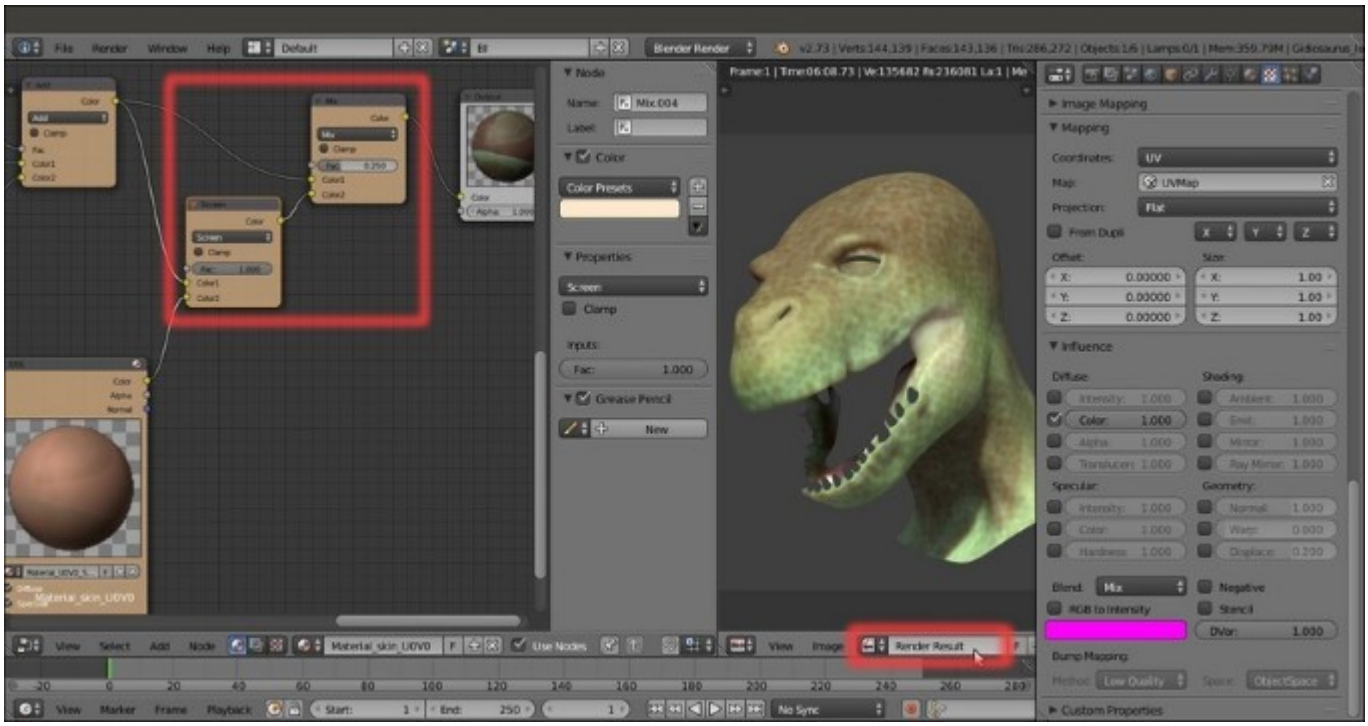
47. Now, select the **COL** material node and press *Shift* + *D* to duplicate it. Label the duplicated one as **SSS**, then through the **Node Editor** window, enable the **Specular** item. Click on the **2** icon button to the right side of the material name datablock to make it single user and rename the new copy of the material as `Material_U0V0_SSS`.
48. Go to the **Material** window and change the **Diffuse Shader Model** to **Minnaert** and the **Diffuse** color to **R 0.439**, **G 0.216**, **B 0.141**. Move down to the **Specular** subpanel and change the color to the same **R 0.439**, **G 0.216**, **B 0.141** brownish hue (copy and paste), then set the **Intensity** to **0.600** and the **Hardness** to **12**.
49. Go down to the **Subsurface Scattering** subpanel and enable it by checking the checkbox: set the **IOR** value to **3.840**, the **Scale** to **0.001**, copy and paste the brownish color also in the scattering color slot, set the **Color** slider to **0.000** and the **Texture** slider to **1.000**. Set the **RGB Radius**: **R 9.436**, **G 3.348**, **B 1.790**.
50. Go to the **Texture** window and set to **0.300** the **Color** channel sliders of the `U0V0`, `U0V0_scales_col_add2`, and `U0V0_scales_col` texture slots, then set to **0.117** the **Color** channel slider of the `U0V0_scales_col_add1` texture slot.
51. Select the last `vcol` texture slot and click on the **X** icon (*Unlink datablock*) button to clear it; click on the double little arrows to the left side of the **New** button and select the `vcol_light` item from the pop-up menu. Set the **Mapping** to **UVMap_norm**, and under the **Influence** subpanel, the **Color** of **Diffuse** to **0.267** and the **Blend Type** to **Screen**.



Testing the output of the SSS component material

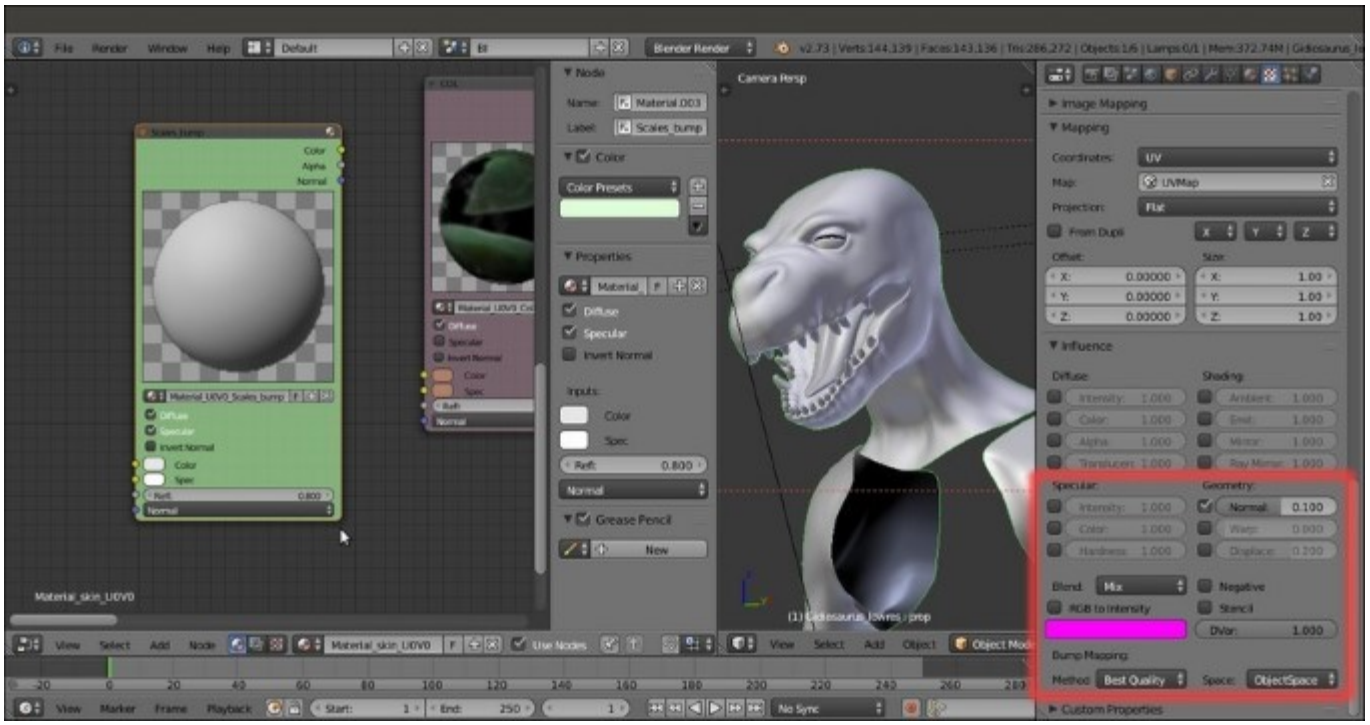
52. Add a new **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**), paste it between the **Add-MixRGB** and the **Output** node and set the **Fac** value to **0.250**.

53. Press *Shift* + *D* to duplicate this **Mix-MixRGB** node; change the **Blend Type** of the duplicated one to **Screen**, set the **Fac** value to **1.000** and connect the output of the **Add-MixRGB** also to the **Color1** input socket of the **Screen-MixRGB** node; connect the output of the **SSS** node to the **Color2** input socket of the **Screen-MixRGB** node.
54. Connect the output of the **Screen-MixRGB** node to the **Color2** input socket of the first **Mix-MixRGB** node.



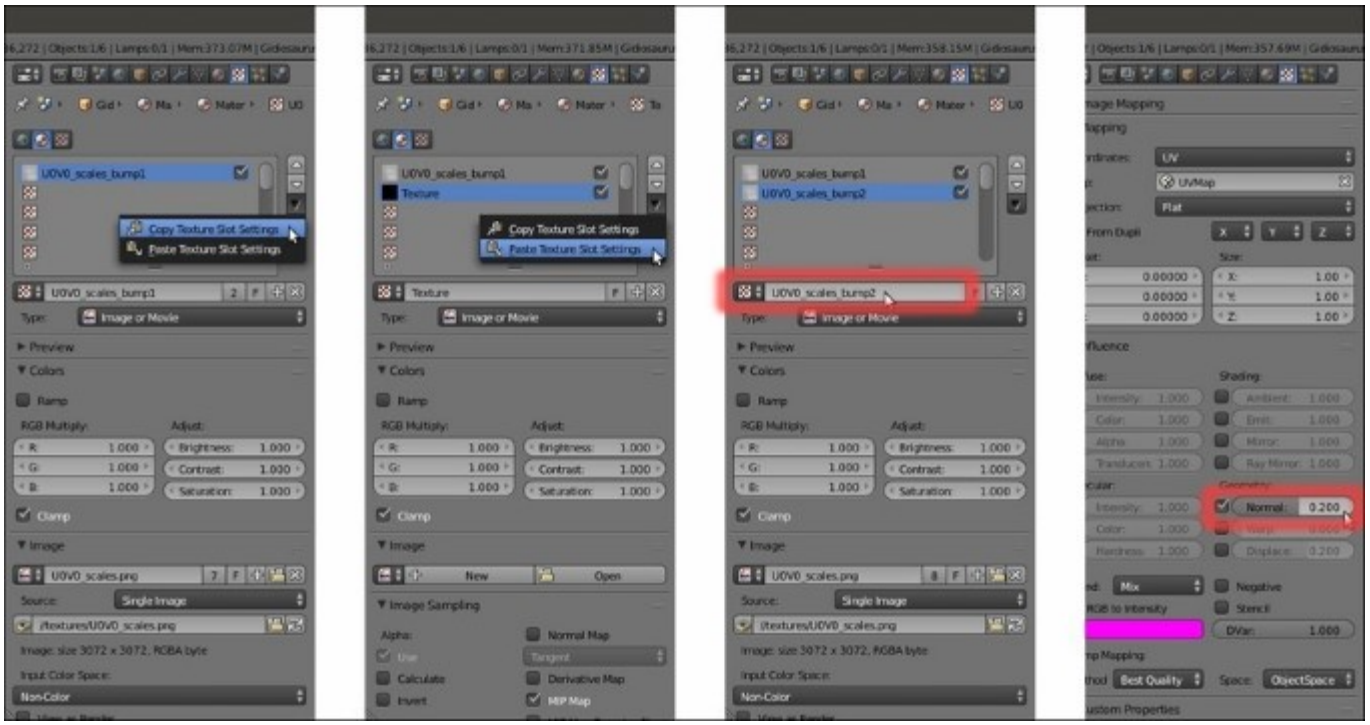
Adding the SSS component to the rest of the shader

55. Add a new **Material** node (*Shift* + *A* | **Input** | **Material**) and click on the **New** button to create a new material; label the node as **Scales_bump** and rename the material as `Material_U0V0_Scales_bump`.
56. In the **Material** window set the **Diffuse Shader Model** to **Oren-Nayar** and the **Specular Shader Model** to **Blinn**, **Intensity** = **0.100** and **Hardness** = 5. In the **Shading** subpanel enable the **Cubic Interpolation** item.
57. Go to the **Texture** window and in the **first** slot load the `U0V0_scales.png` image; rename the ID datablock as `U0V0_scales_bump1`. In the **Image Sampling** subpanel set the **Filter Size** to **5.00**, the **Mapping** to **UVMap** and in the **Influence** subpanel disable the **Color** channel and enable the **Normal** channel under **Geometry**: set the slider to **0.100** and go to the bottom to click on the **Bump Method** slot and select **Best Quality**:



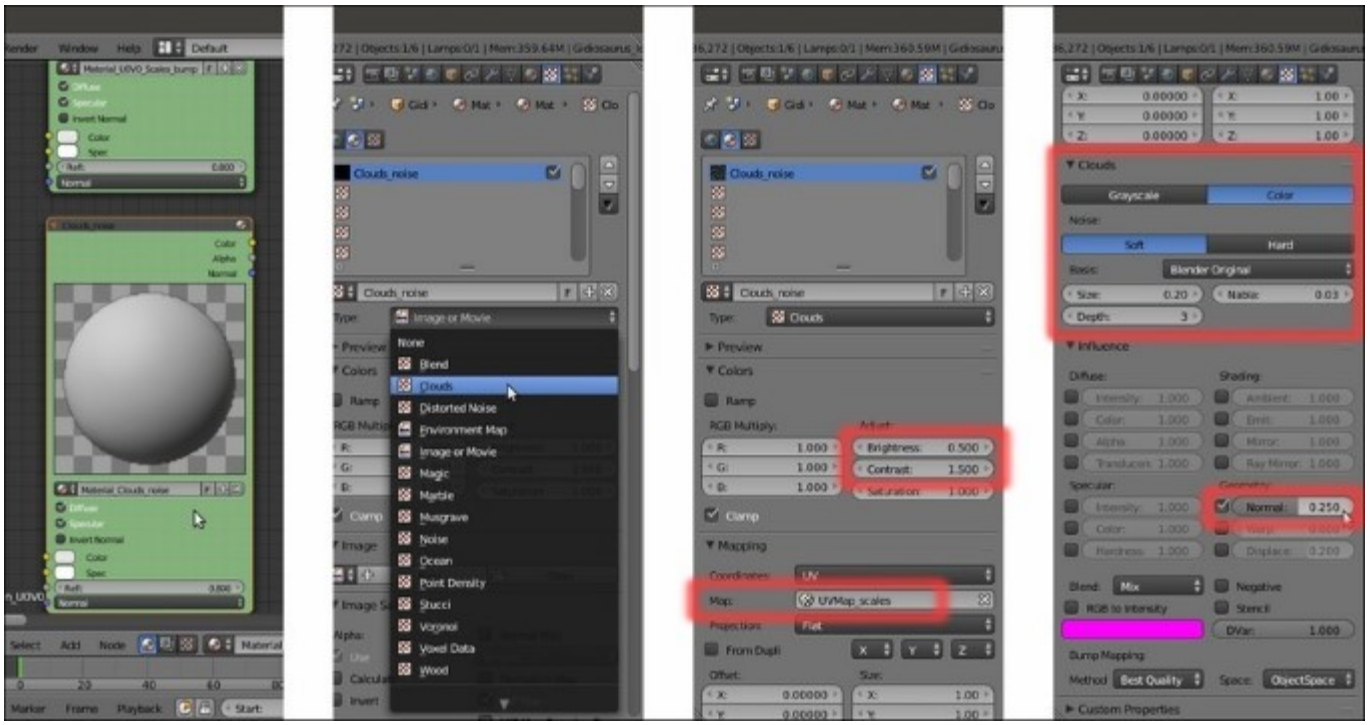
The bump material node

58. Go back to the top of the panel and click on the big black arrow to the right of the texture window; select the **Copy Texture Slot Settings** item.
59. Select the empty **second** texture slot and click on **New** to add a generic texture, then click again on the black arrow to select the **Paste Texture Slot Settings** item.
60. In the *ID datablock* slot, click on the **2** icon button to make it single user and rename it `U0V0_scales_bump2`.
61. Go down to the **Image Sampling** subpanel and set the **Filter Size** to **1.00**, then go down to the **Influence** subpanel and set the **Normal** slider to **0.200**.



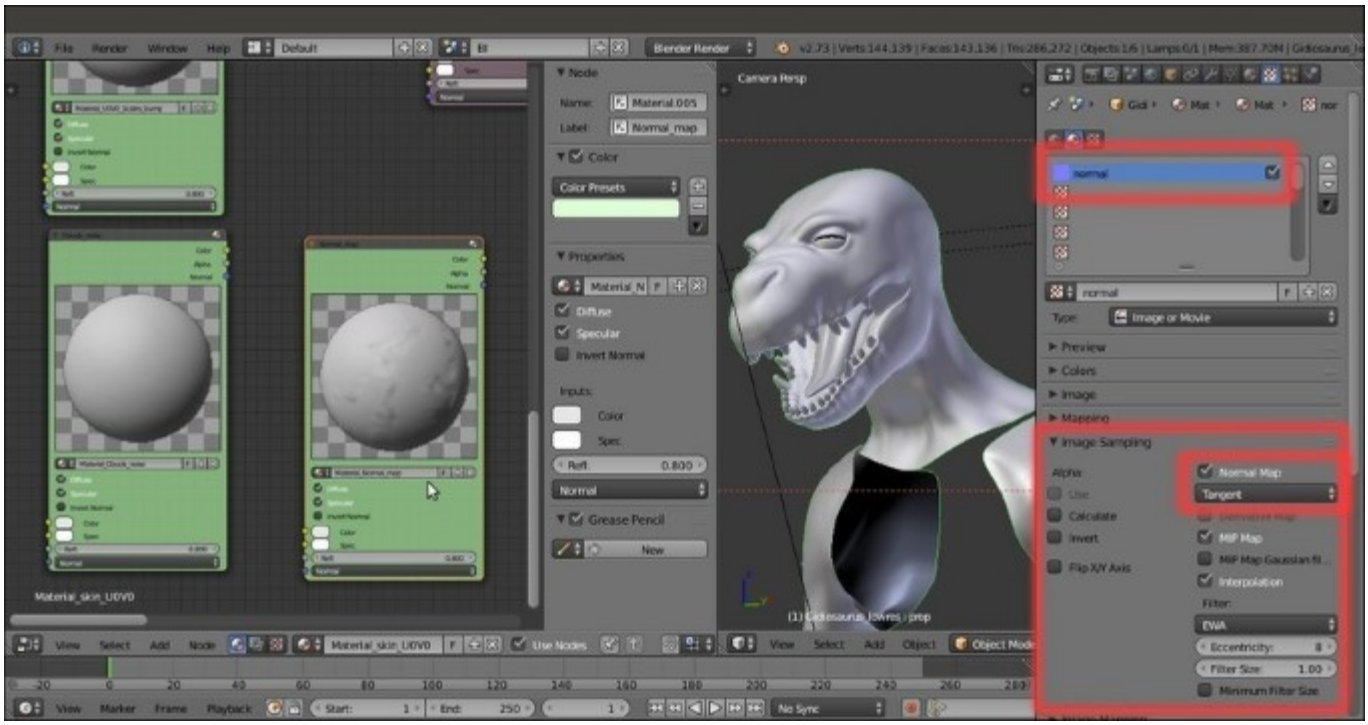
Copying and pasting a texture slot

62. Press *Shift* + *D* to duplicate the node; make the material of the duplicated one single user and rename it as `Material_Clouds_noise`, then label the node as **Clouds_noise**.
63. In the **Texture** window, delete (unlink) `U0V0_scales_bump1` and `U0V0_scales_bump2`, and then select the **first** slot and click on the **New** button: change the automatic `Texture.001` ID datablock name with `Clouds_noise`, click on the **Type** button and in the pop-up menu select the **Clouds** item.
64. In the **Colors** subpanel set the **Brightness** to **0.500** and the **Contrast** to **1.500**, in the **Mapping** subpanel set the **UVMap_scales** coordinates layer, in the **Clouds** subpanel switch from **Grayscale** to **Color**, set the **Size** to **0.20**, the **Depth** to **0.3** and the **Nabla** to **0.05**.
65. In the **Influence** subpanel disable the **Color** channel and enable the **Normal** channel under **Geometry**: set the slider to **0.250** and go to the bottom to click on the **Bump Method** slot and select **Best Quality**:



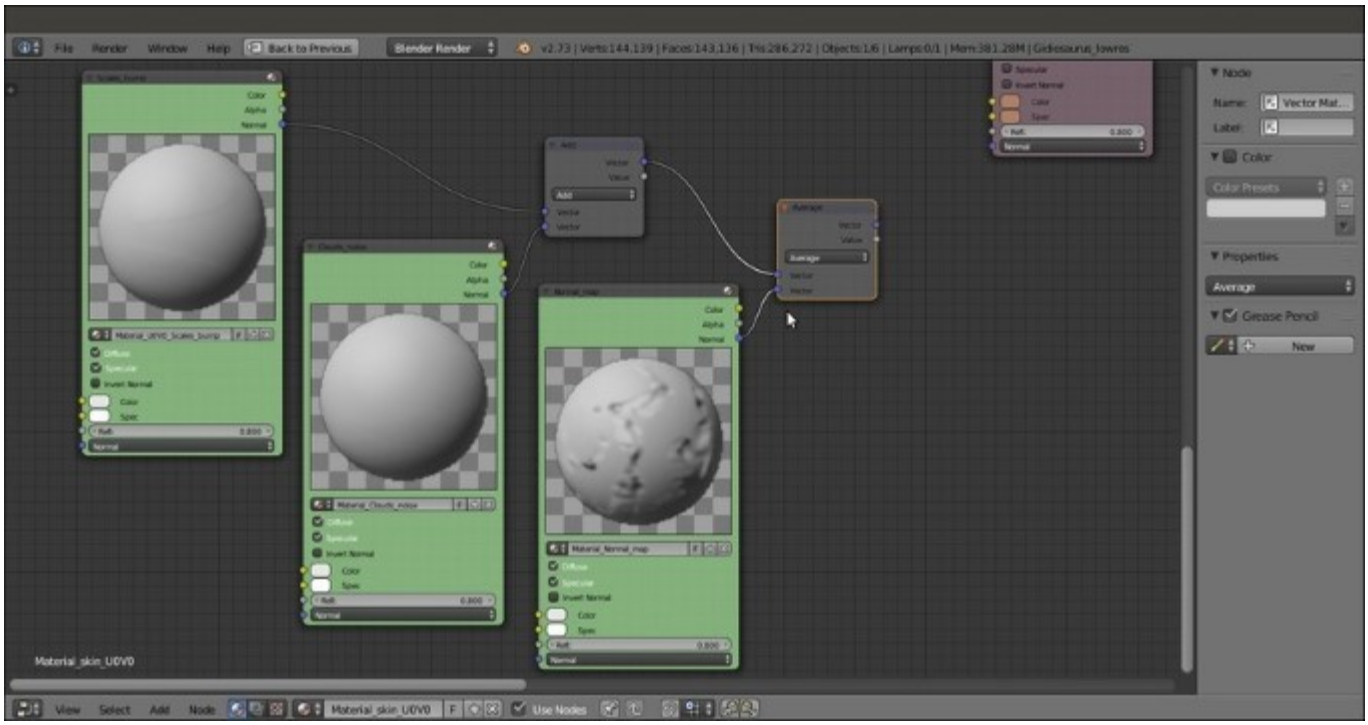
The second bump material node

66. Press *Shift* + *D* to duplicate the **Scales_bump** node; make the material of the duplicated one single user and rename it as `Material_Normal_map`, then label the node as **Normal_map** as well.
67. In the **Texture** window, delete (unlink) the `U0V0_scales_bump1` and `U0V0_scales_bump2`; select the **first** slot and click on the **New** button: change the automatic `Texture.001` ID datablock name to `normal` and then load the `norm.png` image.
68. In the **Mapping** subpanel set the **UVMap_norm** coordinates layer, then go to the **Image Sampling** subpanel and enable the **Normal Map** item; go to the **Influence** subpanel, disable the **Color** channel and enable the **Normal** channel: set the slider to **1.000** (higher values don't have an effect with normal maps in **Blender Internal**).



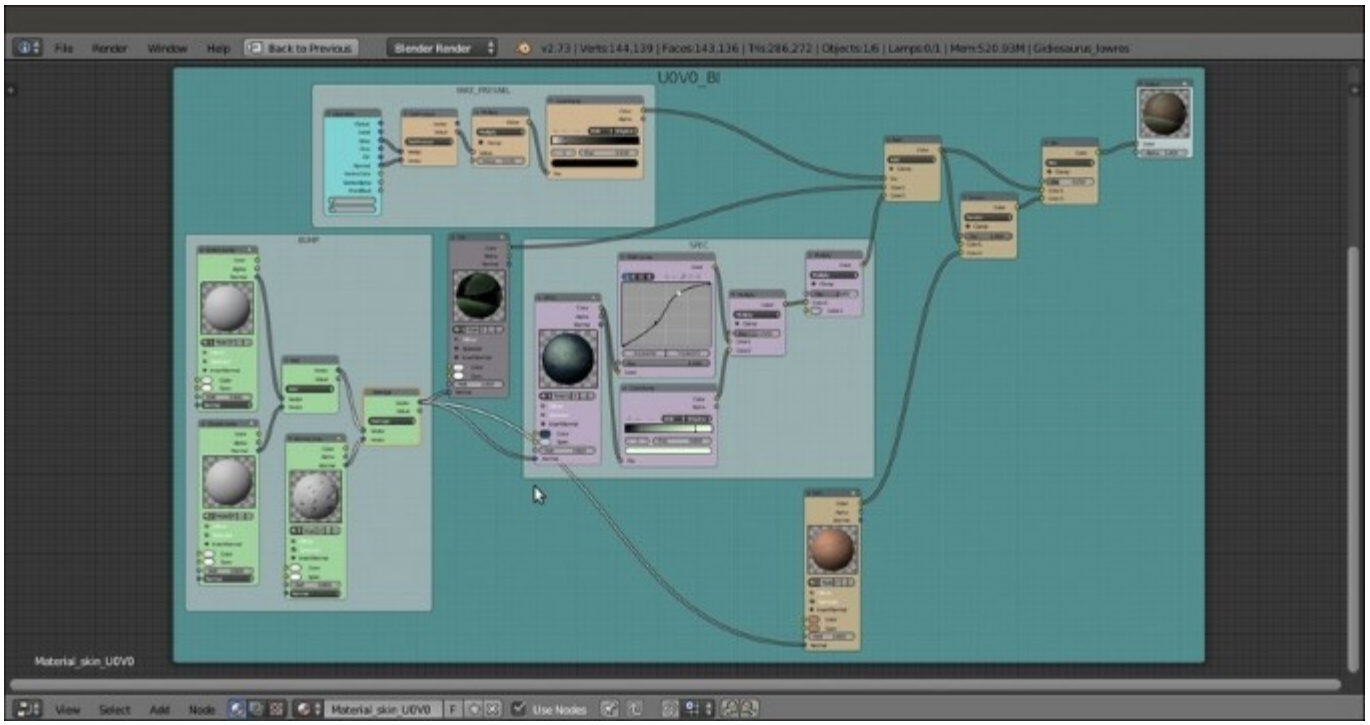
The normal map material node

69. Add a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**) and connect the **Normal** (blue) output of the **Scales_bump** node to the first **Vector** input socket of the **Vector Math** node, and the **Normal** output of the **Clouds_noise** node to the second **Vector** input socket.
70. Press *Shift* + *D* to duplicate the **Vector Math** node, set the **Operation** of the duplicate to **Average** and connect the **Vector** output of the **Add-Vector Math** node to the first **Vector** input socket of the **Average-Vector Math** node, and the **Normal** output of the **Normal_map** node to the second **Vector** input socket.



Connecting the outputs of the three bump nodes

71. Connect the **Vector** output of the **Average-Vector Math** node to the **Normal** input sockets of the **COL**, **SPEC**, and **SSS** material nodes.
72. Add frames everywhere to make things clear but especially to visually group and separate the nodes of the **Blender Internal** material from the **Cycles** ones.



The completed "U0V0_BI" node material

73. Save the file.

How it works…

You have probably noticed that a few of the nodes we can find in the **Cycles** material system are also available for the material nodes in **Blender Internal**; sadly, some are still missing (and probably forever will be), as, for example, a **Fresnel** node that, in fact, we had to approximate with a combination of other different nodes.

Anyway, although not all the same nodes are at our disposal, we had enough of them to try to obtain a result as close as possible as the result we obtained in the **Cycles** material (in the previous Chapter 12, *Creating the Materials in Cycles*).

Note

One thing you should absolutely keep in mind when loading the textures into the material nodes in **Blender Internal** is their order in the texture stack. This is important and must be taken into consideration according to the result we need, because a texture can totally overwrite the texture in the above slot (with the default **Mix** blend type) but can also be added, subtracted, multiplied, divided, and so on; the textures stack works the same as the layer stack system of a 2D graphic editor (**Gimp**, for instance), with the order from the top to the bottom and the different blending options (the **Blend Type** items).

Having said that, let's see the steps:

- From step 1 to step 9 we created a basic **Blender Internal** material node by using both the **Node Editor** and the **Material** window.
- From step 10 to step 23 we assigned the proper textures to the basic material node that becomes, in this case, the `Material_U0V0_Col`, the basic **diffuse color** component of the shader.

  These steps have been described in the most detailed way possible because they are the same steps for all the textures added to the materials; of course, the values and the settings can be different, but basically:
    - We add a texture (image or procedural)
    - We set a mapping orientation
    - We set the influence value on the selected channel (also more than one at a time)
    - Because the same texture can be used (with different settings) more than once, we always rename the *Unique datablock ID* name to make them easily recognizable in the pop-up menu list.
- The **Filter Size** value in the **Image Sampling** subpanel is really useful for blurring an image texture: a value of **1.00** is the default sharpness, while a higher value makes the image more and more blurred.
- From step 24 to step 30 we created the **glossy**/**specular** `Material_U0V0_Spec` node.
- From step 31 to step 34 we added the textures to the `Material_U0V0_Spec` material. This material should represent the **glossy**/**specular**/**mirror** component of the shader, that is probably the most important thing for obtaining a correct visual result; in **Cycles** the **Glossy BSDF** shader node provides the result perfectly, while in **Blender Internal**, we have two options: one, by enabling the (slow and imperfect) internal ray-tracing **Mirror** item, or by faking it. We faked it by setting an image (the same **hdr** we'll use in the next chapter in the **World** both for **Cycles** and **BI**) on the **Reflection** channel of the shader, hence giving the impression of an environment (slightly) mirrored by the character's skin.
- At step 35 we added together the outputs of the **COL** and of the **SPEC** nodes.
- From step 36 to step 40 we tweaked the output of the **SPEC** nodes to obtain a more realistic output/distribution of the glossiness on the mesh's surface, trying to mimic, as much as possible, the glossy output of the **Cycles** shader version.
- From step 41 to step 46 we built a fake **Fresnel** to work as a factor for the blending of the **glossy** and the **diffuse** components; this works by calculating the dot product of the vectors of the point of view and the mesh's normals. Be aware that it isn't actually working as the real **Fresnel** node that you find in **Cycles**, and that it has several limitations. By varying the second **Value** of the **Math** node and/or the black color stop position of the **ColorRamp** node, we can obtain several nice effects in some way visually similar to the real output. If you are wondering why we don't use the output of a **BI** material with a **Fresnel** diffuse shader model, sadly it doesn't seem to work correctly (actually, it doesn't seem to work at all).
- From step 47 to step 51 we built the **SSS** material node by duplicating the **COL** node, making a new copy of the material and renaming it as `Material_U0V0_SSS`, then enabling **Subsurface Scattering** and modifying the influence values of the textures; the values of the subsurface scattering (**IOR**, **Scale**, **RGB Radius**), instead, were borrowed from the **Cycles** version of the shader.
- From step 52 to step 54 we added the output of the **SSS** node to the rest of the shader by using a **Blend Type** set to **Screen** plus a **Mix** one set to a low **Fac** value; basically, the exact copy of what we did in **Cycles**.

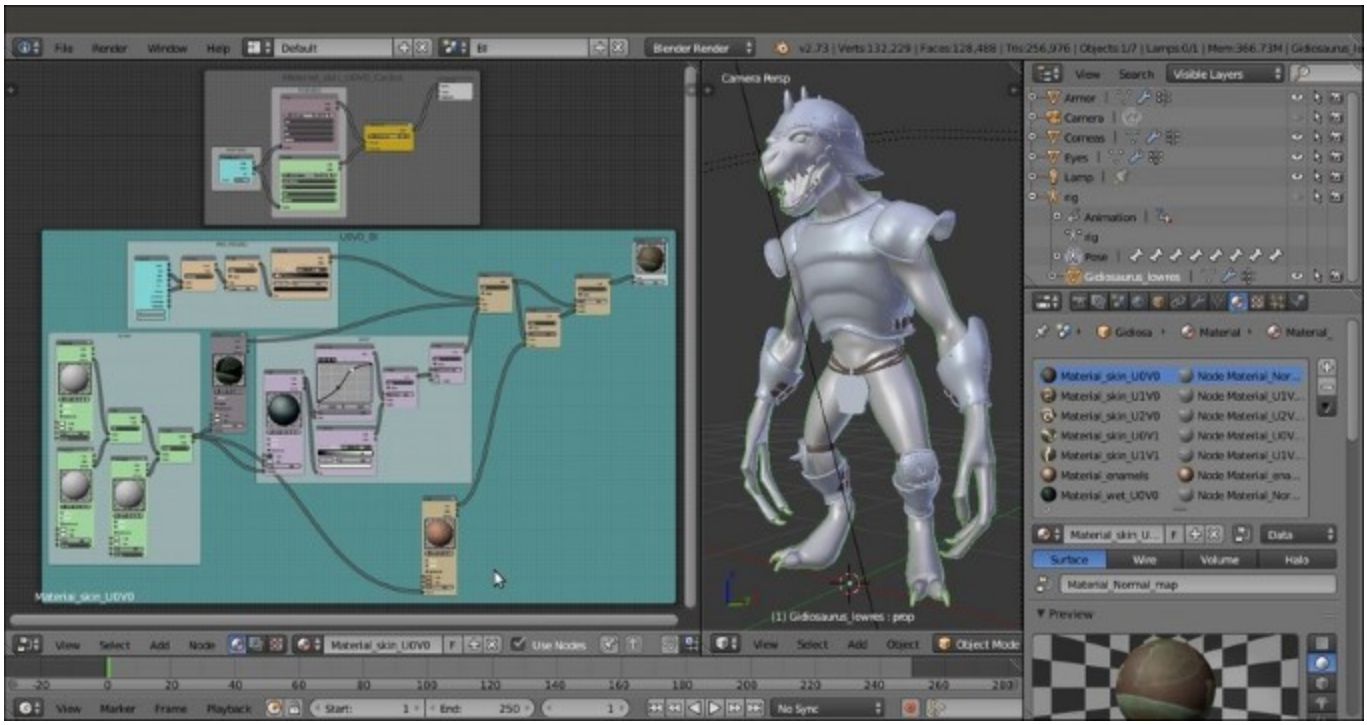
- From step 55 to step 71 we created the **bump pattern**, divided into **three** different material nodes to give us more flexibility in adding and averaging them together in a way that is as similar as possible to **Cycles**:

The F12 rendered final result in Blender Internal

There's more…

The other missing shaders for the **Gidiosaurus** skin are solved in exactly the same way we used in the previous chapter for the other **Cycles** skin shaders: by selecting the entire **BI** frame with all the parented nodes and pressing *Ctrl* + *C* to copy them, then selecting the different material slots and pressing *Ctrl* + *V* to paste everything; in **Blender Internal**, we have to make the single materials inside the nodes single user one by one and then substitute the textures according to the **UDIM** tile the material corresponds to (`U1V0_col.png`, `U1V0_scales.png`, `U2V0_col.png`, and so on).

So, in the end, each material will have *two sets of nodes*, one for the **Cycles** shader and one for the **Blender Internal** shader, and each one works under the respective render engine; this will be useful, as we'll see in the next chapter, for the rendering stage.



Two different sets of nodes for the same material

See also

- http://www.blender.org/manual/render/blender_render/materials/index.html
- http://www.blender.org/manual/render/blender_render/textures/index.html

Building the eyes' shaders in Blender Internal

We'll now see how to make the shaders for the **Gidiosaurus's eyes**; they are composed of two objects, the **Corneas** and the **Eyes** objects, so let's start with the first one.

Getting ready

Enable the **6th** and the **12th** scene layers and select the **Corneas** object; in the **Outliner**, disable the **Eyes** object's visibility in the viewport to hide it, put the mouse pointer inside the **Camera** view, zoom to one of the **eyeballs** and then start the **Rendered** preview.

How to do it…

Let's start to create the `Corneas` material:

1. Put the mouse pointer in the **Node Editor** window and add: a **Material** node (*Shift* + *A* | **Input** | **Material**), a **Geometry** node (*Shift* + *A* | **Input** | **Geometry**), a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and an **Output** node (*Shift* + *A* | **Output** | **Output**).
2. Connect the **Color** output of the **Material** node to the **Color** input socket of the **Output** node, then click on the **New** button on the **Material** node to create a new material and rename it `Cornea_bump`.
3. In the **Material** window, expand the **Render Pipeline Options** subpanel and enable the **Transparency** item; in the **Diffuse** subpanel set the shader model to **Oren-Nayar** and the color to a bright orange = **R 0.930**, **G 0.386**, **B 0.082**. In the **Specular** subpanel set the shader model to **WardIso** and the **Slope** to **0.070**. In the **Shading** subpanel enable the **Cubic Interpolation** item.
4. Go down to expand the **Transparency** subpanel; set the **Fresnel** value to **1.380** and the **Blend** to **1.700**.
5. Go further down to enable the **Mirror** item in the subpanel with the same name, set the **Reflectivity** to **0.200**, the **Fresnel** to **1.380** and the **Blend** to **1.500**.
6. Go to the **Texture** window and in the **first** slot load the image `eyeball_col.jpg`, rename the ID datablock as `eyeball_col` and set the **UVMap.001** as the coordinates layer; in the **Influence** subpanel set the diffuse **Color** channel to **0.200** and the specular **Color** channel to **0.200** as well, set the **Blend Type** to **Color**.
7. In the **second** texture slot load the image `eyeball_bump.jpg`, rename the ID datablock as `Eyeball_bump`, set **UVMap.001** as the coordinates layer and disable the **Color** channel to enable the **Normal** one at **0.007**; set the **Bump Method** to **Best Quality**:

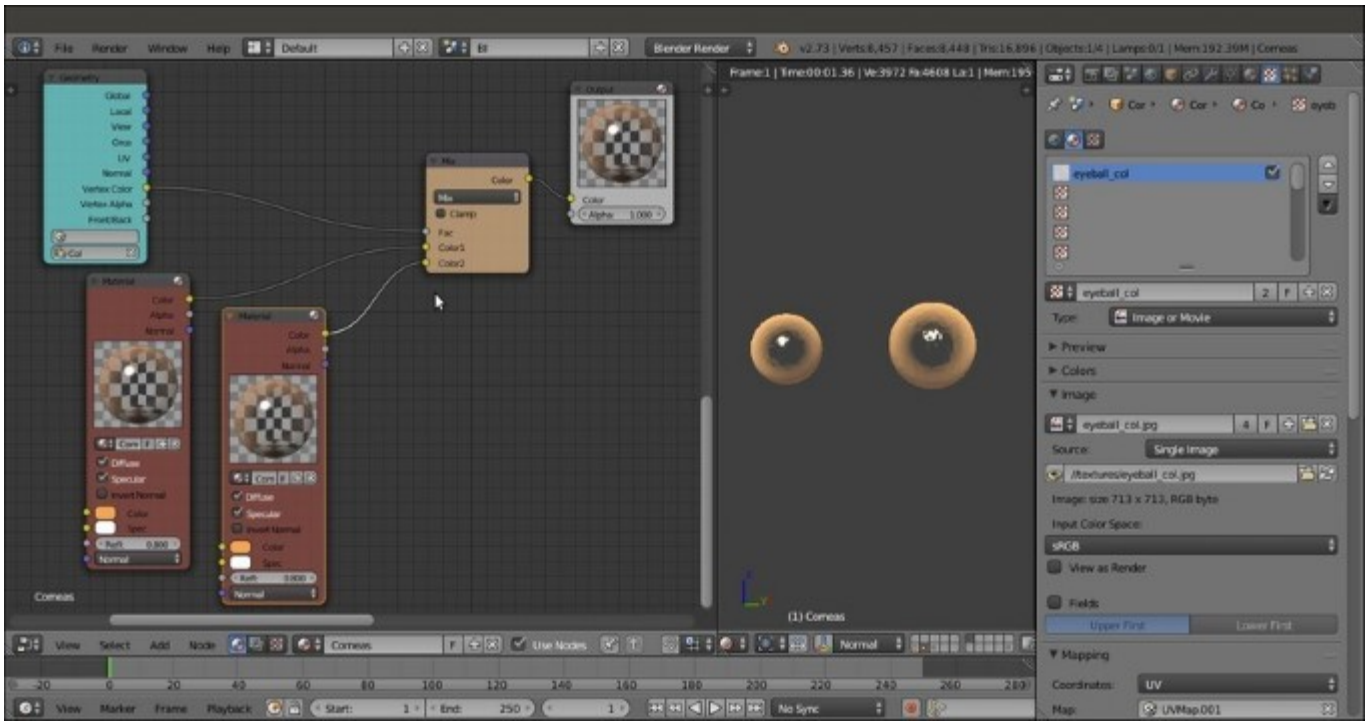
The "Corneas" material nodes

8. Paste the **MixRGB** node between the **Material** and the **Output** nodes; then, connect the **Vertex Color** output of the **Geometry** node to the **Fac** input socket of the **MixRGB** node. Click on the last empty field at the bottom of the **Geometry** node to select the **Col** item from the pop-up list (it's the name of the **Vertex Color** layer that has been created, and mentioned, at the beginning of the *Building the eyes' shaders in Cycles* recipe in Chapter 12, *Creating the Materials in Cycles*).

The gray color of the Color2 socket of the MixRGB node showing in the rendered preview at the location established by the Vertex Color layer output used as factor

In the preceding screenshot, the effect of the **Vertex Color** layer is visible: the two gray dots on the eyeballs are actually the crystalline lens areas filled, only at the moment, with the gray color of the empty **Color2** input socket of the **MixRGB** node.

9. Press *Shift* + *D* to duplicate the **Material** node, click on the **2** icon button to the right side of the name datablock of the duplicated node to make the material single user, rename the new material, simply, `Cornea` and, in the **Texture** window, select the **second** slot texture, `eyeball_bump`, to click on the **X** icon button and delete it.
10. Connect the **Color** output of the second **Material** node to the **Color2** input socket of the **MixRGB** node.



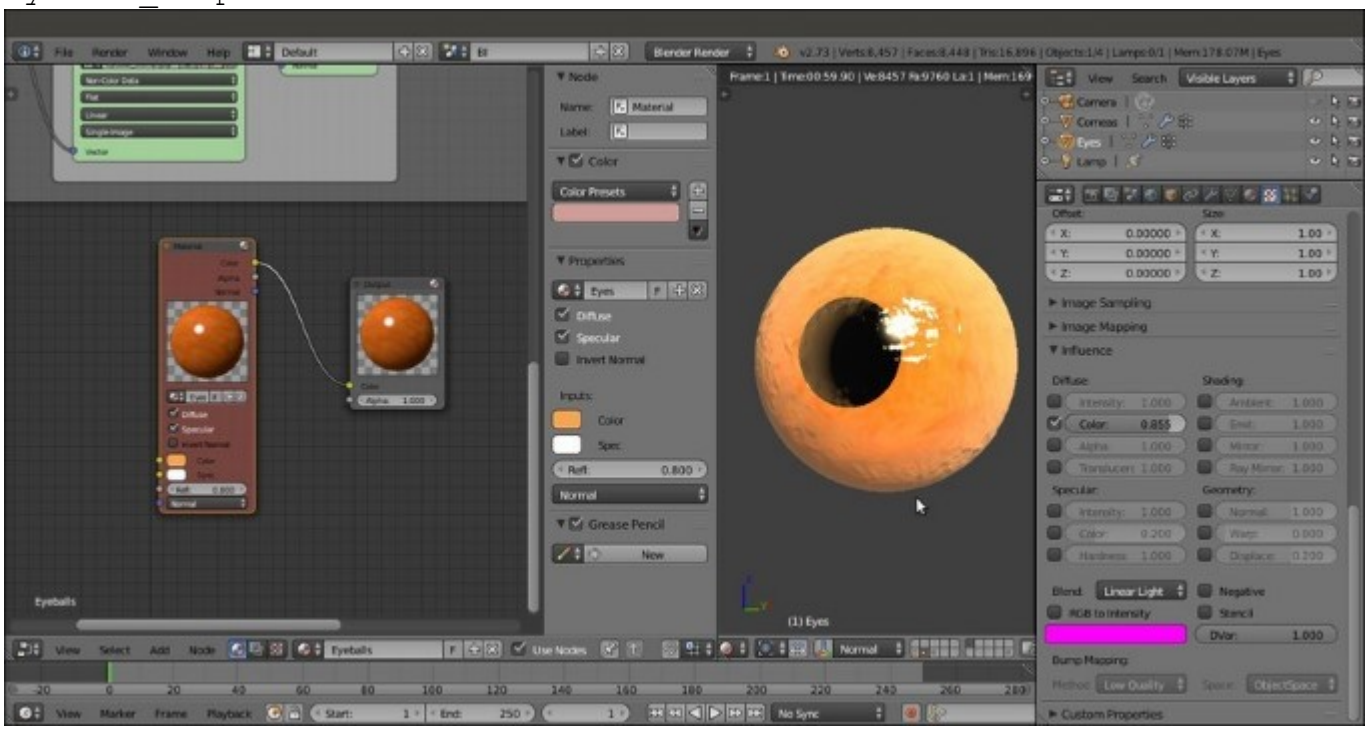
The completed "Corneas" material

Now, go to the **Outliner** and enable the **Eyes** object visibility in the viewport to show it:

11. With the **Corneas** object still selected, put the mouse pointer on the first **Material** node in the **Node Editor** window and press *Ctrl* + *C* to copy it.
12. Select the **Eyes** object and, in the **Material** window, select the `Eyeballs` material slot; put the mouse pointer in the **Node Editor** window and press *Ctrl* + *V* to paste the material node we copied before.
13. Click on the **2** icon button to make the material single user and rename it `Eyes`. Add an **Output** node (*Shift* + *A* | **Output** | **Output**) and connect the **Color** output of the **Eyes** material node to the **Color** input socket of the **Output** node.
14. Go to the **Material** window and disable the **Transparency** item in the **Render Pipeline Options** subpanel; go to the **Mirror** subpanel and disable it.
15. Enable the **Subsurface Scattering** subpanel: set the **IOR** to **1.340**, the **Scale** to **0.001**, the scattering color to the orange **R 0.930**, **G 0.386**, **B 0.082** and the **RGB Radius** to the **R 9.436**, **G 3.348**, **B 1.790** values.

The "Eyeballs" material SSS settings

16. Go to the **Texture** window; select the `eyeball_col` texture and set the diffuse **Color** to **0.855**, disable the specular **Color** channel and set the **Blend Type** to **Linear Light**; select the `eyeball_bump` texture and set the **Normal** channel slider to **0.005**.



The "Eyeballs" material texture settings

Now let's see the **iris**:

17. Box-select both the `Eyes` material node and the connected **Output** node and press *Ctrl* + *C* to copy them; go to the **Material** window and select the `Irises` material slot, then put the mouse pointer in the **Node Editor** window and press *Ctrl* + *V* to paste them.
18. Make the duplicated node's material single user and rename it `Iris`; change the **Diffuse** subpanel color to **R 0.429**, **G 0.153**, **B 0.000**, then go to the **Shading** subpanel and set the **Emit** value to **0.07**. Go to the **Subsurface Scattering** subpanel and change the scattering color to **R 0.220**, **G 0.033**, **B 0.032**.
19. Go to the **Texture** window and delete (unlink) the two texture slots. In the **first** slot, load the image `iris_col.jpg`, rename the ID datablock `iris_col`, and set **UVMap.001** as the UV coordinates layer. In the **second** slot, load the image `iris_bump.jpg`, rename as `iris_bump`, set **UVMap.001** as the UV coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the specular **Intensity** and **Hardness** channels with value **1.000**, then enable also the **Normal** channel with value **1.000**. Set the **Bump Method** to **Best Quality**.
20. In the **third** texture slot, load again the `iris_bump.jpg` image; rename the ID datablock as `iris_ST`. In the **Image Sampling** subpanel, under **Alpha**, disable the **Use** item and enable the **Calculate** item. Set the **UVMap.001** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable only the **Stencil** item at the bottom:

The "Irises" material and the Stencil item

21. In the **fourth** texture slot, load the `iris_col.jpg` image, rename the ID datablock as `iris_emit`. Set the **UVMap.001** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the **Emit** channel at value **1.000**.

The "Irises" material emitting (fake) light

Regarding the `Pupils` material, it's a simple basic material with pure black as **Diffuse** color and the **Intensity** slider under the **Specular** subpanel set to **0.000** to be totally matte.

22. Save the file.

How it works…

For the **Corneas** object: we created two copies of the same transparent material, but then we removed the bump from one of them, because usually a **cornea** has bumps due to the veins on the **eyeball** but not on the **crystalline lens**, that is smooth; the two materials are mixed, exactly as in the **Cycles** version, through the output of the **Col Vertex Color** layer.

Regarding the `Irises` material: the `iris_ST` texture, set as a **stencil map**, works as a mask for the following texture to appear through its black areas.

Although it could have been solved by simply leaving a blank material slot, I assigned a black matte material to the **pupils**; I preferred to assign a material anyway, to avoid possible issues in the following stages such as, for example, in the rendering of the character against an alpha backdrop, of the separated passes and in the compositing.

Note that the `Corneas` is the only material where I enabled the ray-tracing mirror, which in the character's **skin** and in the **armor** are instead faked, to obtain faster rendering times (the **eyes** are really a small surface to be rendered).

Building the armor shaders in Blender Internal

We arrive finally at making the **Armor** shaders under the **Blender Internal** engine; we have **four** materials, here: the two **UDIM** plate shaders, the rivets shaders and the leather material for the tiers.

Getting ready

Enable the **6th** and the **13th** scene layers and select the **Armor** object; if your computer is powerful enough, use the **Rendered** preview while you are working.

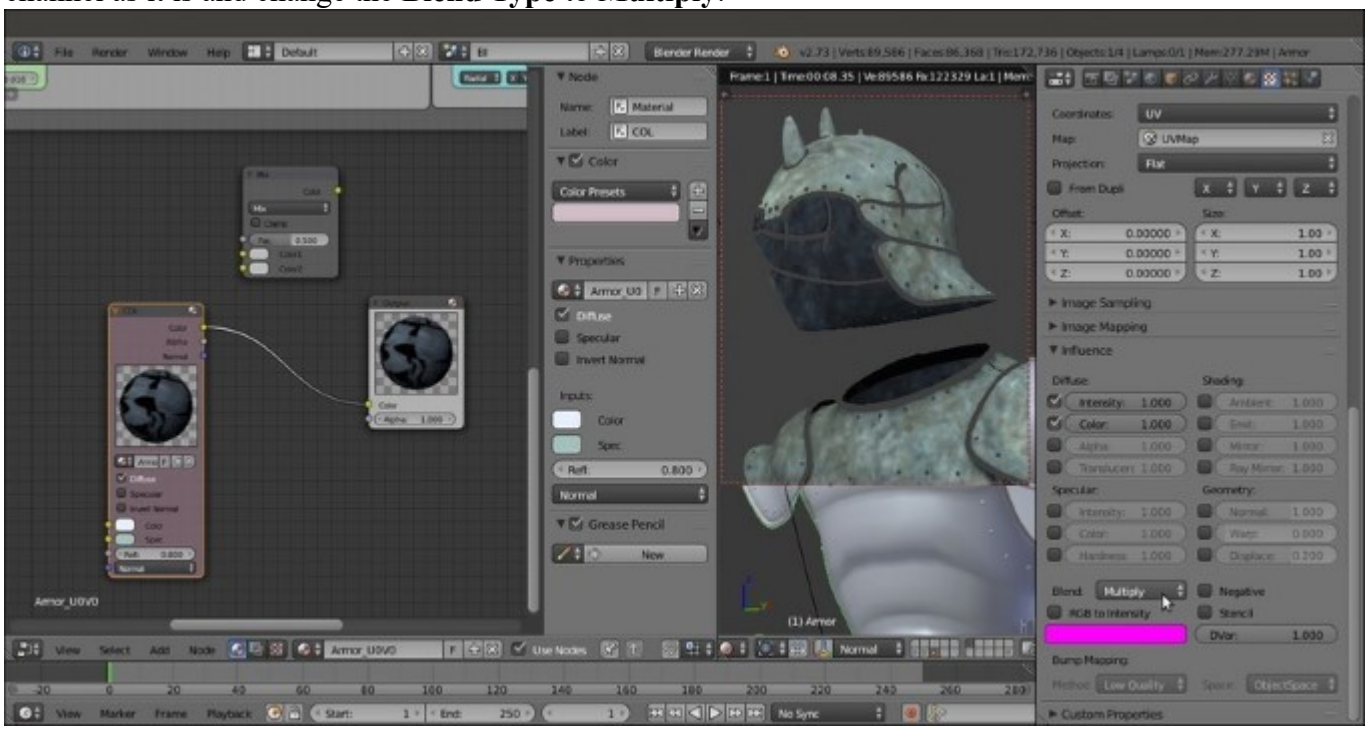
How to do it…

Let's start with the first **UDIM** tile material creation, the main **armor** plates:

1. Put the mouse pointer in the **Node Editor** window and add a **Material** node (*Shift* + *A* | **Input** | **Material**), a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and an **Output** node (*Shift* + *A* | **Output** | **Output**). In the *N* **Properties** sidepanel, label the **Material** node as **COL**.
2. Connect the **Color** output of the **COL** node to the **Color** input socket of the **Output** node, then click on the **New** button on the **Material** node to create a new material and rename it `Armor_U0V0_col`; in the **Node Editor** window, disable the **Specular** item.
3. Go to the **Material** window and find the **Diffuse** subpanel; change the shader model to **Oren-Nayar**, set the color to **R 0.817**, **G 0.879**, **B 1.000** and the **Roughness** value to **0.313**.
4. Go down to the **Specular** subpanel: set the shader model to **WardIso**, the **Intensity** to **1.000**, the **Slope** to **0.270**, and the color to **R 0.381**, **G 0.527**, **B 0.497**. In the **Shading** subpanel enable the **Cubic Interpolation** item.

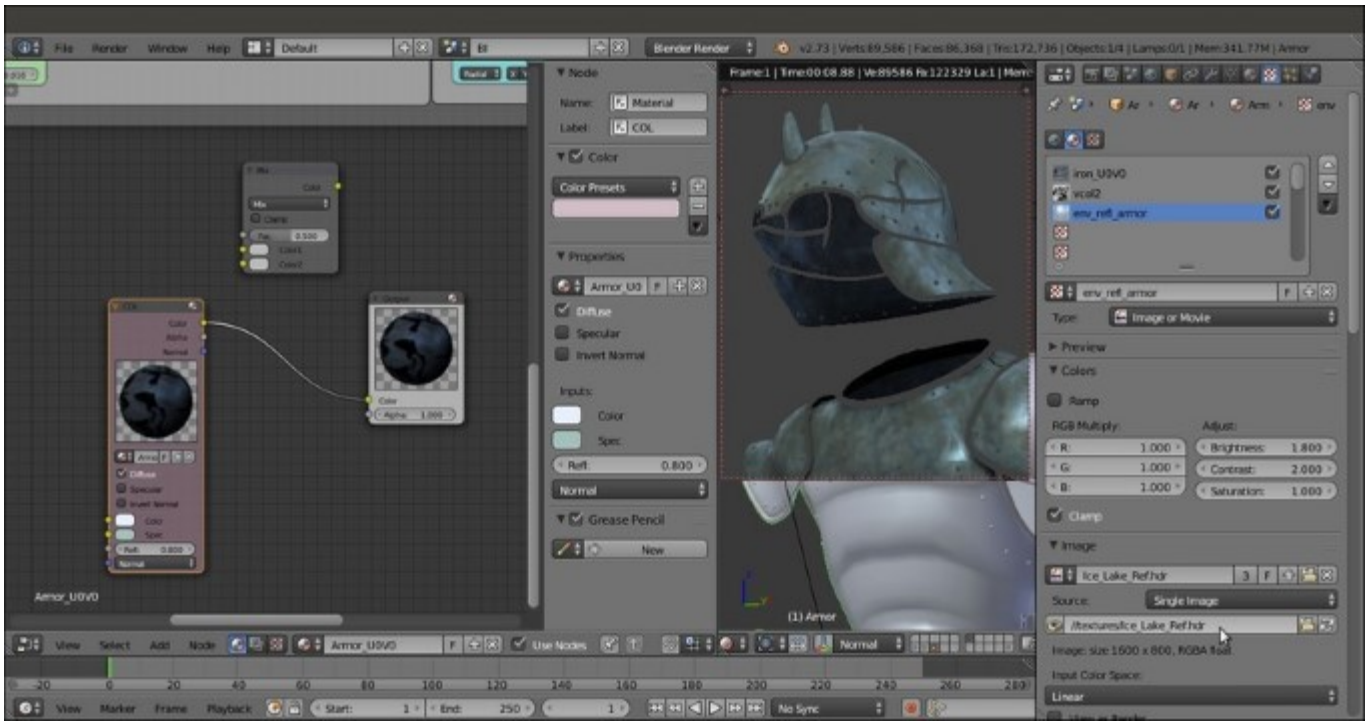
Starting the "Armor_U0V0" material in Blender Internal

5. Go to the **Texture** window and in the **first** texture slot, load the image `iron_U0V0.png`; rename the ID datablock as `iron_U0V0` and set the **UVMap** coordinates layer, then go to the **Influence** subpanel and enable the diffuse **Intensity** channel at value **1.000**, leave the **Color** channel as it is and change the **Blend Type** to **Multiply**:



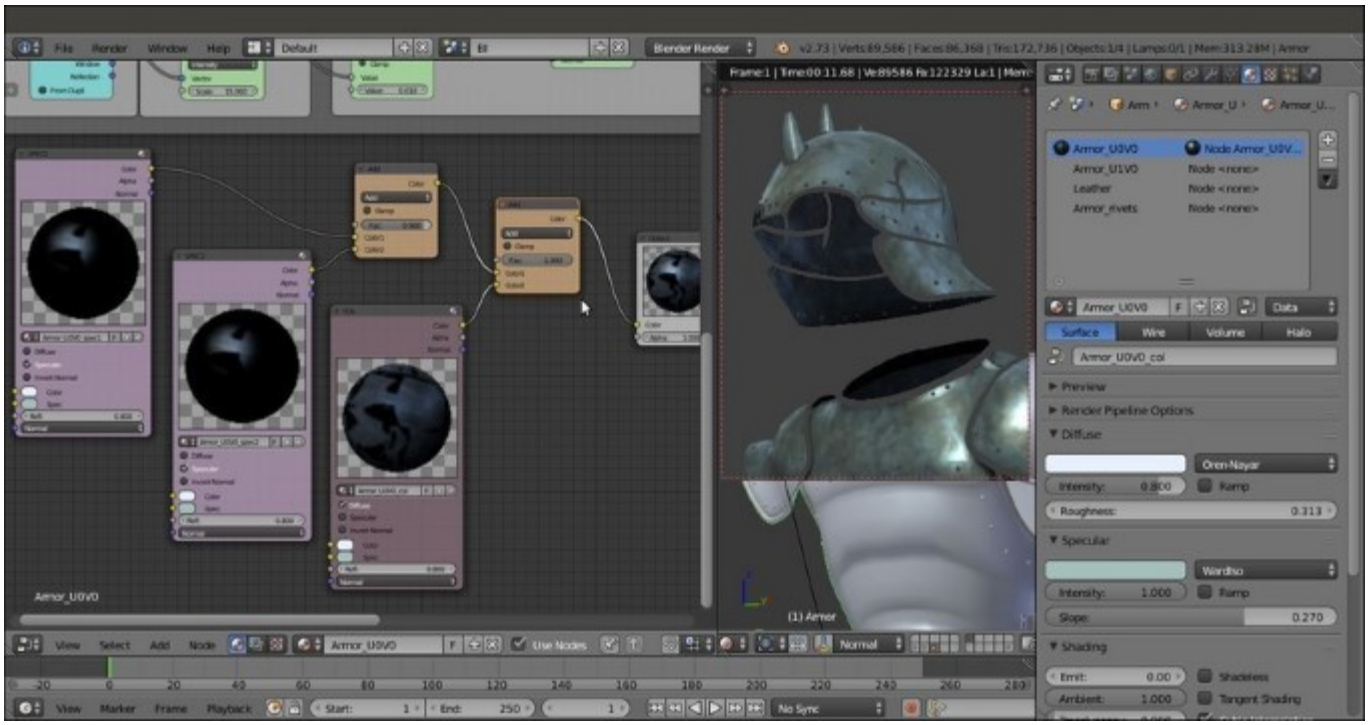
Adding the first texture image

6. In the **second** texture slot, load the image `vcol2.png`, rename the ID datablock as `vcol2` and set the **UVMap_norm** UV coordinates layer; go to the **Colors** subpanel, enable the **Ramp** item and set the **Interpolation** to **B-Spline**, then move the black color stop to position **0.245** and the white color stop to position **0.755**. In the **Image Sampling** subpanel set the **Filter Size** to **1.10** and in the **Influence** subpanel enable the diffuse **Intensity** channel at value **0.500**, the diffuse **Color** channel at **0.300**, set the **Blend Type** to **Difference** and enable the **Negative** item.
7. In the **third** texture slot, load the image `Ice_Lake_Ref.hdr` and rename the ID datablock as `env_refl_armor`. Set the **Mapping** coordinates to **Reflection** and, in the **Image Sampling** subpanel, the **Filter Size** to **6.00**; in the **Colors** subpanel set the **Brightness** to **1.800** and the **Contrast** to **2.000**, then move to the **Influence** subpanel and set both the diffuse **Intensity** and **Color** channels to **0.600** and the **Blend Type** to **Multiply**:



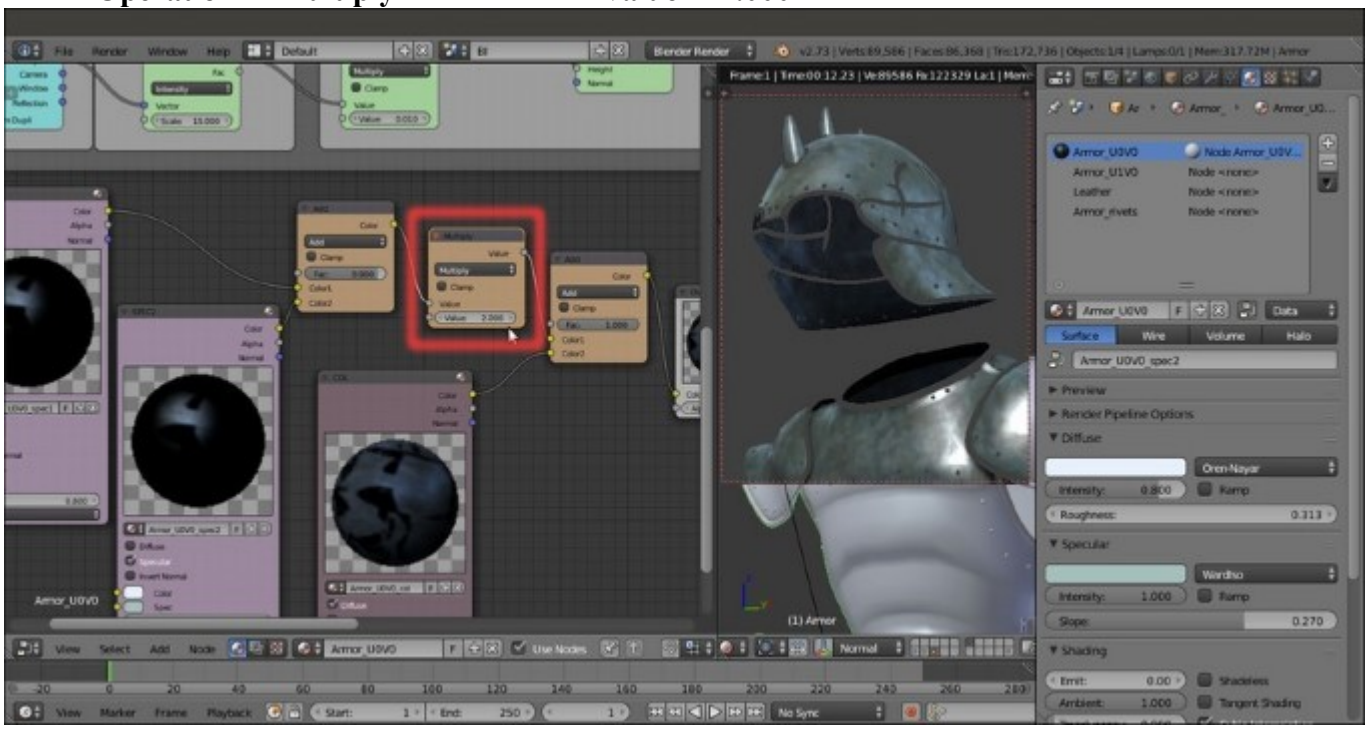
Adding the hdr image as reflection map

8. Go to the **Node Editor** window and press *Shift* + *D* to duplicate the **COL** node; label the duplicate as **SPEC1**, then make the material single user and rename it `Armor_U0V0_spec1`; disable the **Diffuse** item on the node interface and enable back, the **Specular** one.
9. Go to the **Texture** window; select the first `iron_U0V0` texture slot and go straight to the **Influence** subpanel: disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity**, **Color** and **Hardness** channels at **1.000**. Set the **Blend Type** to **Mix**.
10. Select the **second** `vcol2` texture slot, disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity** channel at **0.300**.
11. Select the **third** `env_refl_armor` texture slot, disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity** and **Color** channels at **0.500**.
12. Press *Shift* + *D* to duplicate the **SPEC1** node and label the duplicated one as **SPEC2**: make the material single user and rename it as `Armor_U0V0_spec2`. In the **Material** window go to the **Specular** subpanel and set the **Slope** to the maximum = **0.400**.
13. Now, connect the **Color** output of the **SPEC1** material node to the **Color1** input socket of the **MixRGB** node and the **Color** output of the **SPEC2** node to the **Color2** input socket; set the **Blend Type** of the **MixRGB** node to **Add** and the **Fac** value to **0.900**.
14. Press *Shift* + *D* to duplicate the **MixRGB** node and connect the output of the first **MixRGB** node to the **Color1** input socket of the duplicated **MixRGB** node, and the **Color** output of the **COL** node to the **Color2** input socket; set the **Fac** value of the second **MixRGB** node to **1.000** and connect its output to the **Color** input socket of the **Output** node.



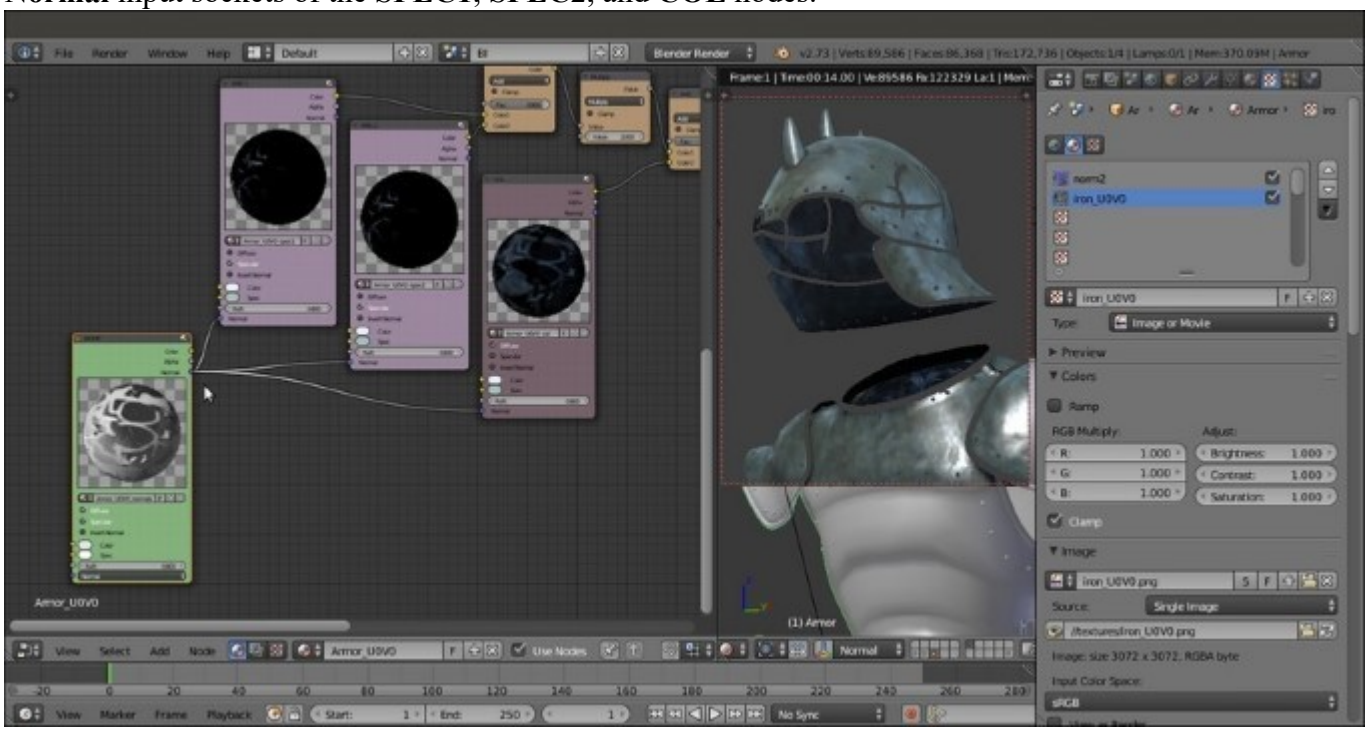
Adding the specular component

15. Add a **Math** node (*Shift + A* | **Converter** | **Math**) and paste it between the two **MixRGB** nodes; set the **Operation** to **Multiply** and the second **Value** to **2.000**.



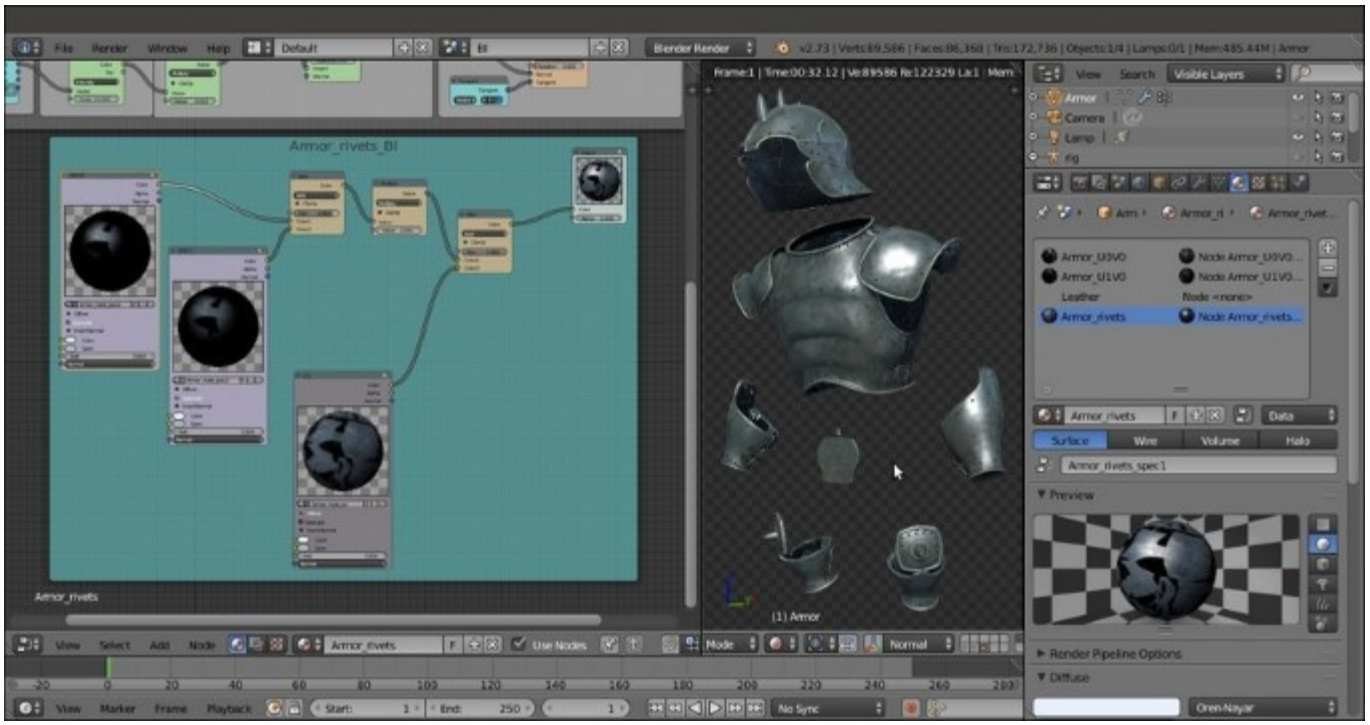
Enhancing the specularity

16. Add a **Material** node (*Shift* + *A* | **Input** | **Material**) and label it as **BUMP**; create a new material and rename it as `Armor_U0V0_normals`; in the **Shading** subpanel enable the **Cubic Interpolation** item.
17. Go to the **Texture** window and in the **first** texture slot, load the image `norm2.png`, rename the ID datablock as `norm2` and in the **Image Sampling** subpanel enable the **Normal Map** item; in the **Mapping** subpanel set the **UVMap_norm** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the **Normal** one at **0.500**.
18. In the **second** texture slot, load the image `iron_U0V0.png`, mapping to **UVMap** layer and **Influence** to **Normal** at **0.010**; set the **Bump Method** to **Best Quality**.
19. Go to the **Node Editor** window and connect the **Normal** output of the **BUMP** node to the **Normal** input sockets of the **SPEC1**, **SPEC2**, and **COL** nodes.



Adding the bump pattern

20. Box-select and press *Ctrl* + *C* to copy all these nodes, go to the **Material** window to select the `Armor_U1V0` material slot and, back in the **Node Editor** window, paste the copied nodes: then make the materials inside the nodes as single users, rename them accordingly and go to the **Texture** window to substitute the `iron_U0V0.png` image with the `iron_U1V0.png` image.
21. Copy and paste again, the nodes for the `Armor_rivets` material slot, but don't substitute the texture image: instead, simply delete the **BUMP** node, which wouldn't be of any use in such small parts.



The completed armor shaders in Blender Internal

22. Save the file.

How it works…

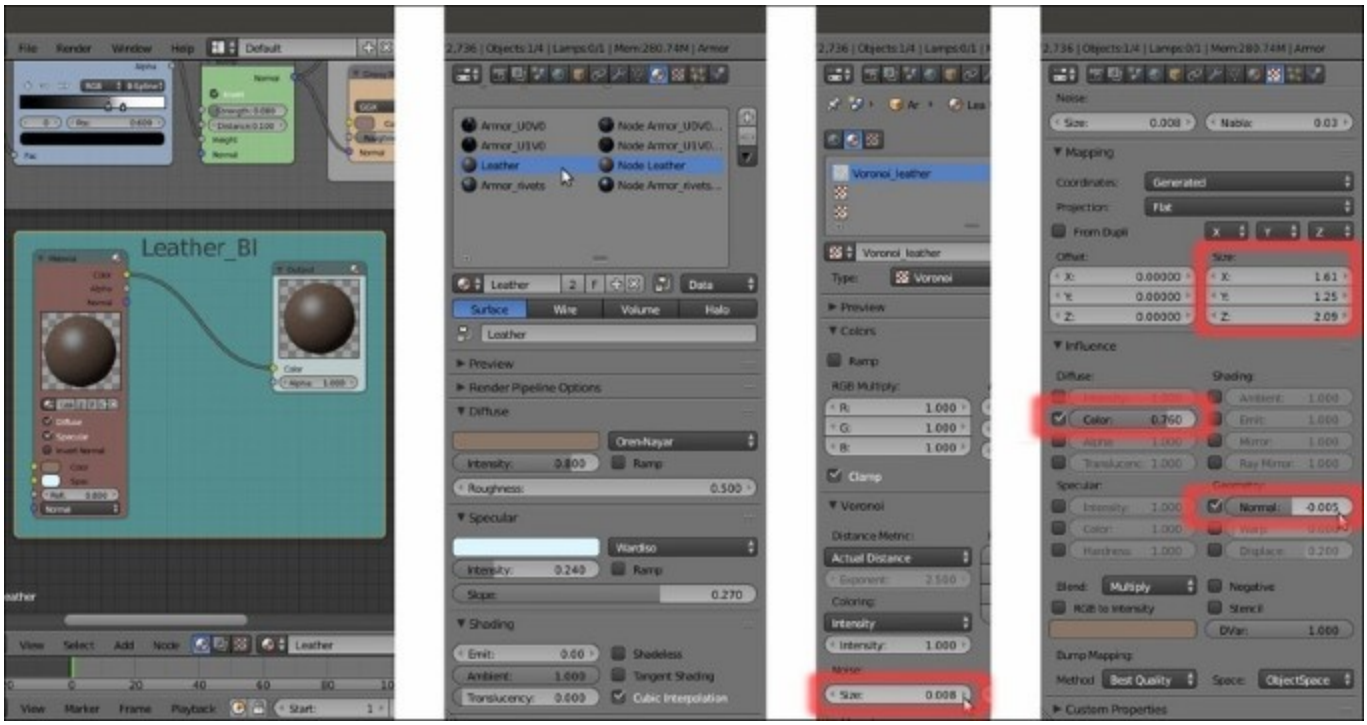
These shaders work, and have been built, exactly the same way as for the **Gidiosaurus' skin** and **eyes**; the only thing worth noting here is the order of the normal map and of the texture used for the bump pattern: in fact, to work together, in **Blender Internal**, the normal map must be placed higher in the texture stack, otherwise it will overwrite the effect of the bump map.

There's more…

The `Leather` material is a simple basic **Oren-Nayar** diffuse shader with the usual **WardIso** specular shader model, provided with a bump effect obtained through a **Voronoi** procedural texture with default values, except for the **Size**.

Note that the **Voronoi** texture influences the **Color** channel with the **Multiply** blend type and the **Normal** channel with a **negative** low value to obtain an actual bulging out pattern, instead of a concave one; negative values, in fact, reverse the *direction* of the bump.

The size of the procedural texture along the three axes is also further tweaked in the **Mapping** subpanel, scaling the three axes differently to resemble the dimensions of the **Texture Space** (basically the mesh bounding box, than can be made visible through the subpanel of the same name in the **Object Data** window), in order to avoid stretching along the mesh.



The "Leather" material in Blender Internal

Note also that in all these **Blender Internal** materials, simple mono materials (as for example the `Leather BI` material shown here earlier) are loaded inside a **Material** node and then connected to an **Output** node in the **Node Editor** window even if this wouldn't be necessary for the material itself to work: but, to let the **Blender Internal** and the **Cycles** render engines work together (through the compositor, as we'll see in the next chapter), it is mandatory to have all the shaders as nodes.

Chapter 14. Lighting, Rendering, and a Little Bit of Compositing

In this chapter, we will cover the following recipes:

- Setting the library and the 3D scene layout
- Setting image based lighting (IBL)
- Setting a three-point lighting rig in Blender Internal
- Rendering an OpenGL playblast of the animation
- Obtaining a noise-free and faster rendering in Cycles
- Compositing the render layers

Introduction

In this last chapter, we are going to see recipes about the more common stages needed to render the complete final animation: lighting techniques in both the render engines, fast rendering previews, rendering settings, and the integrated compositing.

But first, let's see the necessary preparation of the 3D scene layout.

Setting the library and the 3D scene layout

In this recipe, we are going to prepare a little both the file to be used as the library and the *hero* blend file, which is the file that will output the final rendered animation.

Getting ready

Start Blender and load the `Gidiosaurus_shaders_Blender_Internal.blend` file:

1. Go to the **Object Modifiers** window and check that the **Armature** modifiers are correctly enabled for *all the objects* (that is, the **Armature** modifiers must be enabled both for the rendering and for the 3D viewport visibility), then save the file.
2. Press *Ctrl* + *N* and click on the **Reload Start-Up File** pop-up panel to confirm a new brand file: immediately save it as `Gidiosaurus_3D_layout.blend`.

   ### Tip

   Saving the file at this point is necessary to automatically have a *relative path* for all the assets we are going to link.

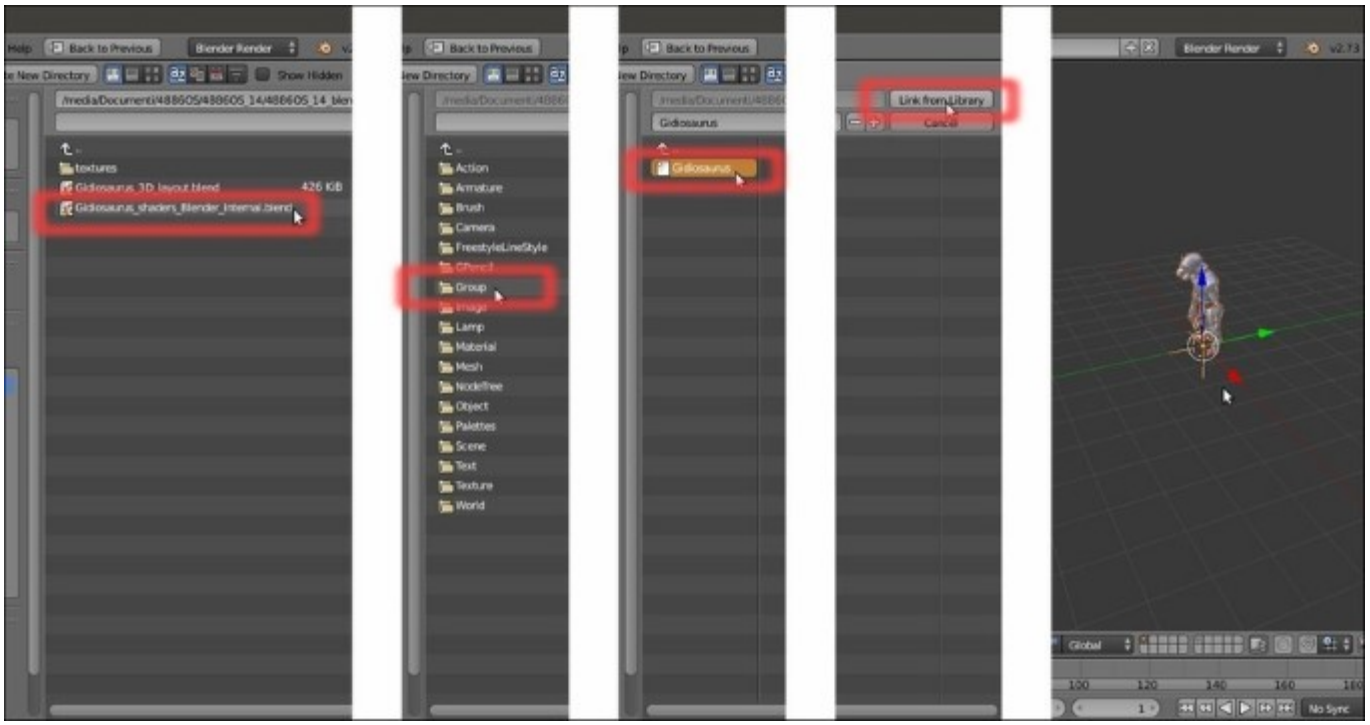
How to do it…

Let's load the assets as links in the file:

1. Select and delete (*X* key) the default **Cube** primitive in the middle of the 3D scene, then *Shift*-select both the **Camera** and the **Lamp** and move them (*M* key) to the **6th** scene layer.
2. Still in the **1st** scene layer, click on the **File** item in the top main header and then navigate to select the **Link** item; or else, just press the *Ctrl* + *Alt* + *O* keys shortcut.

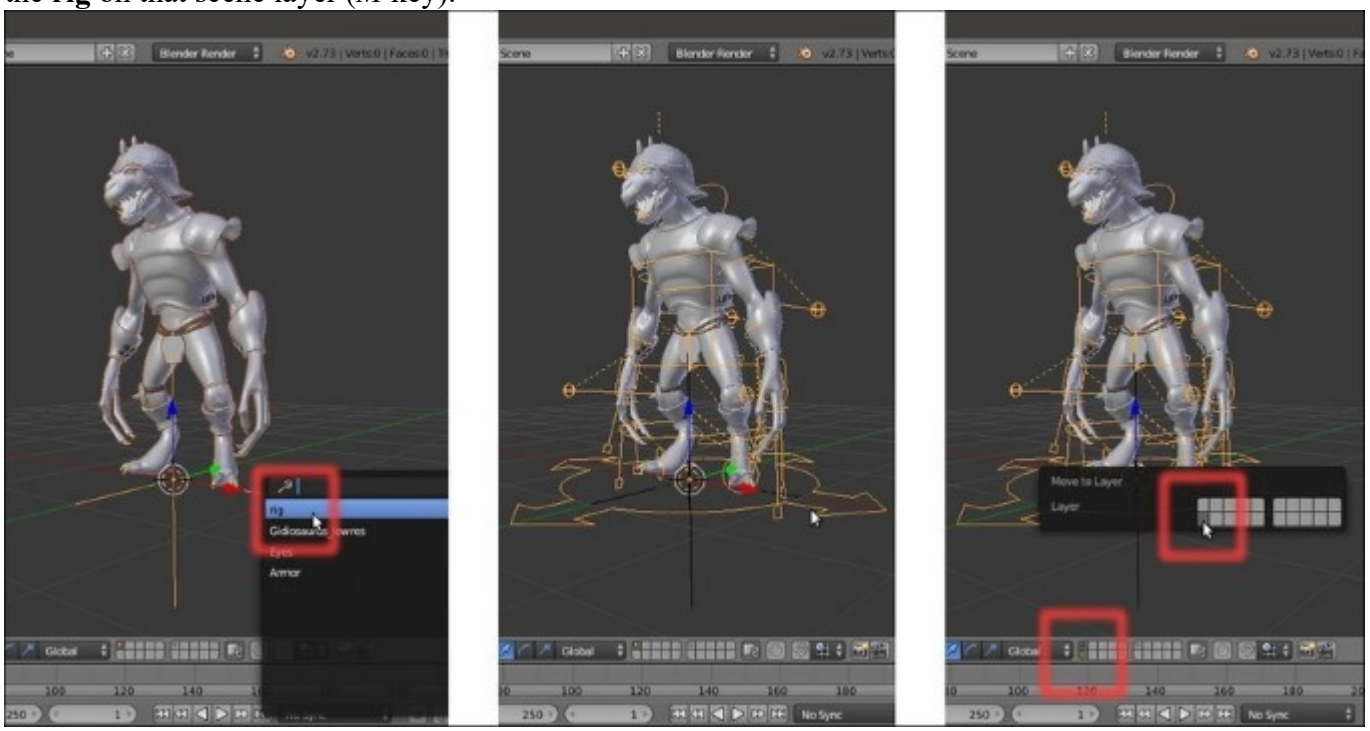
   In the blend files provided with this cookbook, I moved a copy of the `Gidiosaurus_shaders_Blender_Internal.blend` file to the `4886OS_14_blendfiles` folder, to simplify the process, but anyway:
3. Browse to the folder where the `Gidiosaurus_shaders_Blender_Internal.blend` file is saved; click on it and browse further to click on the **Group** item/folder, then select the **Gidiosaurus** item and click on the top right **Link from Library** button.

   The linked **Gidiosaurus** character appears at the **3D Cursor** position, in our case in the middle of the scene:



Linking the Gidiosaurus group

4. Zoom to the **Gidiosaurus** object and press *Ctrl* + *Alt* + *P* to make a proxy; in the pop-up menu panel that appears, select the proxified **rig** item, then *Shift*-enable the **11th** scene layer and move the **rig** on that scene layer (*M* key):

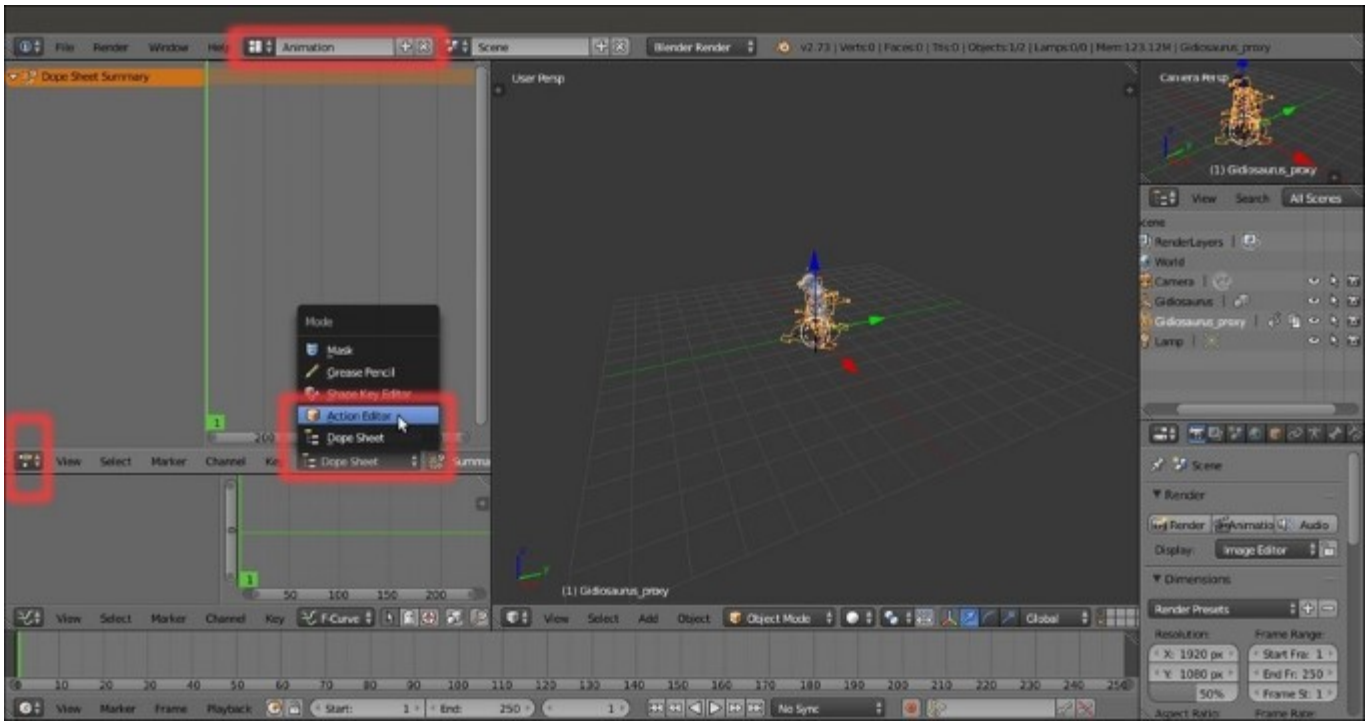


Making a proxy of the rig and moving it to the 11th scene layer

5. Click on the *Screen datablock* button on the top main header to switch from the **Default** screen layout to the **Animation** screen layout:

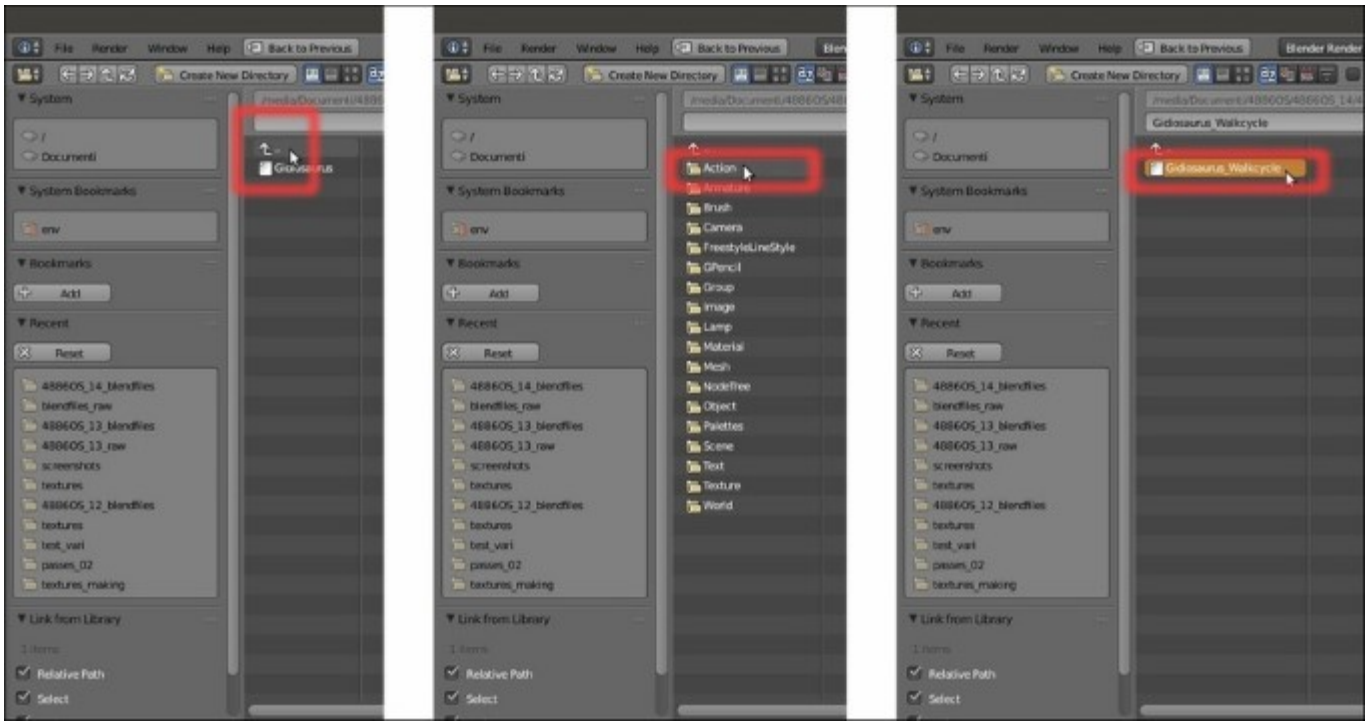
Switching to the Animation screen layout

6. Click on the **Mode** button in the **Dope Sheet** toolbar and switch from the **Dope Sheet** window to the **Action Editor** window:



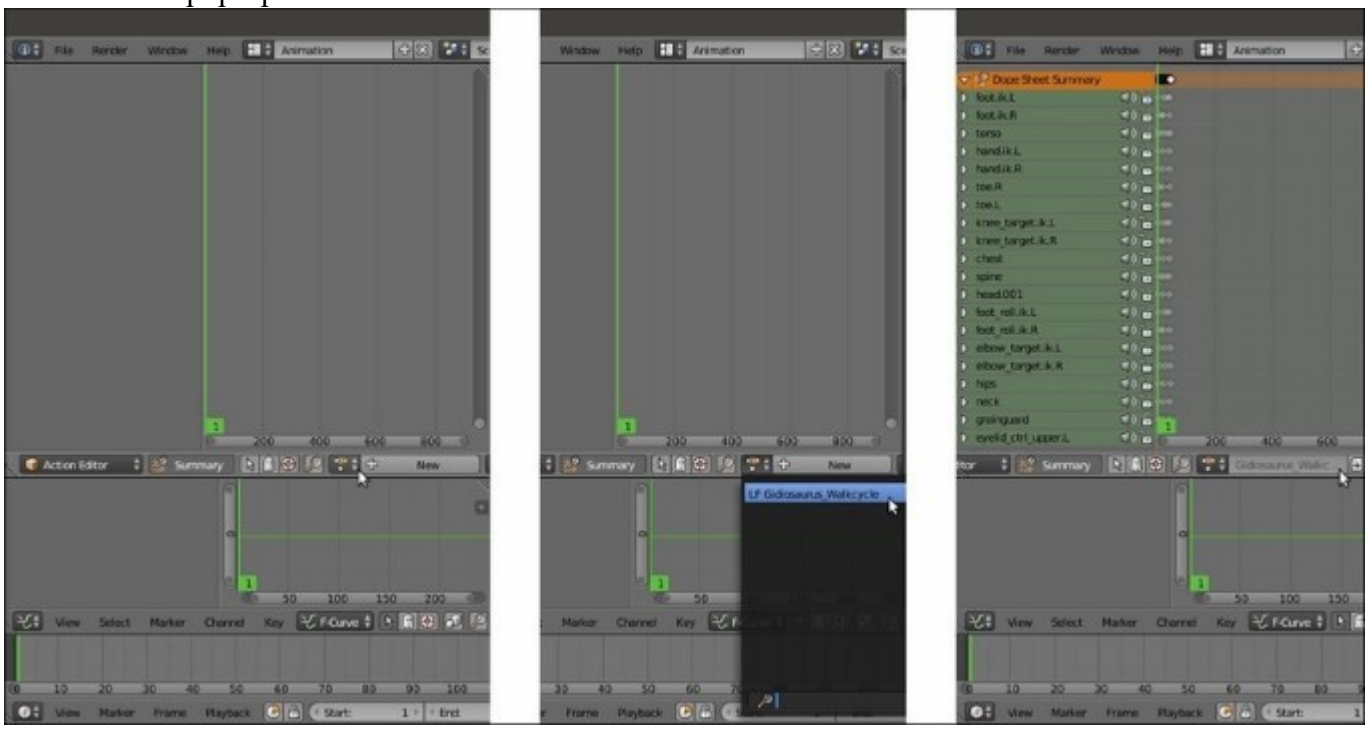
Switching to the Action Editor window

7. Click on the **File** item in the top main header and then navigate to select the **Link** item, or just press the *Ctrl* + *Alt* + *O* keys shortcut; the screen opens automatically at the last location we previously browsed to.
8. Click on the two dots above the **Gidiosaurus** item to navigate backward (to go up one level) and click on the **Action** item/folder to select the **Gidiosaurus_walkcycle** item; then click as before on the **Link from Library** button:



Browsing to the Action folder directory

9. Scroll a bit to the left of the **Action Editor** window's toolbar to reveal the **New** button; click on the double arrows to the left side of the **New** button to select the **LF Gidiosaurus_walkcycle** item from the pop-up menu.



Loading the linked action in the Action Editor window

10. Go back to the **Default** screen and click on the **Play Animation** button in the **Timeline** toolbar.

    Depending on the power of your system, you will see the animated character start to move, more or less fluidly, in the 3D viewport; the frame-rate (number of frames per second) played by Blender in real-time is shown in red at the top left corner of the 3D view-port:

The 3D view-port showing the animation and the frame-rate at the top left corner

    It should be around **24** frames per second; in my case, it barely arrives at **0.70** to **0.80**... so an arrangement must be found to show a faster and natural-looking movement.

11. Go to the **Scene** window and enable the **Simplify** subpanel: set the **Subdivision** level to **0**.

The Simplify subpanel under the Scene window in the main Properties panel

Without subdivision levels, even on my old laptop the real-time frame-rate is now **24** frames per second.

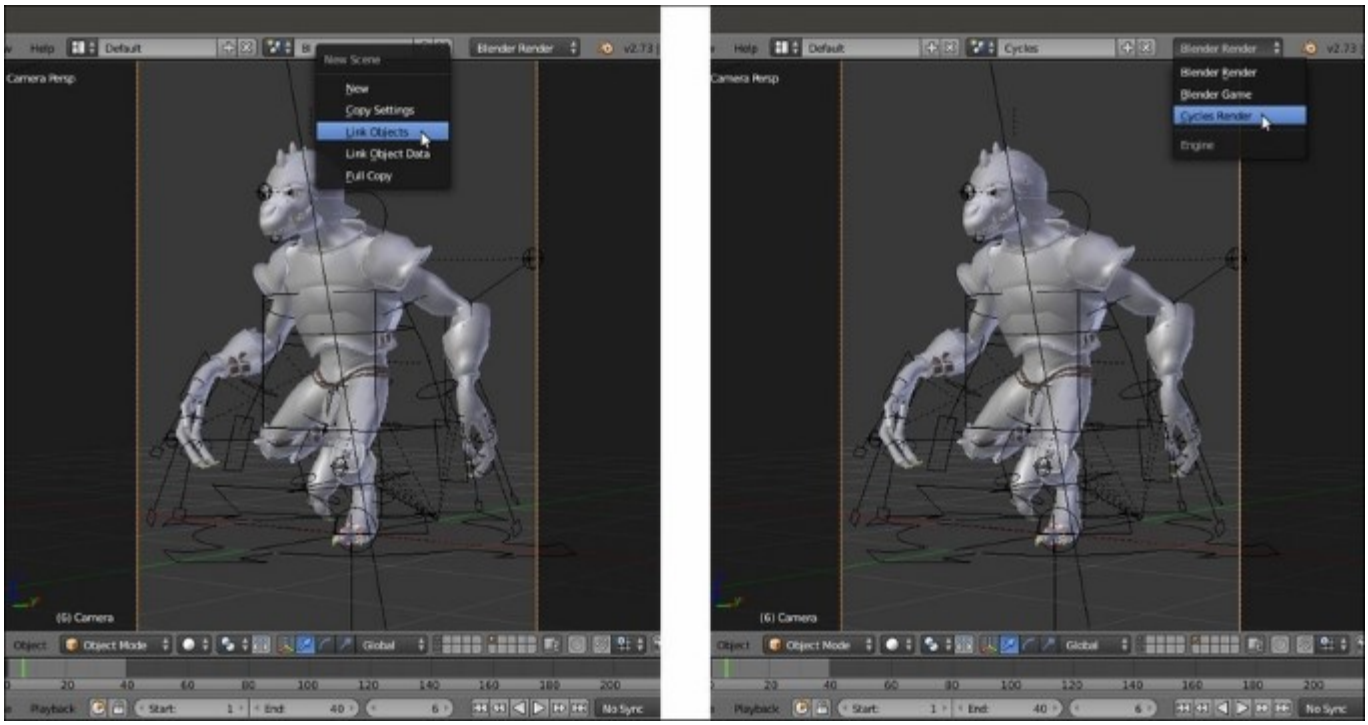
12. Go to the **Render** window and, in the **Dimensions** subpanel, under **Frame Range**, set the **End Frame** to **40**; under **Resolution** switch the **X** and **Y** values, that is **X** = **1080** px and **Y** = **1920** px.
13. Go to the **Outliner** and click on the **Display Mode** button to switch from **All Scenes** to **Visible Layers**. Then *Shift*-enable the **6th** scene layer and select the **Lamp**.
14. Go to the **Object Data** window and, in the **Lamp** subpanel, change the lamp type from **Point** to **Spot**; set the color to **R 1.000**, **G 1.000**, **B 0.650** and the **Energy** to **14.000**.
15. Put the mouse pointer in the 3D viewport and press *N* to call the side **Properties** panel; go to the top **Transform** subpanel and set **Location** as **X** = **6.059204**, **Y** = **-9.912249**, **Z** = **7.546275** and **Rotation** as **X** = **55.789°**, **Y** = **0°**, **Z** = **30.562°**.
16. Back in the **Object Data** window, go down to the **Shadow** subpanel and switch from **Ray Shadow** to **Buffer Shadow**; under **Filter Type** set the **Shadow Filter Type** to **Gauss**, the **Soft** to **12.000**, the **Size** to **4000** and the **Samples** to **16**. Set the **Clip Start** value to **9.000** and the **Clip End** value to **19.000**.

Setting the Lamp

17. Select the **Camera** and in the **Object Data** window set the **Focal Length** under **Lens** to **60.00**; in the **Transform** subpanel under the *N* side **Properties** panel input these values: **Location** as **X** = **5.095385**, **Y** = **-6.777483**, **Z** = **1.021429** and the **Rotation** as **X** = **91.168°**, **Y** = **0°**, **Z** = **37.526°**. Put the mouse pointer in the 3D viewport and press the *0* numpad key to go in **Camera** view:

Setting the Camera

18. Now click on the **Scene** datablock button on the top main header and rename it **BI** (**Blender Internal**). Click on the + icon button to the right and from the **New Scene** pop-up menu select the **Link Objects** item.
19. Change the **BI.001** name of the new scene in **Cycles** and click on the *Engine* button to the right to switch to the **Cycles Render** engine:



Adding a new scene with linked objects

20. Go to the **Outliner** and select the **Lamp**; go to the **Object Data** window and, in the **Nodes** subpanel, click on the **Use Nodes** button and then set the **Strength** to **10000.000**. Go to the **Lamp** subpanel and set the **Size** to **0.500** and enable the **Multiple Importance** item.
21. Save the file.

How it works…

A scheme of what we made in this recipe is: we prepared a blend file that *links* both the **character** and the **action**; this means that *neither* of them is *local* to the file and they cannot be directly edited in this file. Moreover, the character, in its library file, links the textures that are contained in the `textures` folder (which is at the same level as the blend files).

The **Simplify** subpanel in the **Scene** window allows us to *globally* modify some of the settings that can usually slow a workflow, such as the **subdivision** levels, the number of **particles**, the quality of **ambient occlusion** and **subsurface scattering**, and the **shadows** samples; through this panel they can be temporarily lowered or even disabled to have faster and more responsive previews of the rendering and the animation. Just remember that the **Simplify** subpanel also affects the rendering, so you have to disable it before starting the final rendering task.

See also

- http://www.blender.org/manual/data_system/introduction.html#copying-and-linking-objects-between-scenes
- http://www.blender.org/manual/data_system/scenes.html

- http://www.blender.org/manual/data_system/linked_libraries.html

Setting image based lighting (IBL)

The **image based lighting** technique is almost essential in computer graphics nowadays; as the name itself says, it's a technique to light a scene based on the pixel color information of an image, usually an **hdr** image (**High Dynamic Range** image); other image formats can also work, although not so well.

In Blender it's possible to obtain **IBL** both in **BI** and in **Cycles**, although with different modalities.

Getting ready

Start Blender and load the previously saved `Gidiosaurus_3D_layout.blend` file; save it as `Gidiosaurus_IBL.blend`.

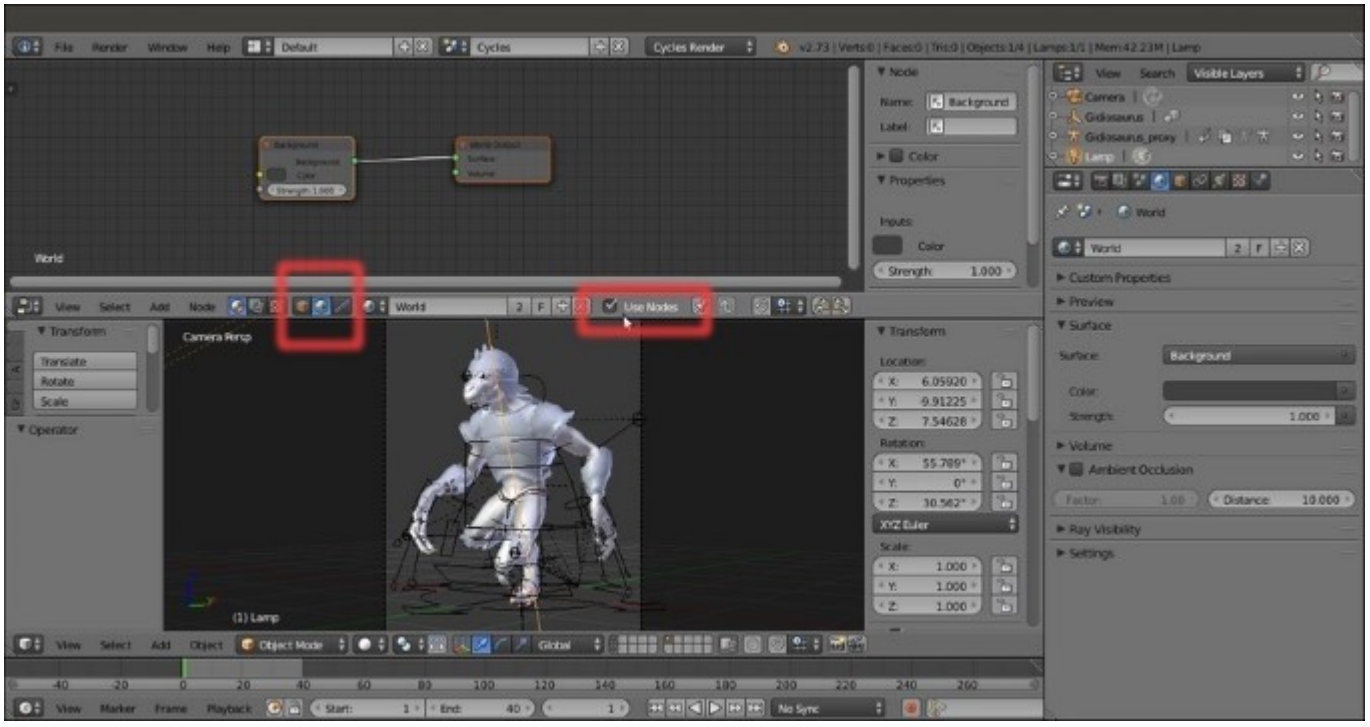
How to do it…

We can divide this recipe into two parts: **IBL** in **Cycles** and in **Blender Internal**.

Image based lighting in Cycles

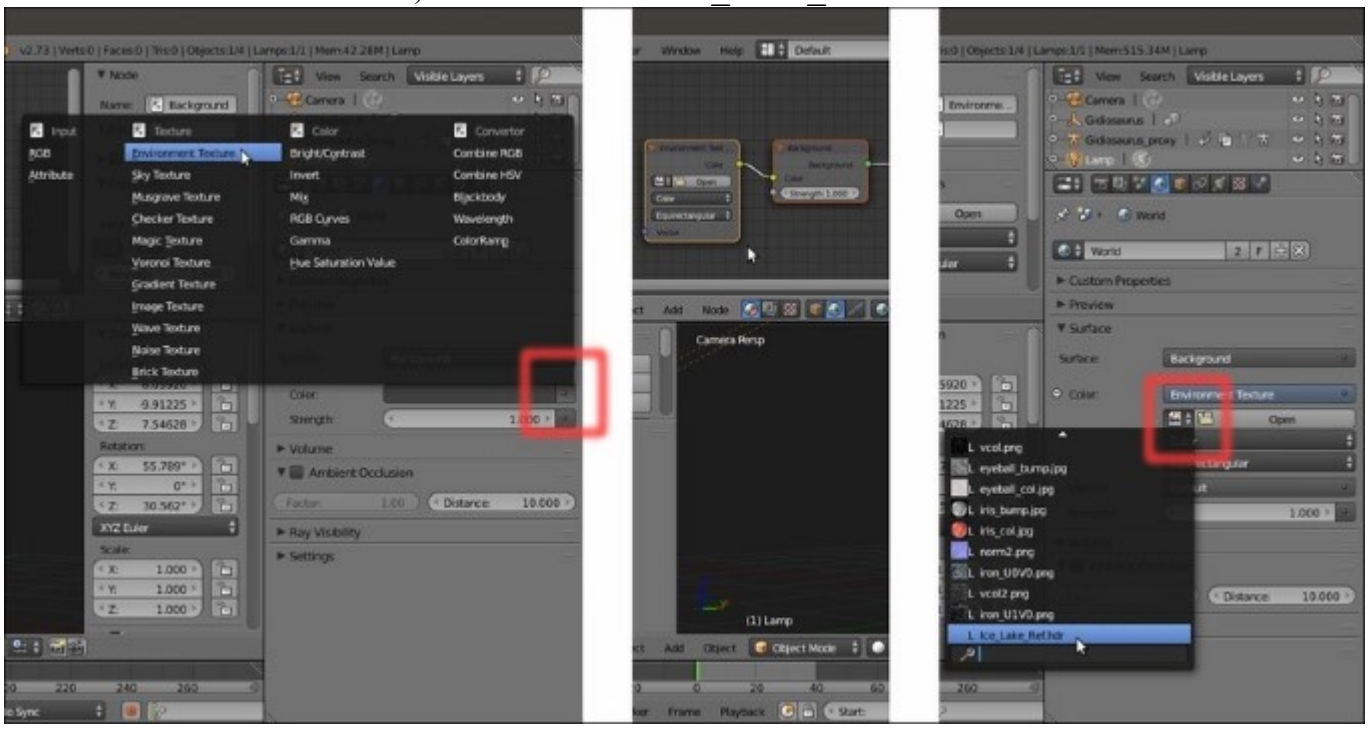
Let's start with the **Cycles Render** engine:

1. First, split the 3D window vertically into two windows, then change the upper one into a **Node Editor** window. In the toolbar, click on the **World** icon button to the right side of the **Object** icon button (selected by default; it's the one enabled for building the objects' shaders). Check the **Use Nodes** checkbox (or, click on the **Use Nodes** button inside the **Surface** subpanel in the **World** window); a **Background** node connected to a **World Output** node will appear in the **Node Editor** window.



Enabling the World nodes in the Node Editor window

2. Click on the dotted button to the right side of the **Color** slot in the **Surface** subpanel under the **World** window, to call the pop-up menu and select an **Environment Texture** node, which is automatically added and correctly connected to the **Color** input socket of the **Environment** node; then, click on the double arrows to the left side of the **Open** button (both in the **Node Editor** or in the **World** window) and select the `L_Ice_Lake_Ref.hdr` item.



Adding an Environment node to the World and loading the hdr image

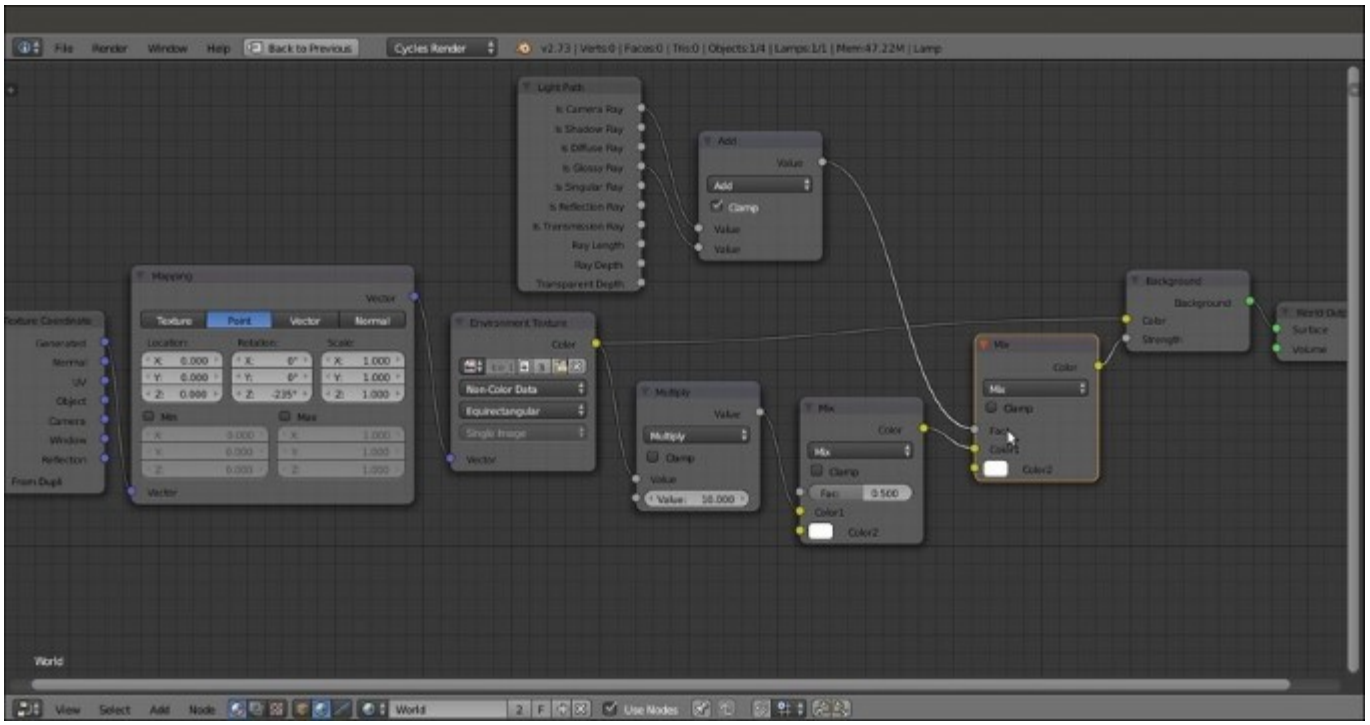
3. In the **World** window or in the **Node Editor** window, set the **Color Space** to **Non-Color Data**.
4. In order to gain some feedback, start the **Rendered** preview in the bottom **Camera** view, then go back to the **Node Editor** and add a **Texture Coordinate** node (*Shift* + *A* | **Input** | **Texture Coordinate**) and a **Mapping** node (*Shift* + *A* | **Vector** | **Mapping**).
5. Connect the **Generated** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node and the output of this latter node to the **Vector** input socket of the **Environment Texture** node; set the **Rotation Z** value of the **Mapping** node to **-235**.

Rotating the hdr image to match the position of the Lamp

6. Now add a **Math** node (*Shift* + *A* | **Converter** | **Math**) and a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**); connect the **Color** output of the **Environment Texture** node to the first **Value** input socket of the **Math** node, set the **Operation** of this latter node to **Multiply** and the second **Value** to **10.000**.
7. Connect the output of the **Multiply-Math** node to the **Color1** input socket of the **MixRGB** node and set the **Color2** to pure white; connect the Color output of the **MixRGB** node to the **Strength** input socket of the **Background** node:

Adding nodes to the World

8. Press *Shift* + *D* to duplicate both the **Math** and the **MixRGB** nodes: paste the duplicated **MixRGB** node between the first **MixRGB** and the **Background** nodes; set the **Operation** of the duplicated **Math** node to **Add**.
9. Add a **Light Path** node (*Shift* + *A* | **Input** | **Light Path**); connect its **Is Camera Ray** output to the first **Value** input socket of the duplicated **Add-Math** node and the **Is Glossy Ray** output to the second **Value** input socket; connect the **Value** output of the **Add-Math** node to the **Fac** input socket of the second **MixRGB** node and enable the **Clamp** item:



The completed IBL World setup for the Cycles render engine

10. Go to the **Settings** subpanel and enable the **Multiple Importance** item, then click on the *World datablock* to change the name in **World_Cycles**.
11. Go to the **Render** window and in the **Film** subpanel enable the **Transparent** item.

Renaming the World and some more settings in the main window

12. Save the file.

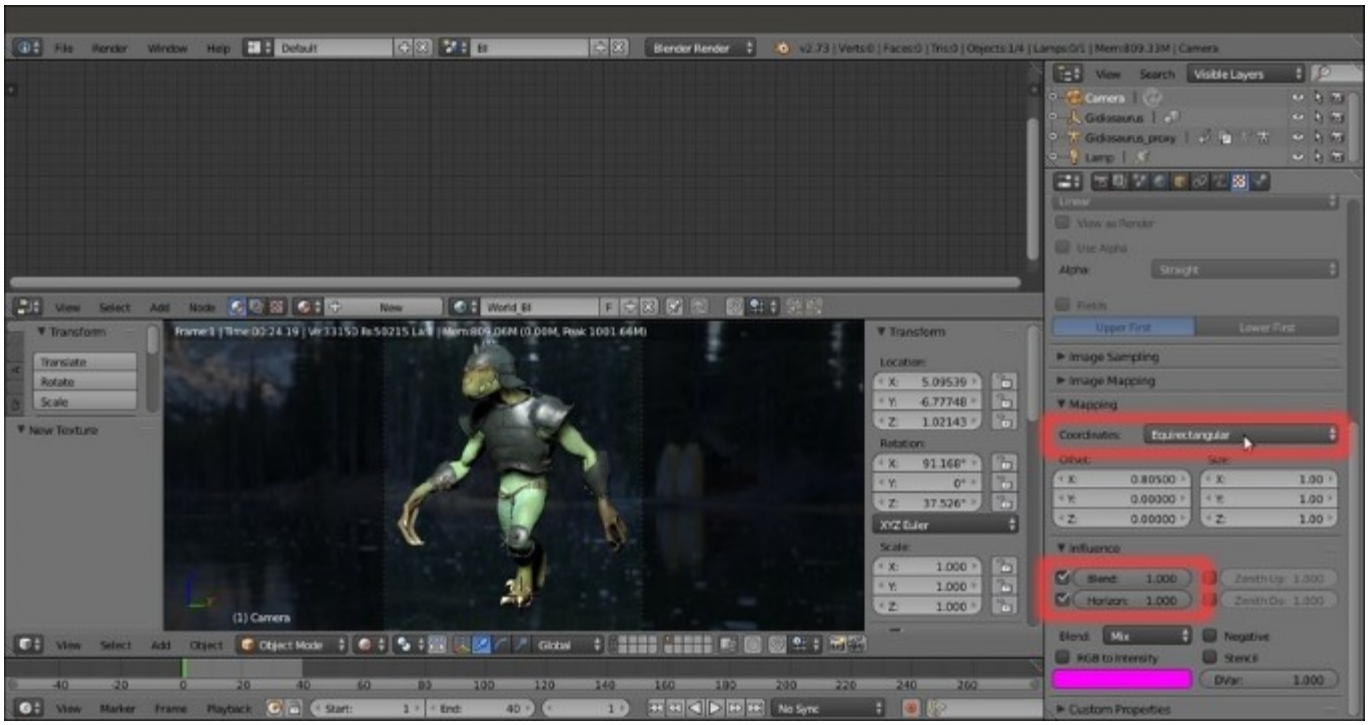
Image based lighting in Blender Internal

Now let's see the same thing in **Blender Internal**:

1. Click on the *Scene datablock* button in the top main header to switch from **Cycles** to **BI**.
2. In the **World** window to the right, click on the **2** icon button to the right side of the **World** name datablock to make it single user, then rename it **World_BI**.
3. Go directly to the **Texture** window: click on the **New** button, then click on the double arrows to the side of the image datablock to select the `L Ice_Lake_Refl.hdr` item from the pop-up menu:

Selecting the hdr image in the Blender Internal World

4. Rename the *ID name datablock* to **Ice_Lake_Refl**, then go down to the **Mapping** subpanel and click on the **Coordinates** slot to select the **Equirectangular** item; set the **Offset** to **X** = **0.80500** and then go further down to the **Influence** panel and enable the **Horizon** item.



First BI World settings

5. Back in the **World** window, in the **World** subpanel enable the **Real Sky** item.
6. Enable the **Environment Lighting** subpanel and click on the **Environment Color** button to select the **Sky Texture** item.
7. In the **Gather** subpanel, enable the **Approximate** method, the **Pixel Cache** item, the **Falloff** item and set the **Strength** value of this latter item to **0.900**.

More BI World settings

8. Go to the **Render** window and in the **Shading** subpanel click on the **Alpha Mode** slot to switch from the **Sky** to the **Transparent** item:

Enabling the transparent background for the rendering

9. Save the file.

How it works…

In **Cycles**: at steps 6 and 7 we added nodes to increase the source light intensity of the **hdr** image; because this also increased the contrast of the image, at steps 8 and 9 we made it less contrasted again but kept the same light intensity, thanks to the **Light Path** node. The light rays shoot from the **Camera** position and directly hit a surface (**Is Camera Ray**) or any glossy surface (**Is Glossy Ray**) and have value = **1.000**, hence corresponding to the **Color2** socket of the second **MixRGB** node, therefore giving a pure white (**1.000**) value to the **Background** node's **Strength**; any other ray (transmitted, shadows, reflected, transparent, and so on) has the high contrast **Strength** values we established at steps 6 and 7.

We used the **Mapping** node for the sole reason of matching (visually and thanks to the **World Background** item enabled in the **Display** subpanel under the *N* side **Properties** panel) the source light direction of the image with the position of the **Lamp** in the 3D scene: that's why we rotated the **hdr** image to negative **235** degrees on the *z* (vertical) axis.

In **Blender Internal**: we can't rotate the image, so instead we offset it on the *x* axis to (almost perfectly) match the position it has in **Cycles**.

The **Approximate** gathering method is the one developed during the production of the short open movie *Big Buck Bunny* (https://peach.blender.org/) to have faster rendering and absence of noise in **Ambient Occlusion**, inevitable with the default **Raytrace** method (that still remains the more accurate, by the way).

Note that, in both the render engines, we didn't load a brand new `Ice_Lake_Ref.hdr` image from the `textures` folder, but we instead used the linked one coming from the materials of the character, as indicated by the `L` in front of the name and by the name itself and all the settings grayed in the image datablock subpanel.

See also

The free **sIBL** addon currently, only works with **Cycles** materials but it can read the `.ibl` file provided with the free **hdr** images at the **sIBL Archive** (link provided further) and therefore, in one click, it can create the complete nodes setup to provide image based lighting in Blender.

- The official documentation about the addon (http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Import-Export/sIBL_GUI)
- An updated and bug-fixed version of the addon (https://raw.github.com/varkenvarken/blenderaddons/master/sibl.py)
- The sIBL archive (http://www.hdrlabs.com/sibl/archive.html)
- Official documentation about the **World** in Blender:
    - http://www.blender.org/manual/render/cycles/world.html
    - http://www.blender.org/manual/render/blender_render/world/index.html

Setting a three-point lighting rig in Blender Internal

Thanks to the global illumination, a path-tracer like **Cycles** doesn't necessarily need big lighting setups; in fact, in the recipes we made, we only used one single **Spot** lamp in addition to the **IBL** and the results have been quite good anyway.

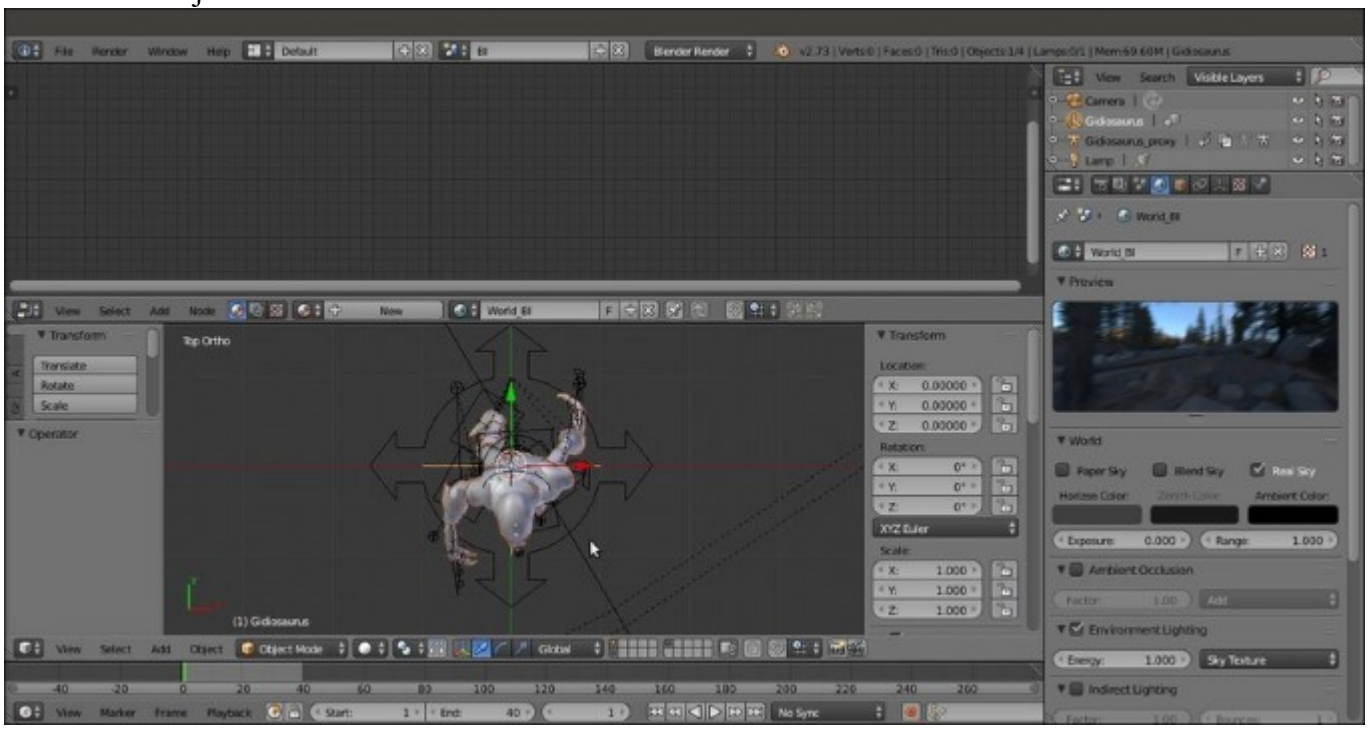
In **Blender Internal**, instead, a minimum arrangement of lamps must be done to obtain satisfying results, even with the aid of the **World** settings we have previously seen.

In this recipe, we are therefore going to see a classic *movie* three-point lighting rig, an industry standard. The effect of the main **key light** is enhanced by the other two lamps: the **fill light**, to brighten (and color) the shadow areas on the subject, and the **backlight**, to create a light rim on the subject edges thus making it stand out against the background.

Getting ready

Start Blender and load the previously saved `Gidiosaurus_IBL.blend` file; if necessary, switch to the **Blender Render** engine by the *Engine to use for rendering* button in the top main header.

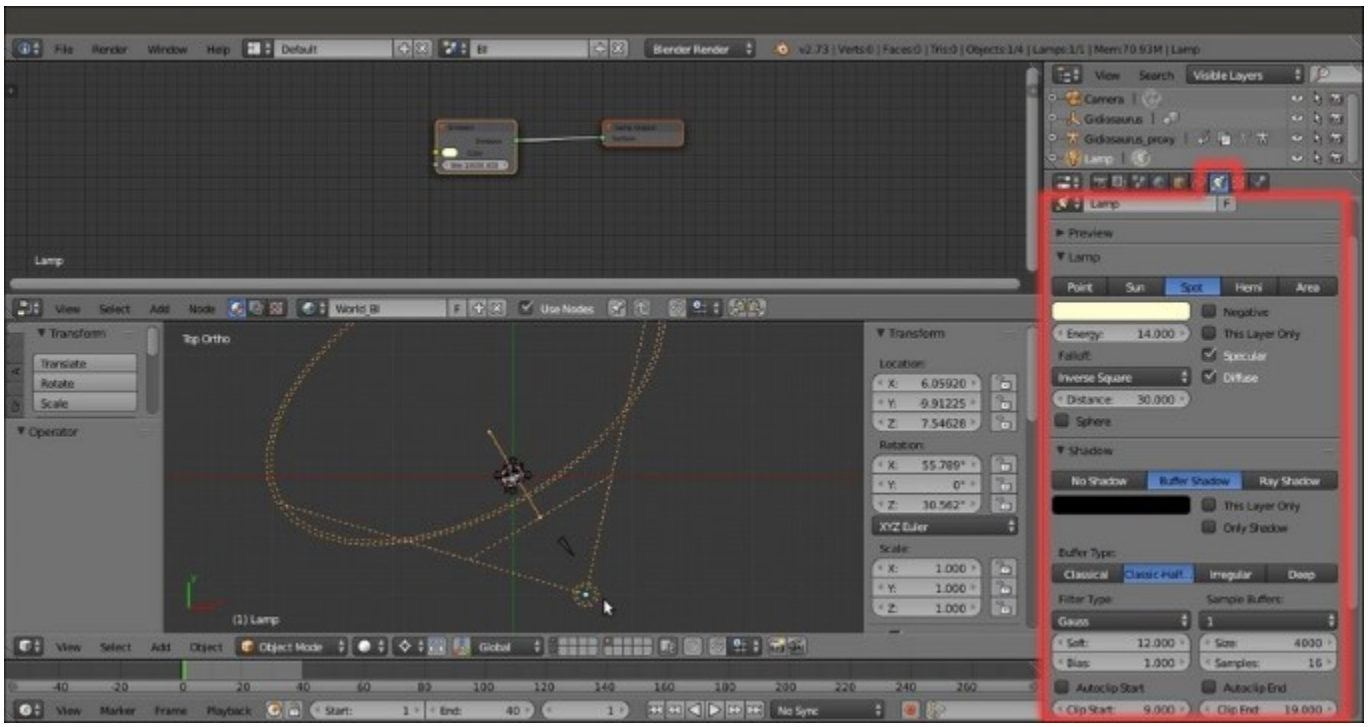
1. Put the mouse pointer inside the **Camera** view and press the numpad *7* key to go in **Top** view, then press numpad *5* to switch from **Perspective** to **Ortho** view.
2. Press *Shift* + *C* to put the **3D Cursor** at the center of the scene and then go to the **Outliner** to select the **Gidiosaurus** item: press the numpad period (.) key to center and zoom the view on the selected object:



Centering the top view on the character

3. Press *Ctrl* + Spacebar to disable the widget and scroll the mouse wheel to zoom backward and show the **Spot** lamp, then press the dot (.) key to switch the **Pivot Point** from **Median Point** (or whatever else) to **3D Cursor**.
4. Select the **Lamp**, then go to the **Object Data** window.
5. Remember to go to the **Scene** window and disable the **Simplify** subpanel!
6. Save the file as `Gidiosaurus_lighting.blend`.

Don't take into consideration the **Node Editor** window at the top showing the **Lamp** nodes under **Cycles**; the settings to look for are those inside the main **Properties** panel to the right:

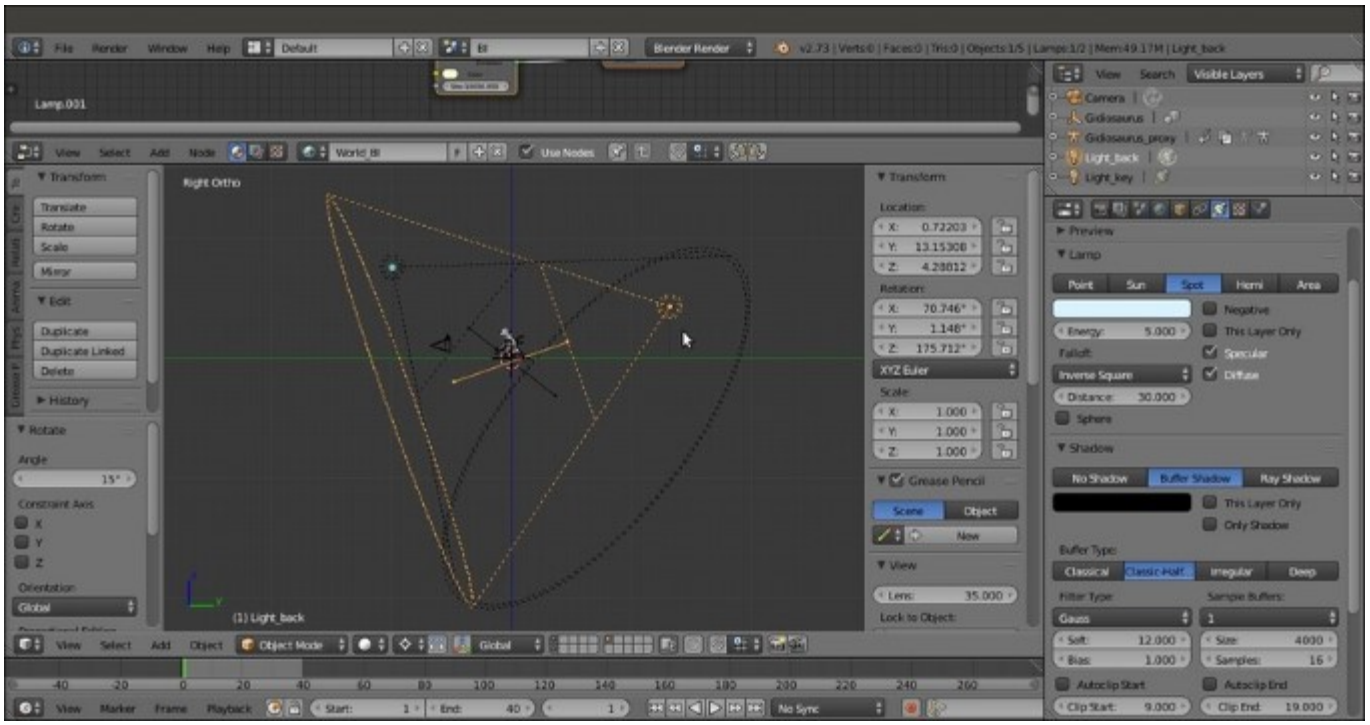


These are the settings you are looking for...

How to do it…

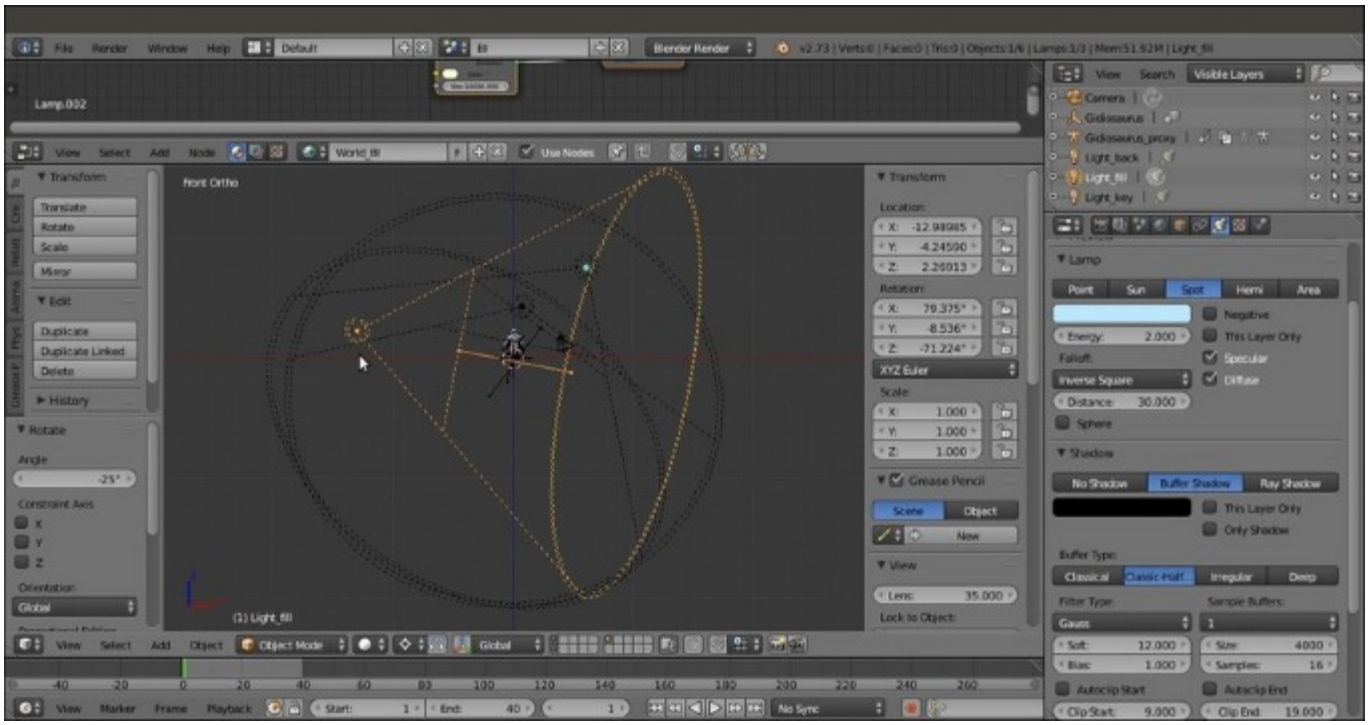
Let's go with the settings of the lights:

1. In the **Outliner**, rename the **Lamp** item as **Light_key**.
2. Press *Shift* + *D* to duplicate the **Key_light** lamp, press *R* and, while still in **Top** view, rotate the duplicated lamp approximately **-145** degrees, then go in **Side** view (numpad *3* key) and rotate it around **15** degrees: in the **Outliner**, rename it as **Light_back**.
3. In the **Object Data** window, set the color to a light blue = **R 0.700**, **G 0.900**, **B 1.000** and the **Energy** to **5.000**:



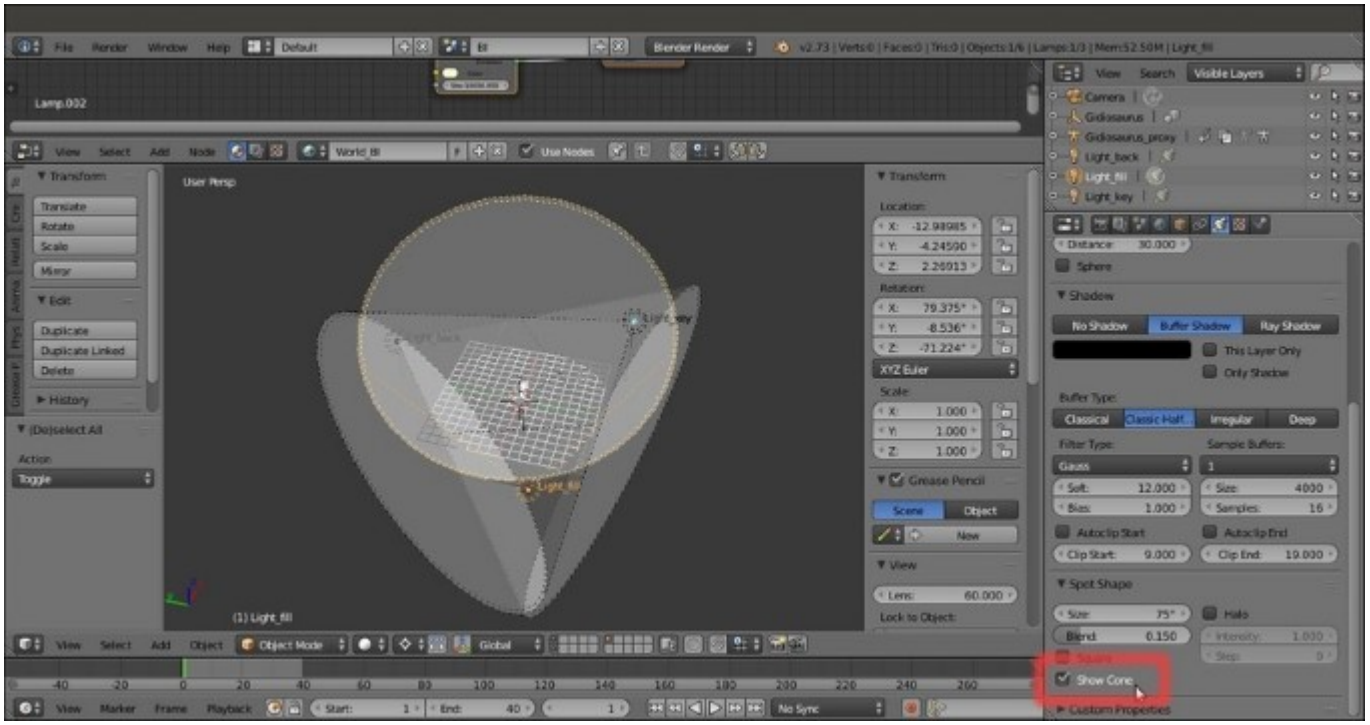
Positioning the Light_back lamp

4. Go back in **Top** view (numpad *7* key) and re-select the **Light_key** lamp, press *Shift* + *D* and rotate the duplicate by **100** degrees; in the **Outliner**, rename it as **Light_fill**. Go in **Front** view (numpad *1* key) and rotate it around **-25** degrees.
5. In the **Object Data** window, set the color to a lighter blue = **R 0.500**, **G 0.800**, **B 1.000** and the **Energy** to **2.000**:



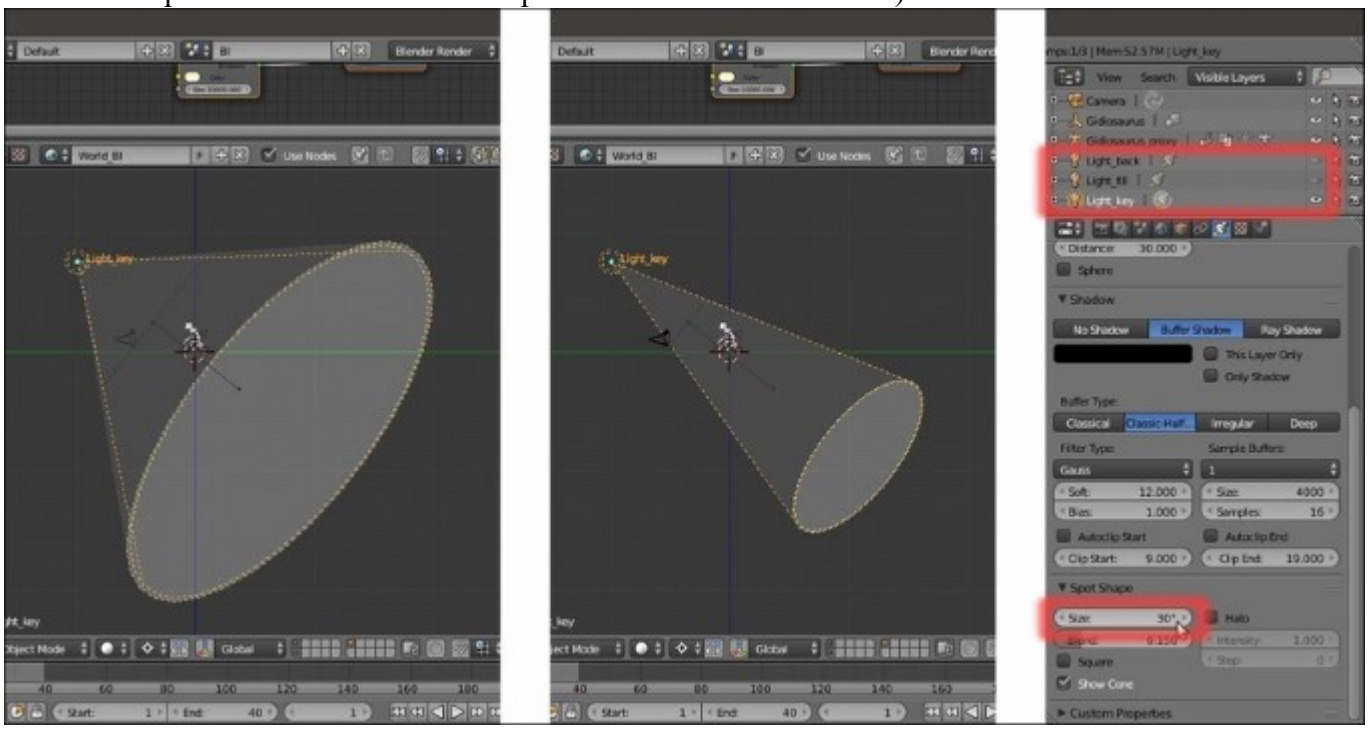
Positioning the Light_fill lamp

6. Go to the **Object** window and in the **Display** subpanel enable the **Name** item for the three lamps; then, back to the **Object Data** window and in the **Spot Shape** subpanel, enable the **Show Cone** item for each one:



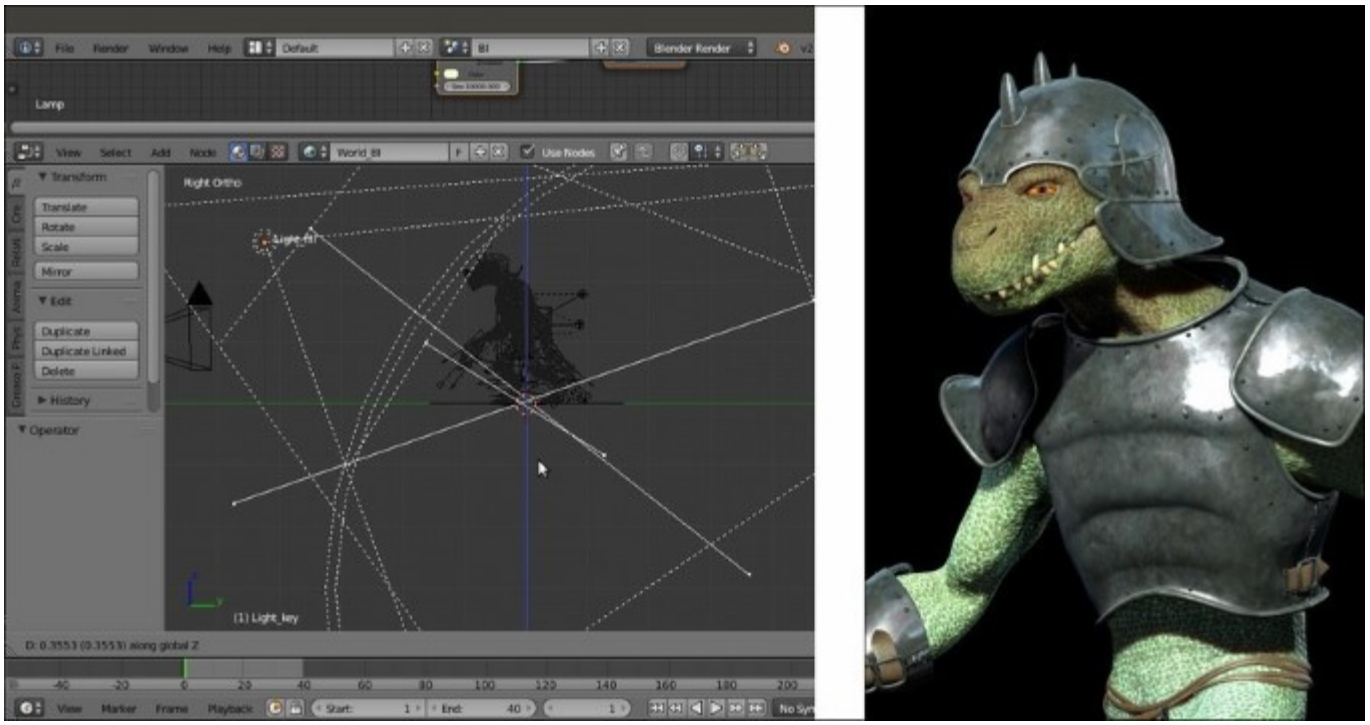
The three spot lamps showing their cones of influence

7. In the **Outliner**, disable the 3D viewport visibility of the **Light_back** and **Light_fill** lamps by clicking on the respective eye icon, then go in **Side Ortho** view and select the **Light_key** lamp.
8. Go to the **Spot Shape** subpanel again and lower the **Size** value from the default **75°** to **30°** (or the smallest possible value that still comprehends the whole character):



Lowering the spot lamp size value

9. Repeat steps 7 and 8 for the other two lamps as well, then *Shift*-select all the three lamps and move them upward (on the *z* axis) a bit, just to better center the light cones' centers on the position of the feet of the character.
10. Save the file.



The final result of the three-point lighting rig

How it works…

A classic three-point lighting rig can in some way compensate for the lack of real global illumination in **Blender Internal**, although to obtain really good results, three lamps are usually not enough; in any case, the lighting rig of this recipe can be used as a base for even more complex setups.

When using more than one lamp in **Blender Internal**, we should always be sure that the shadows are enabled for all of them, unless we want particular effects; in fact, a back lamp with disabled shadows can easily *shine* through the model and also illuminate parts that shouldn't be in light, giving unrealistic results.

To calculate the buffered shadows, **Spot** lamps take into consideration everything inside their cone from the **Clip Start** to the **Clip End** values; this is why we lowered the **Size** values of the cones as much as possible.

One other crucial factor that can slow the calculation and the rendering times is, obviously, the size of these buffers, which we set to **4000** for each one of the three lamps; quite big, but because we set the cones that large enough to just comprehend the shape of their target object. This means we could use big shadow buffers, to obtain more details in the shadows if needed.

We do all of this, even though the **Gidiosaurus** was the only object to be rendered in the scene.

See also

- http://www.blender.org/manual/render/blender_render/lighting/index.html

Rendering an OpenGL playblast of the animation

Playblast is a term used by a famous commercial package to indicate the preview of the animation in true speed; although I've heard only very few people using it in relation to Blender, I thought it might be a good way to indicate the fast OpenGL preview rendering obtained for checking the animated action.

Getting ready

Start Blender and load the `Gidiosaurus_lighting.blend` file.

1. In the **Outliner**, select the **Light_key** lamp item and go to the **Object Data** window, under the **Spot Shape** subpanel, to disable the **Show Cone** item.
2. Repeat the procedure for the **Light_back** and **Light_fill** lamps, then disable their visibility in the 3D viewport by clicking on the respective eye icon.
3. Disable the visibility in the viewport for the **Gidiosaurus_proxy** item (the linked and proxified rig) also and/or disable the **11th** scene layer.
4. Save the file as `Gidiosaurus_playblast.blend`.

How to do it…

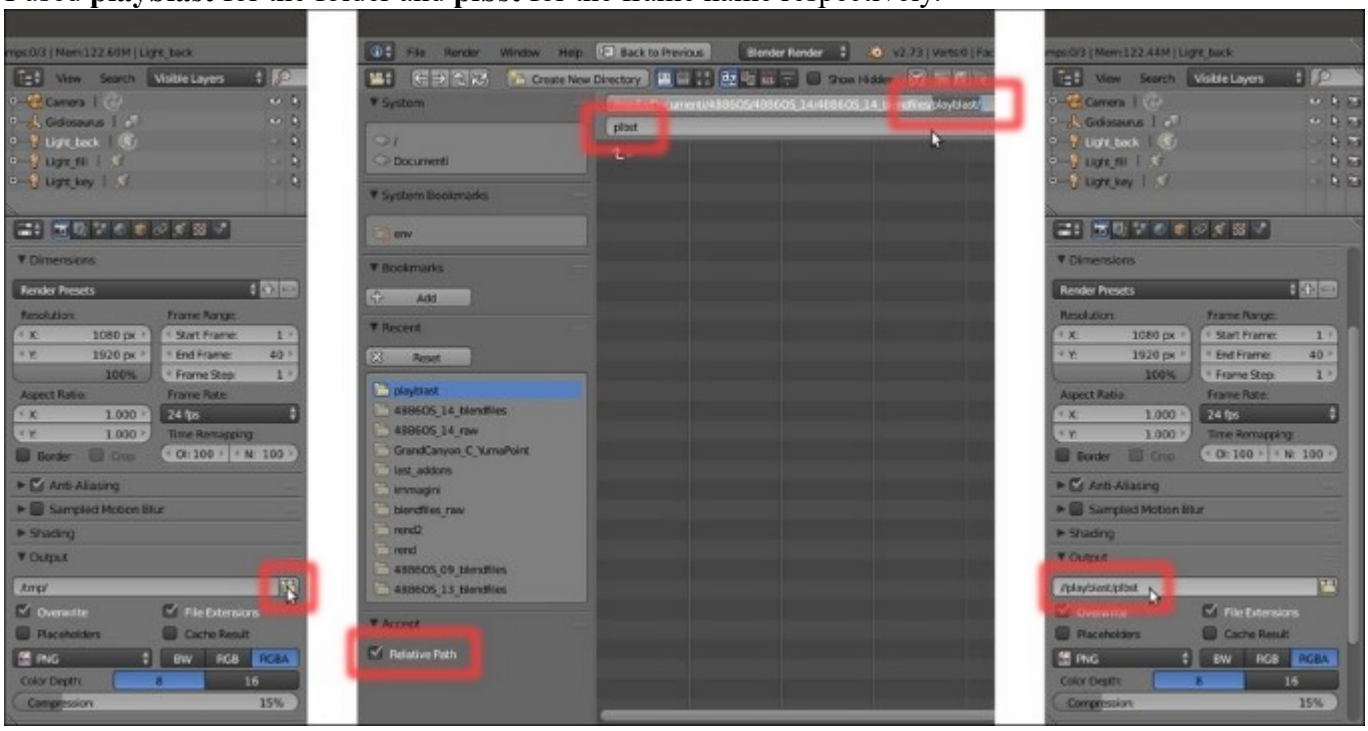
Here are the steps to begin with the OpenGL rendering:

1. Put the mouse pointer inside the 3D viewport and press the numpad *0* key to go in **Camera** view; press the *Z* key to go in **Solid** viewport shading mode, then scroll the mouse wheel to zoom the **Camera** view inside the window:

The Camera view in Solid viewport shading mode

2. Go to the **Render** window and to the **Dimensions** subpanel; check for the **X** and **Y** sizes of the rendering under **Resolution**, specified in pixels, and move the *Percentage scale for render resolution* slider, usually set to **50%**, to **100%**.
3. Go down to the **Output** subpanel and click on the folder icon button to the end of the path slot; browse to the location you want to save your rendering, then type in the first line of the path to the folder you want to create at that location, followed by the slash (/) and press *Enter*.
4. A pop-up will ask you to confirm the creation of the new directory; confirm and then type a generic frame name in the second line, go to the left side vertical bar to be sure that the bottom **Relative Path** item is enabled and finally click on the **Accept** button at the top left of the screen.

   I used **playblast** for the folder and **plbst** for the frame name respectively.



The new directory and the rendered frames name

5. Save the file, then go to the **Camera** view toolbar and click on the last *ciak* icon button to the left to start the OpenGL playblast:

The two buttons to start the OpenGL rendering (for a still to the left, for the animation to the right)

How it works…

In our example, the OpenGL playblast rendered single `.png` images with an alpha background because, as you can see in the **Render** window visible in the previous screenshot, these are the settings of the **Output** subpanel. Be aware that the resolution, the format and the path where the playblast frames are saved, always depend on the settings in the **Render** window, the same settings that will be used for the final real rendering (but of course the resolution of the playblast can be easily and temporarily be made smaller with the slider of the percentage scale).

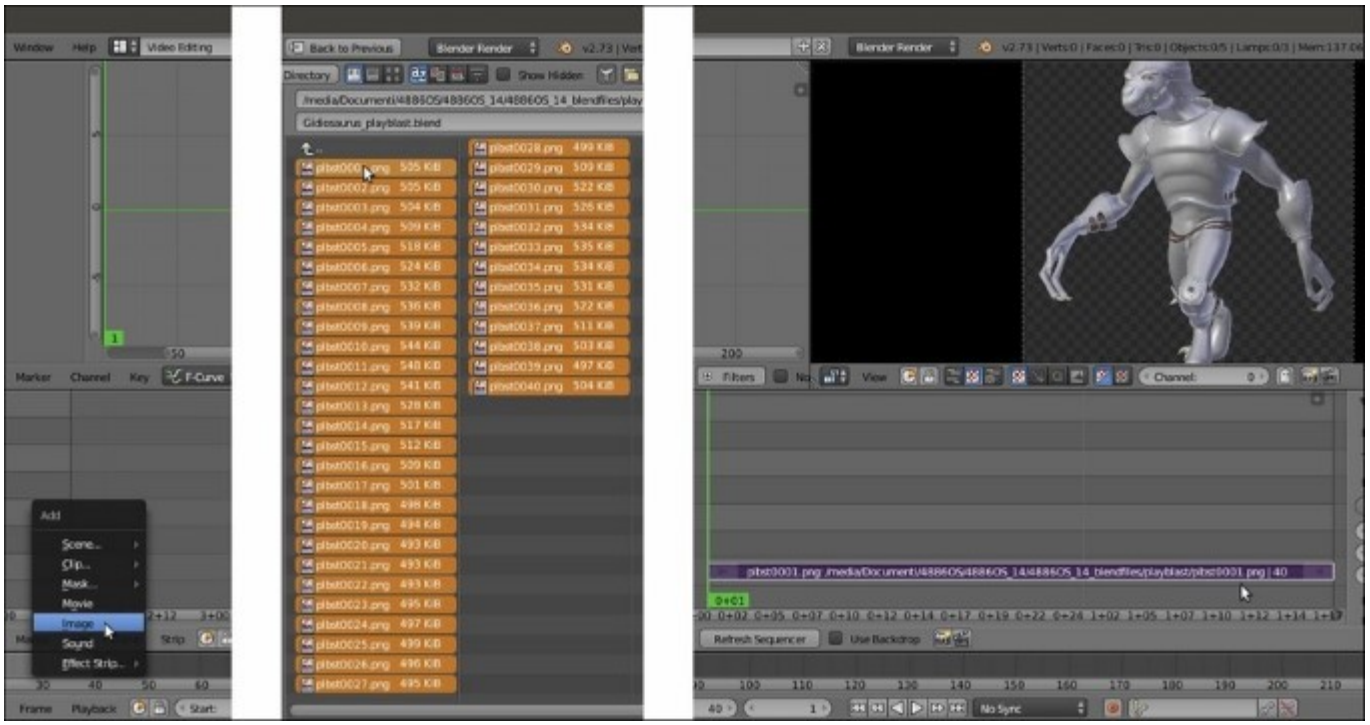
There's more…

Once we have rendered all the frames, we can use an external player to see them in sequence (in **Ubuntu**, I use the free player **DJV Imaging**, http://djv.sourceforge.net) or, just quickly build a movie through the **Blender Sequencer**:

1. Go to the *Screen datablock* button on the top main header and click it to switch to the **Video Editing** screen:

Switching to the Video Editing screen layout

2. Put the mouse pointer in the **Video Sequence Editor** window at the bottom and press *Shift* + *A*; from the pop-up menu select the **Image** item (*Add an image or image sequence to the sequencer*), then browse to the `playblast` folder location, click on it and once inside, press the *A* key to select all the contained frames, then press *Enter* to confirm. The frames are added to the **Video Sequence Editor** window as a single strip and the current frame appears in the preview window:



Loading the rendered frames in the Video Sequence Editor

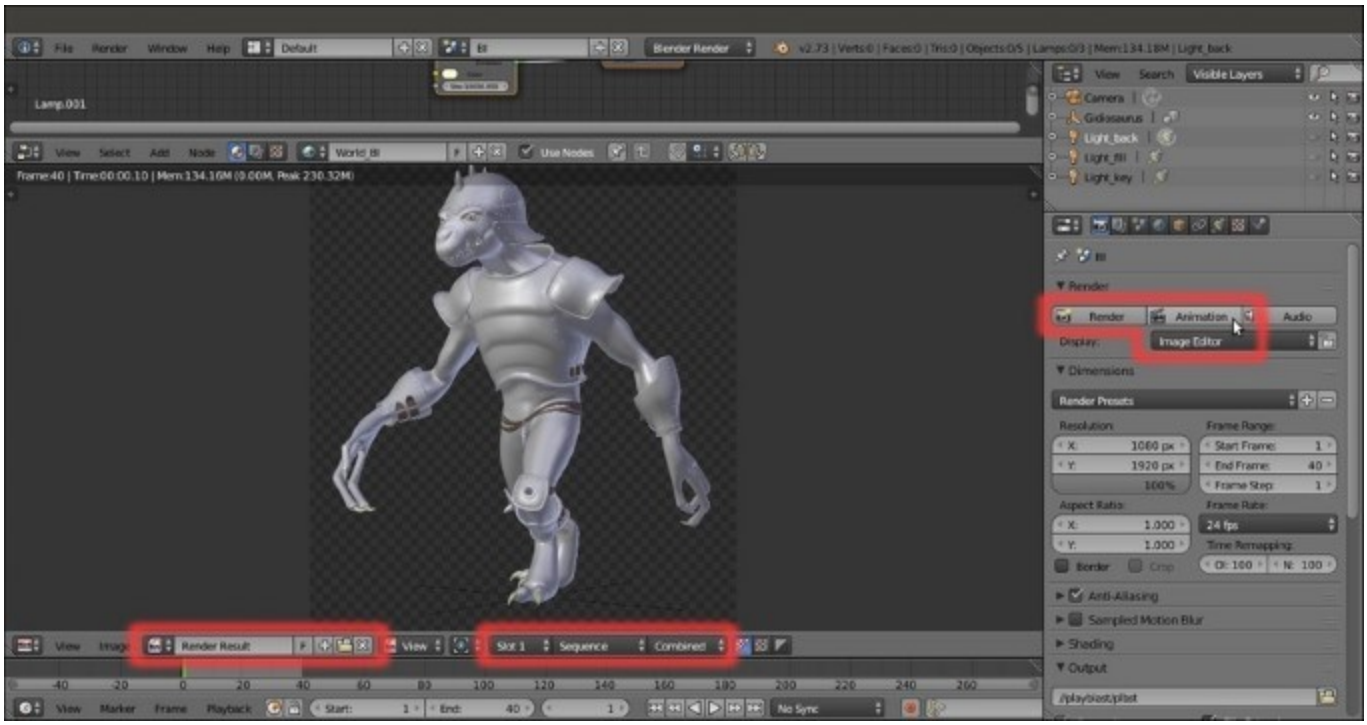
3. Go back to the **Default** screen and to the **Render** window under the main **Properties** panel. In the **Output** subpanel, where you can change the path to save the movie in a different location (or also leave it as it is), click on the **File Format** button to select a **Movie** format, for example, **AVI JPEG**. Choose **BW** or **RGB** and the **Quality** compression ratio (but the default **90%** is usually OK); then go to the **Post Processing** subpanel and ensure that the **Sequencer** item is enabled:

The Output and the Post Processing subpanels inside the Render window

4. Go to the top of the **Render** window and click on the **Animation** button; remember that Blender uses two different buttons to start the rendering of a still image or of an animation, both for the final rendering and for the 3D viewport toolbar OpenGL preview we have seen in the *How to do it…* section.

The rendering starts and the **Sequencer** processes all the `.png` images outputted by the playblast, transforming them into a single compressed `.avi` movie then saved in the same directory as the frames.

The process is visible in the **UV/Image Editor** window that replaced the **Camera** view, indicated in the toolbar by the **Render Result** label on the image datablock to the left (because the **Image Editor** item is the one selected in the **Display** slot under the **Render** subpanel) and by the **Sequence** label visible in the **Layer** slot to the right:



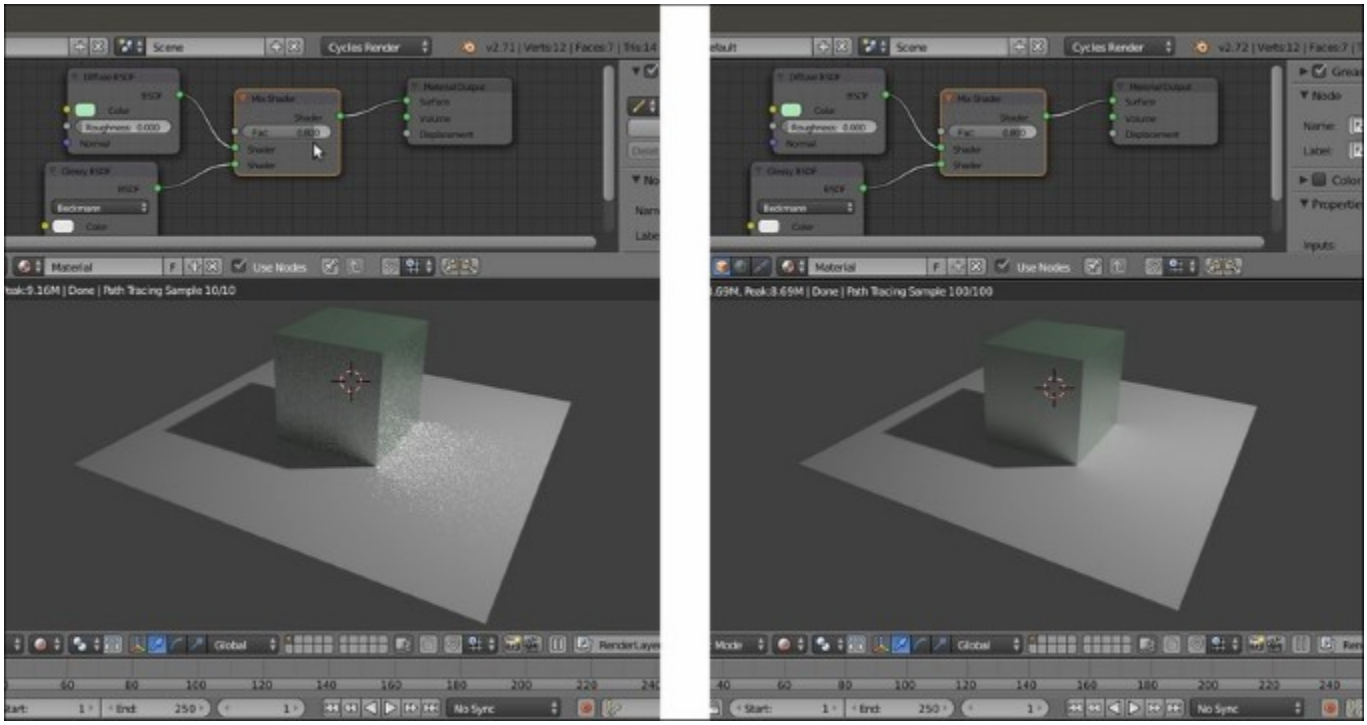
The Render Result window

See also

- http://www.blender.org/manual/render/display.html
- http://www.blender.org/manual/render/output/video.html
- http://www.blender.org/manual/render/output.html

Obtaining a noise-free and faster rendering in Cycles

The **Cycles Render** engine can be very slow compared to **Blender Internal**; by the way, some of the rendering settings can be tweaked to make it work faster; the goal here is to avoid fireflies and noise, usually due to low samples and a light source that is too bright.



Rendered previews of an example scene, showing a cube on a plane with and without noise and fireflies

Getting ready

Start Blender and load the `Gidiosaurus_playblast.blend` file.

1. Click on the *Scene datablock* button in the top main header to switch from **BI** to **Cycles**.
2. Go to the **Outliner** and enable the visibility of the **Light_key** lamp in the viewport by clicking on the grayed eye icon.
3. Put the mouse pointer inside the **Camera** view and press *Shift* + *B* to draw a box around the character's head, then zoom to it.
4. Save the file as `Gidiosaurus_render.blend` and press *Shift* + *Z* to start the **Rendered** preview.

How to do it…

If you have a capable graphic card supporting **GPU** (go to the last *See also* section for the link to a list of supported **GPU** graphic cards for **Cycles**), the next thing to do is:

1. Call the **User Preferences** panel (*Ctrl* + *Alt* +*U*) and go to the **System** tab; on the bottom left there is the **Compute Device** item and the slot you can click on to select the device for the rendering: if you have a graphic card that supports this feature, set the GPU instead of the default CPU, and to make this permanent, click on the **Save User Settings** button, or press *Ctrl* + *U*, and close the panel.
2. Go to the **Render** window under the main **Properties** panel and, in the top **Render** subpanel it is now possible to select the **GPU** item as a rendering device, but only if your graphic card supports **CUDA**.

   This will boost your rendering speed several times, making it possible to significantly increase the rendering samples in the **Sampling** subpanel to reduce or even avoid the noise and keep good rendering times. Using the **GPU**, it's also possible to increase the size of the **X** and **Y Tiles** in the **Performance** subpanel (two or three times the default size is **64**).

   But, not everyone has a **GPU** graphic card yet, and there are also cases where you have to mandatorily use the **CPU** instead (for example, for very big scenes with a lot of geometry that doesn't fit inside the somewhat limited **RAM** of a graphic card).

   In such cases, there are things you can do to try to obtain faster and better quality render results:
3. Select the **Light_key** lamp and in the **Node Editor** window add a **Light_Falloff** node (*Shift* + *A* | **Color** | **Light Falloff**); connect its **Linear** output to the **Strength** input socket of the **Emission** node, set the **Strength** to **1000.000** and the **Smooth** value to **1.000**.
4. Click on the color box of the **Emission** node and change the color to **R 0.800**, **G 0.800**, **B 0.650**.
5. Go to the **Render** window and in the **Sampling** subpanel set the **Samples** to **200** or a higher value both for **Render** and **Preview**, to reduce the noise as much as possible.
6. Set the **Clamp Direct** and the **Clamp Indirect** values to **3.00** or **4.00** or even higher (they are set to **0.00** by default); when possible, it is better to leave the **Clamp Direct** item at **0.00** or use values higher then **2.00**, otherwise you could get weird effects in the texturing.
7. In the **Light Path** subpanel, disable both the **Reflective Caustics** and the **Refractive Caustics** items (unless you really need to have caustics in your render) and set the **Filter Glossy** value to **4.00** – **6.00**.
8. In some cases, it won't be possible to totally eliminate the noise or the fireflies; but, because we are going to render an animation, that is several frames in sequence, at least we can make the noise less noticeable and more *natural* looking: go back to the **Sampling** subpanel and click on the **Seed** slot to type `#frame`. This creates an automated driver that takes the seed value from the current frame number, in order to have different noise at every frame.

The Light Falloff node for the Lamps, the Seed driver for the noise, the Caustics items and the Filter Glossy value

See also

- http://www.blender.org/manual/render/cycles/reducing_noise.html
- http://www.blender.org/manual/render/cycles/settings/index.html
- http://www.blender.org/manual/render/cycles/gpu_rendering.html
- http://www.blender.org/manual/render/workflows/animations.html
- http://www.blender.org/manual/render/blender_render/performance.html
- A list of supported GPU graphic cards for Cycles can be found at https://developer.nvidia.com/cuda-gpus

Compositing the render layers

We have seen that the rendering in the **Cycles Render** engine is quite slow but of very good quality, while the scanline **Blender Render** engine is faster but with a lower quality.

Thanks to the Blender integrated **Compositor** and to the **render layers**, it is possible to mix separated and different passes of both the renderers, obtaining a compromise between quality and speed, for example, by over-imposing the **glossy pass** obtained in **Cycles** on the **diffuse pass** obtained in **BI**, and so on.

Getting ready

To mix different passes obtained from the two render engines, we must first apply some modification to the materials of the library file:

1. Start Blender and load the `Gidiosaurus_shaders_Blender_Internal.blend` file, which is the file we used as library for the proxified character and the walkcycle action.
2. In the **Outliner** select the **Gidiosaurus_lowres** item, be sure to be in the **BI** scene and go to the **Material** window; select the first material slot, that is the `Material_skin_U0V0` slot, and in the **Node Editor** window select the **SPEC** material node:

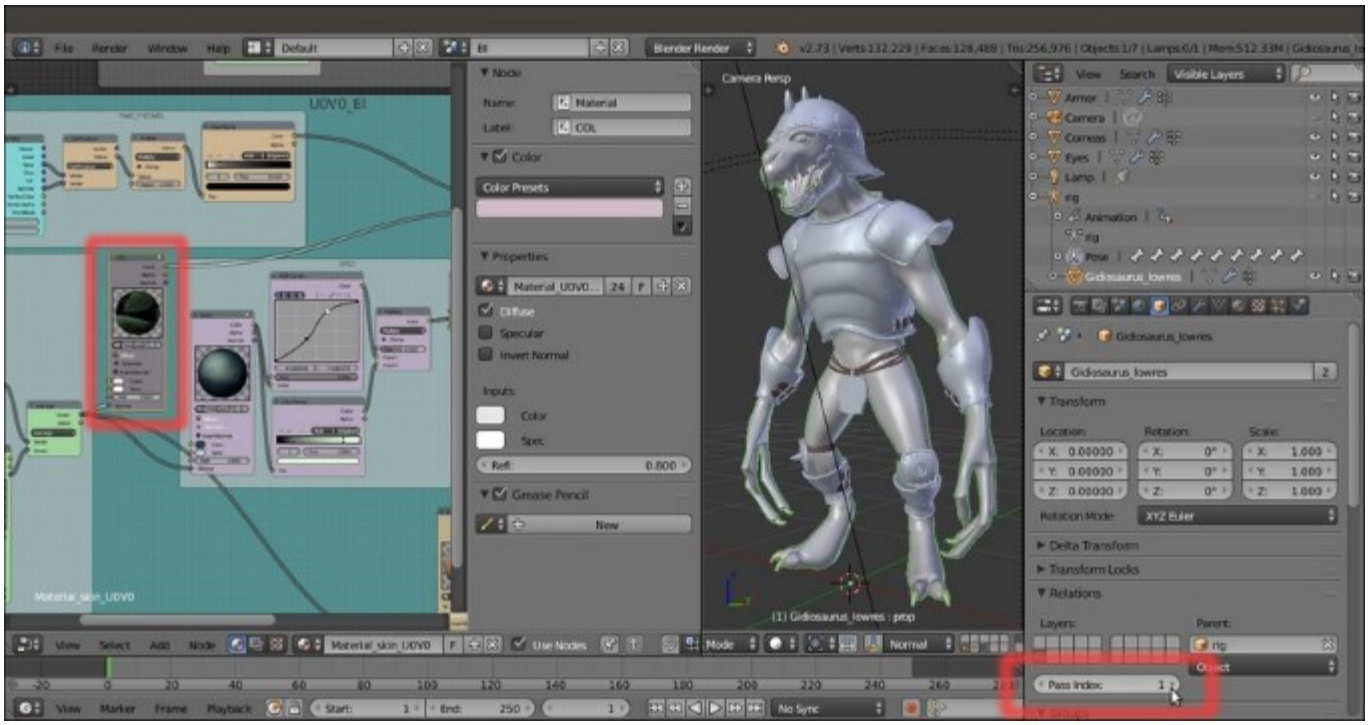
The SPEC node material for Blender Internal

3. Go to the **Texture** window and click on the *Enable/Disable each texture* checkbox at the right side of the **env_refl_skin** texture slot to disable it:

The disabled "env_refl_skin" texture slot

4. Go back to the **Node Editor** window and select the **COL** node (this seems to be important, because of a bug in the **Blender Internal** working method with linked material nodes and the **despgraph** that doesn't update the materials and, therefore, doesn't render the diffuse color correctly).
5. Select the second `Material_skin_U1V0` slot, go to the **Node Editor**, select the **SPEC** material node, disable the `env_refl_skin` texture slot, select the **COL** material node, then go to the third `Material_skin_U2V0` slot, and so on: repeat for all the materials of the **Gidiosaurus** and of the **Armor** objects (for the **Armor** disable the `env_refl_armor` slots on both the **SPEC1** and **SPEC2** material nodes; **Eyes** and **Corneas** have the real ray-tracing mirror enabled and don't need any textures disabled).

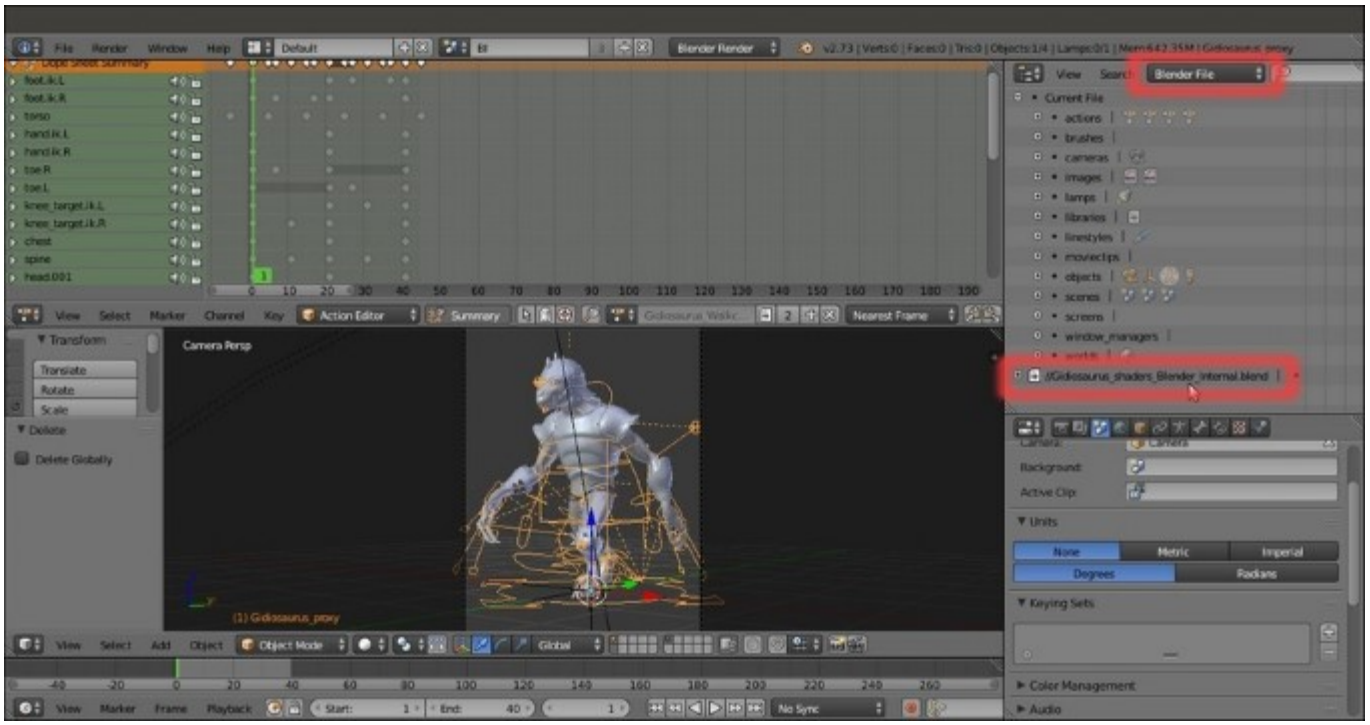
   Remember to leave the **COL** material nodes as the selected ones in the **Node Editor** window, or the diffuse color won't show in the rendering.
6. Go to the **Object** window and, in the **Relations** subpanel, assign a different **Pass Index** to the objects: assign **1** to the **Gidiosaurus_lowres** object, **2** to the **Armor** object, **3** to the **Eyes**, and **4** to the **Corneas**.



The Pass Index slot

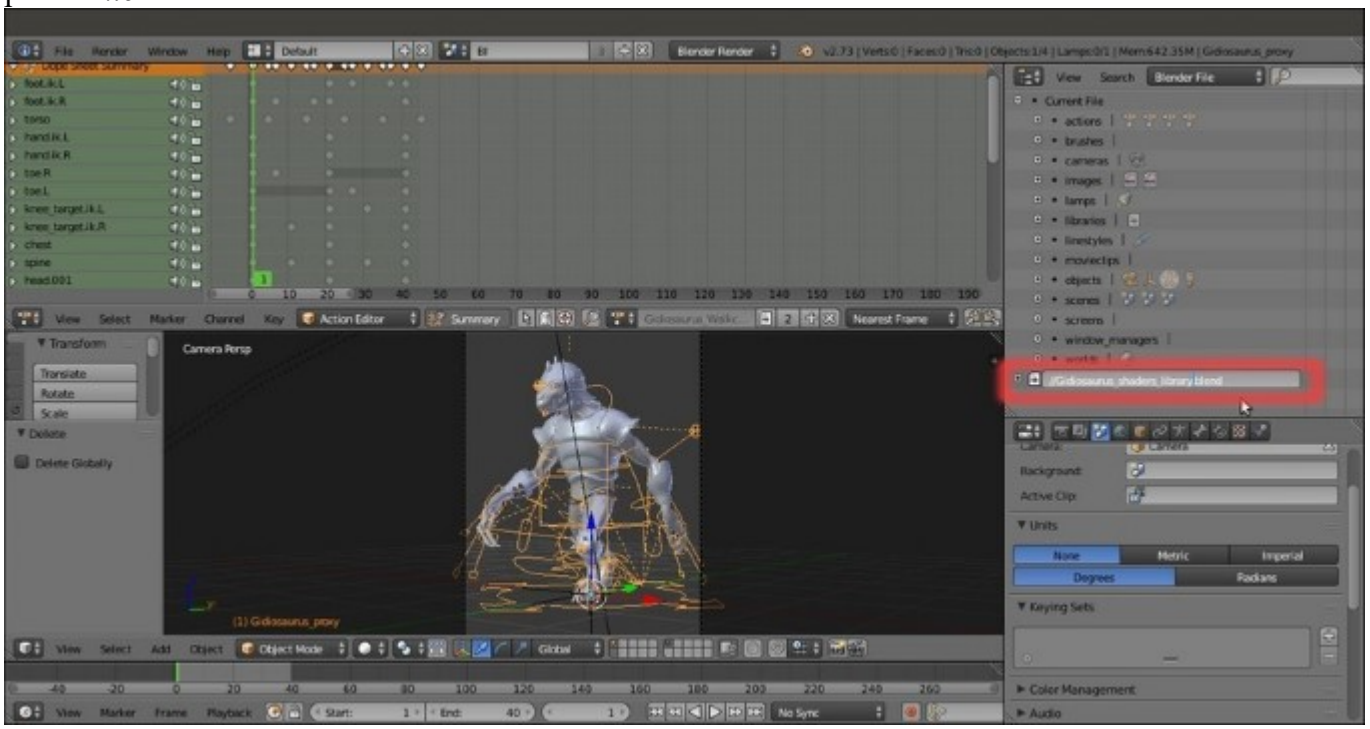
7. Save the file as `Gidiosaurus_shaders_library.blend`.

   Now, because up to this point we have used the `Gidiosaurus_shaders_Blender_Internal.blend` file as library source, we must now substitute the file to be rendered the path to the new library source:
8. Open the `Gidiosaurus_render.blend` file and go to the **Outliner** window: click on the *Type of information to display* button at the top to switch from **Visible Layers** to **Blender File**; expand the panel to find the **//Gidiosaurus_shaders_Blender_Internal.blend** item at the bottom:



The library path in the Outliner

9. Double left-click on the item and rename it as **//Gidiosaurus_shaders_library.blend**, then press *Enter* to confirm and save the file:



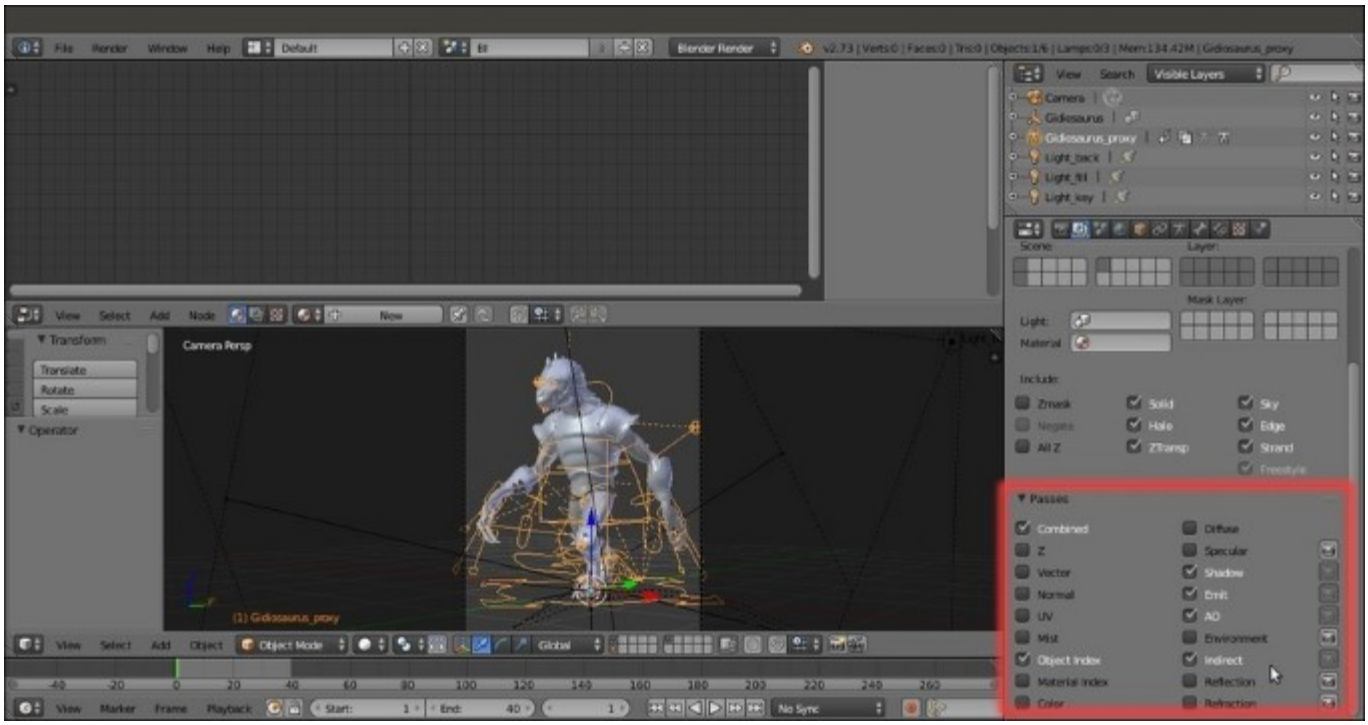
The modified library path

10. Press *Ctrl* + *O* | *Enter* to re-load the file: now all the assets should be linked from the new library file.
11. Save the file as `Gidiosaurus_compositing.blend`.

How to do it…

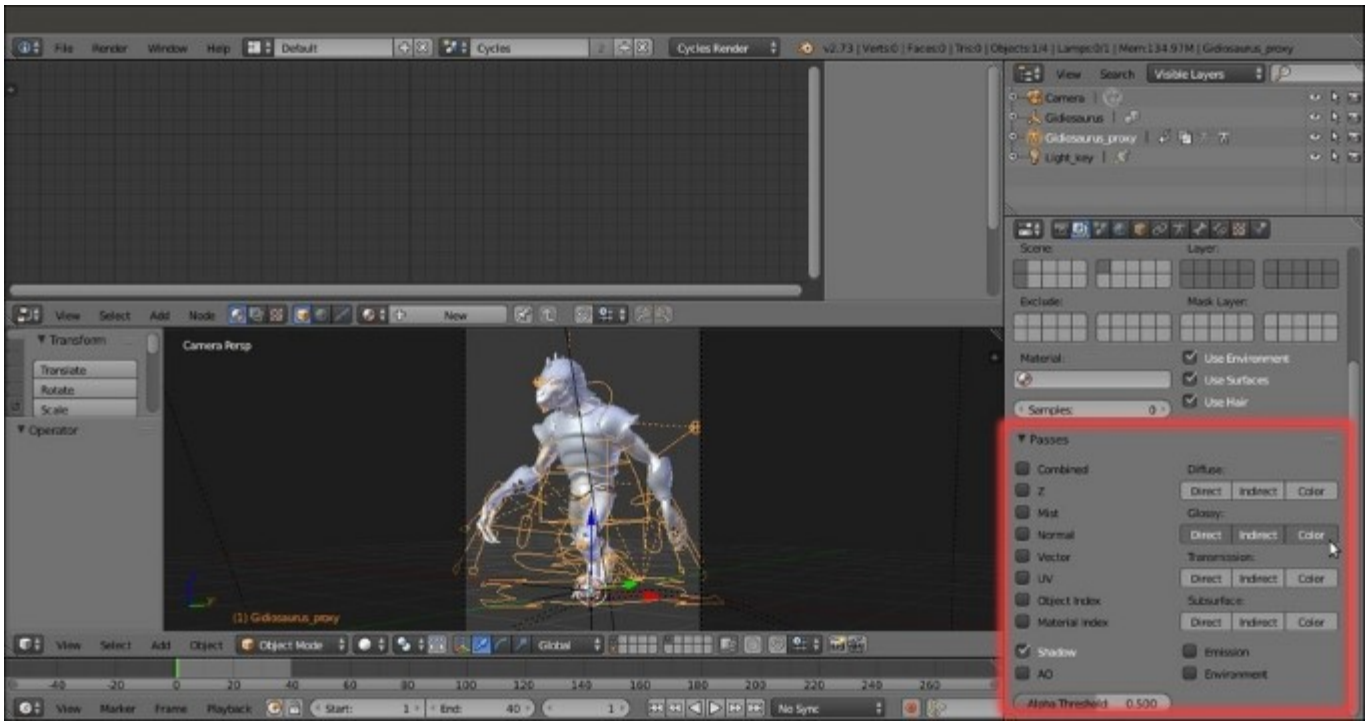
At this point we must prepare the passes for the two scenes that will be used later, as elements to be mixed through a third new *compositing* scene:

1. Click on the **+** icon button to the right side of the *Scene datablock* button in the main top header and, in the **New Scene** pop-up menu, select the **New** item: this creates a new **empty** scene; rename it **comp**.
2. Click on the *Screen datablock* button to the left and switch to the **Compositing** screen, then click again on the *Scene datablock* button and re-select the newly created **comp** scene.
3. Click again on the *Screen datablock* button and go back to the **Default** screen; go to the *Scene datablock* button and select either the **BI** or the **Cycles** scene.
4. From now on, it's enough to select the **Compositing** layout in the *Screen datablock* button to switch automatically to the **comp** scene, and the **Default** layout to go to the **BI** or the **Cycles** scene (depending on the last one selected).
5. Go to the **Default** screen and, if not already loaded, load the **BI** scene; in the main **Properties** panel, go to the **Render Layers** window (the second icon button from the left in the *Type of active data to display and edit* windows row).
6. Double click on the **RenderLayer** name in the first slot at the top of the subpanel to rename it as **BI**. Go down to the **Passes** subpanel, disable the **Z** pass item and enable **Object Index**; then go to the second column and enable the **Shadow** pass but then also click on the *Exclude shadow pass from combined* button to its extreme right side: do the same also for the **Emit**, the **AO** and the **Indirect** passes.



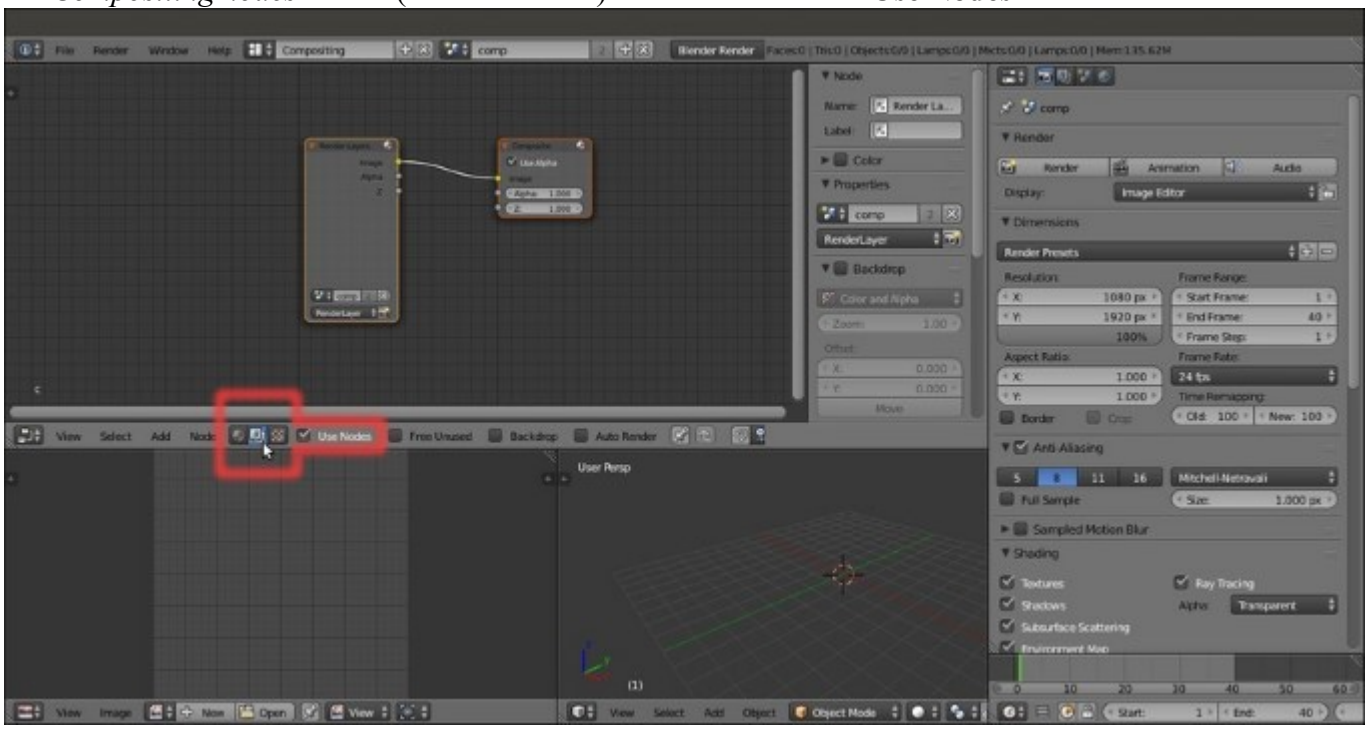
The Passes setting for the Blender Render scene

7. Click again on the *Scene datablock* button in the top main header and switch to the **Cycles** scene.
8. Double click on the **RenderLayer** name in the first slot at the top of the subpanel to rename it as **Cycles**, then disable the **Combined** and the **Z** passes, leave the already enabled **Shadow** pass as it is and enable also the **Glossy Direct**, **Indirect** and **Color** passes:



The Passes setting for the Cycles Render scene

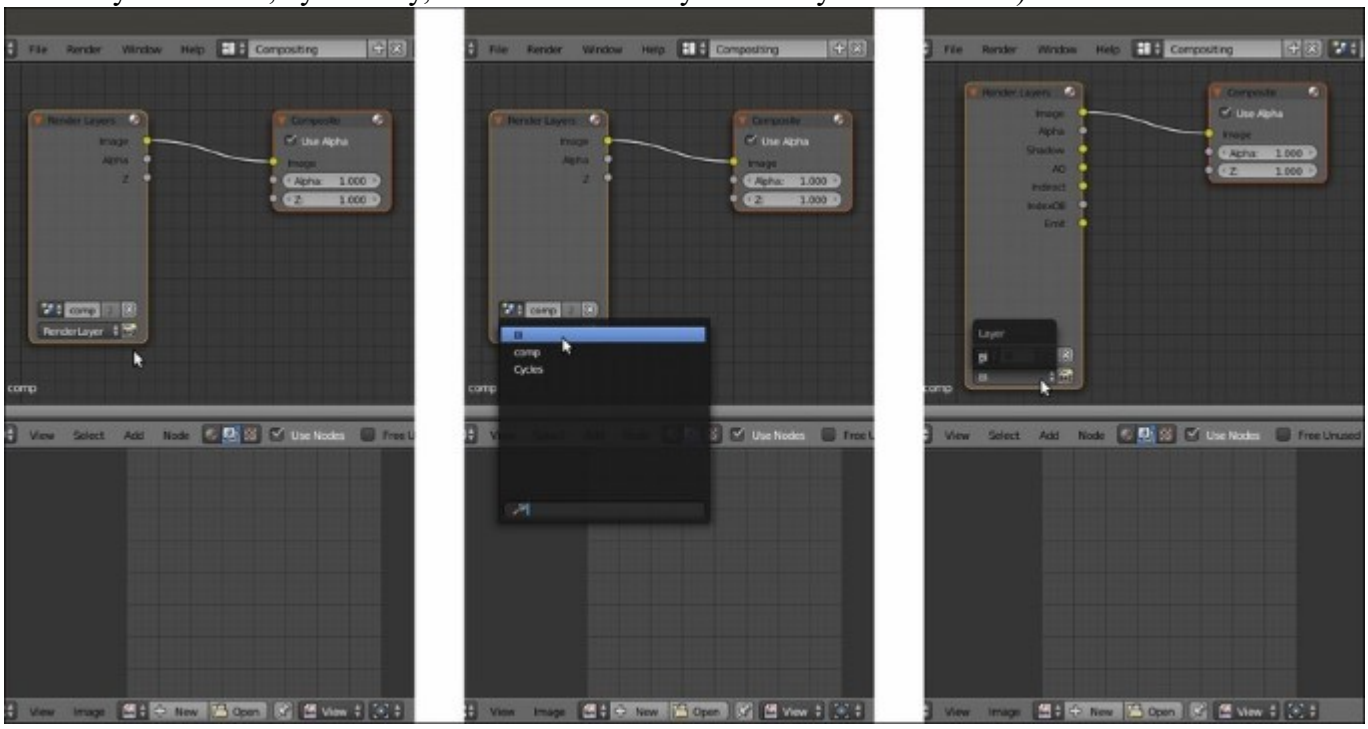
9. Now click on the *Screen datablock* button and switch to the **Compositing** screen.
10. In the **Node Editor** window toolbar, go to the *Node tree type to display and edit* row, click on the *Compositing nodes* button (the middle one) and then enable the **Use Nodes** checkbox:



The Compositing nodes button

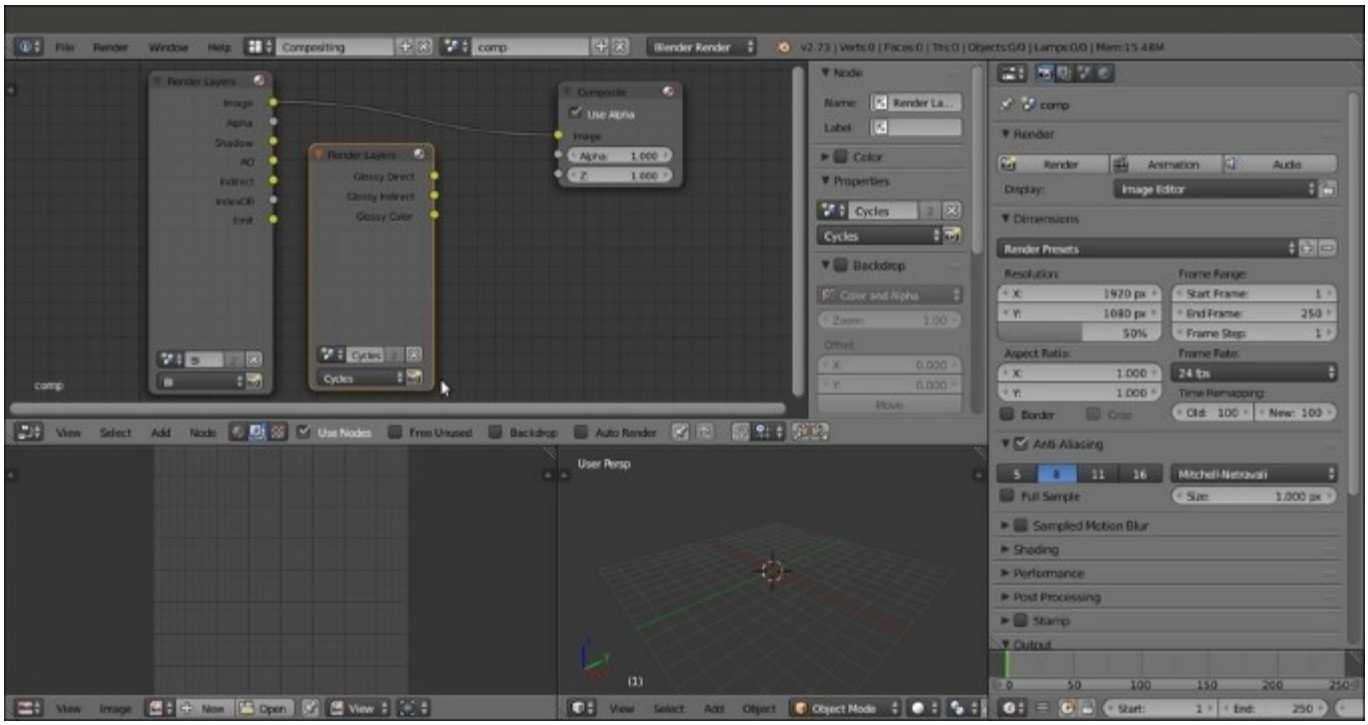
Two compositing nodes are automatically added in the **Node Editor** window: a **RenderLayers** node connected to a **Composite** node.

11. Click on the double arrows to the side of the *Scene datablock* on the **RenderLayers** node to switch from the **comp** scene to the **BI** scene, and, if necessary, in the bottom **Layer** button, select the name of the respective render layer (that we labeled as **BI** again; it shouldn't be necessary to select it, by the way, because it's the only render layer in the scene).



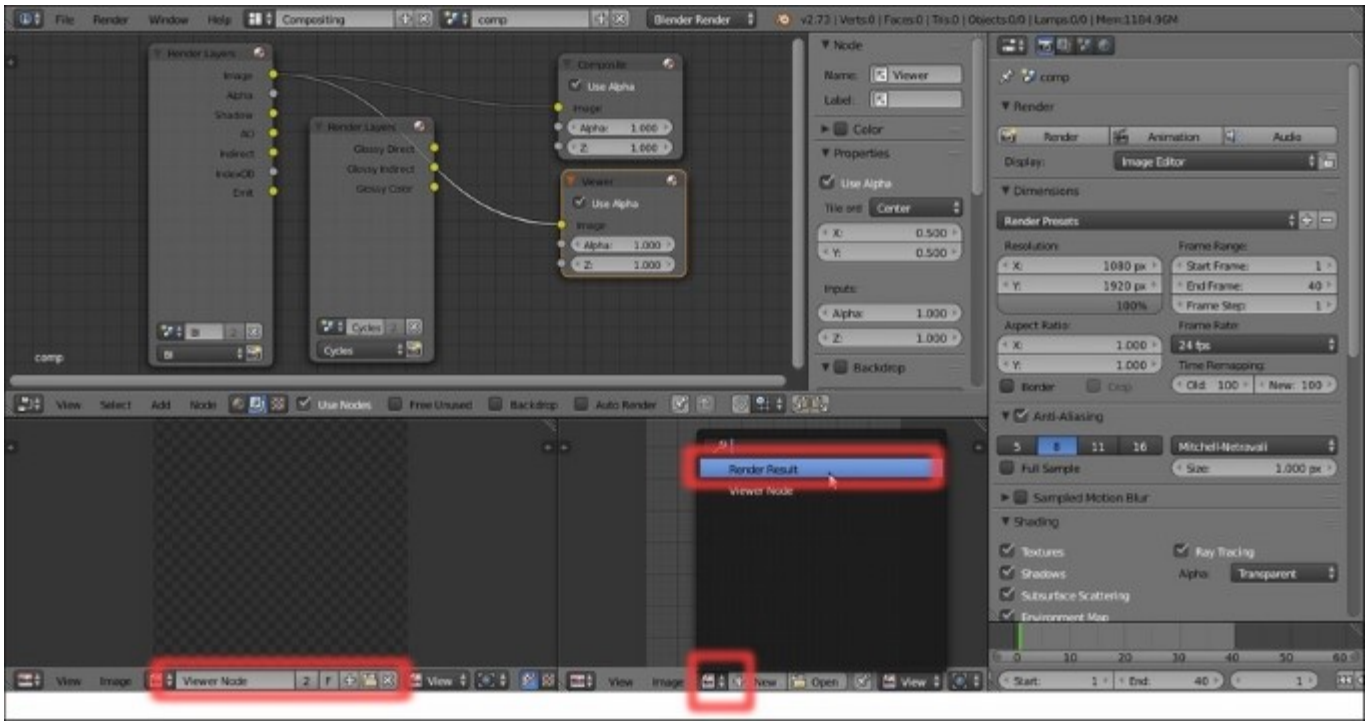
The render layer selector in the RenderLayers node and the output sockets to the Composite node

12. Press *Shift* + *D* to duplicate the **RenderLayers** node and repeat the procedure in the duplicated one, this time selecting the **Cycles** scene datablock and render layer:



The duplicated RenderLayers node

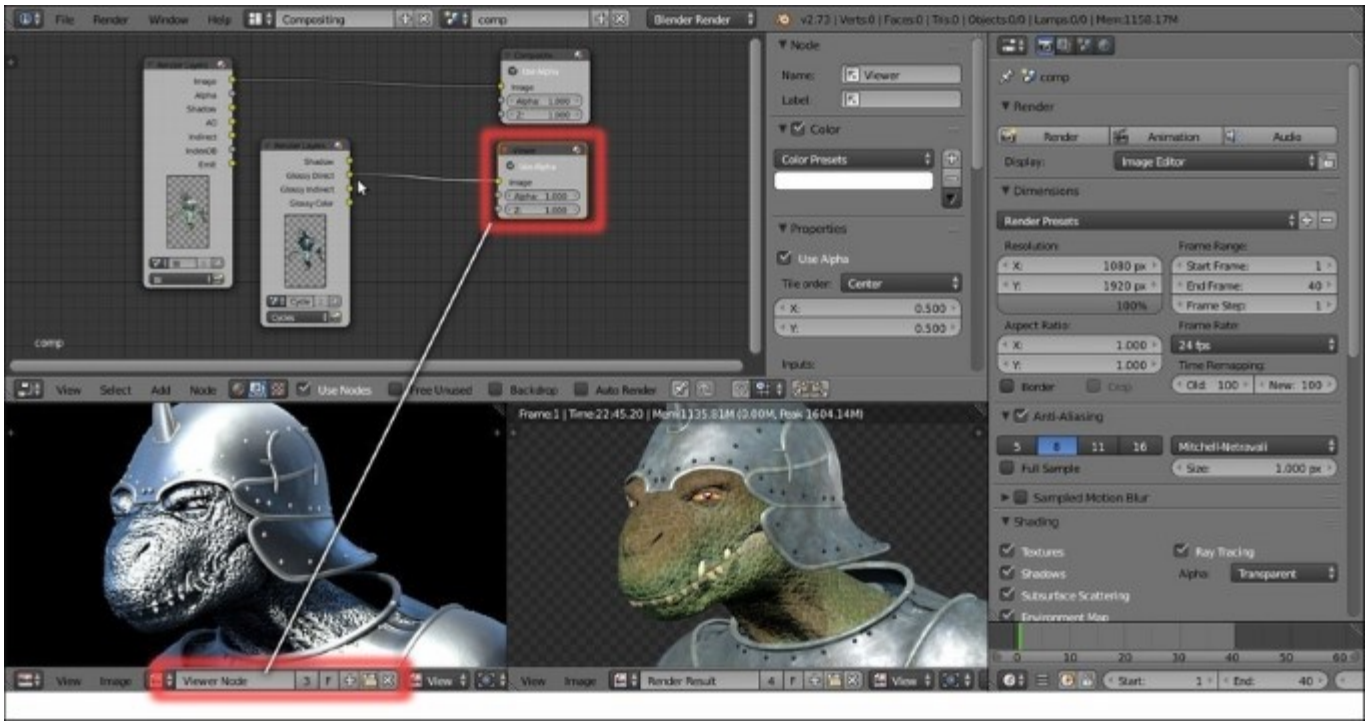
13. Put the mouse pointer inside the **Node Editor** window and press *Shift + A* to add a **Viewer** node (*Shift + A* | **Output** | **Viewer**); connect the **Image** output of the **BI RenderLayers** node also to the **Image** input socket of the **Viewer** node.
14. Move down and switch the **3D View** window with another **UV/Image Editor** window.
15. In the left image editor window, click on the double arrows to the left of the image datablock (*Browse Image to be linked*) and from the pop-up menu select the **Viewer Node** item.
16. In the right image editor window, instead, select the **Render Result** item.



The Viewer node and the Viewer Node and Render Result windows

17. At this point, press *F12* or click on the **Render** button inside the main **Properties** panel to start the rendering.

    When the rendering is done, only the **BI** render is visible in the two bottom editor windows, because at the moment it's the only **RenderLayers** node connected to the **Viewer** and to the **Composite** nodes.
18. Connect the **Glossy Direct** output of the **Cycles RenderLayers** node to the **Image** input socket of the **Viewer** node to see the single *specularity* pass in the left **UV/Image Editor**.



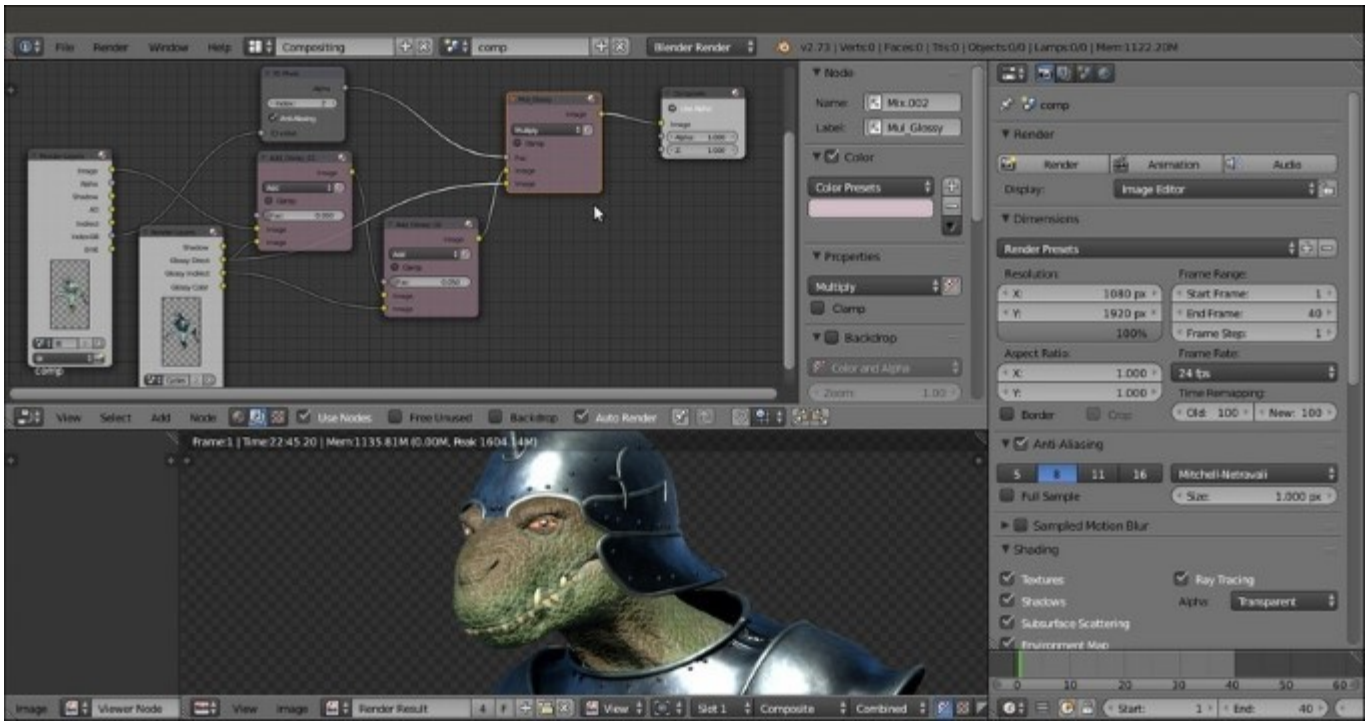
The Glossy Direct pass visualized in the Viewer Node window

In fact, we could also add more than one **Viewer** node to the **Node Editor** window and use them to visualize the different passes of **RenderLayers**, by connecting each pass output to each **Viewer** node; the last selected **Viewer** node will be the one visualized in the **Viewer Node** bottom window.

19. Enable the **Auto Render** item at the extreme right side on the **Node Editor** window toolbar, add a **Mix** node (*Shift* + *A* | **Color** | **Mix**) and paste it between the **Image** output of the **BI RenderLayers** node and the **Image** input socket of the **Composite** node; change the **Blend Type** to **Add** and then connect the **Glossy Direct** output of the **Cycles RenderLayers** node to its second **Image** input socket. Set the **Fac** value to **0.050** and label it as **Add_GLOSSY_01**.
20. If you want, also connect the output of the **Add_GLOSSY_01** node to the **Viewer** node.

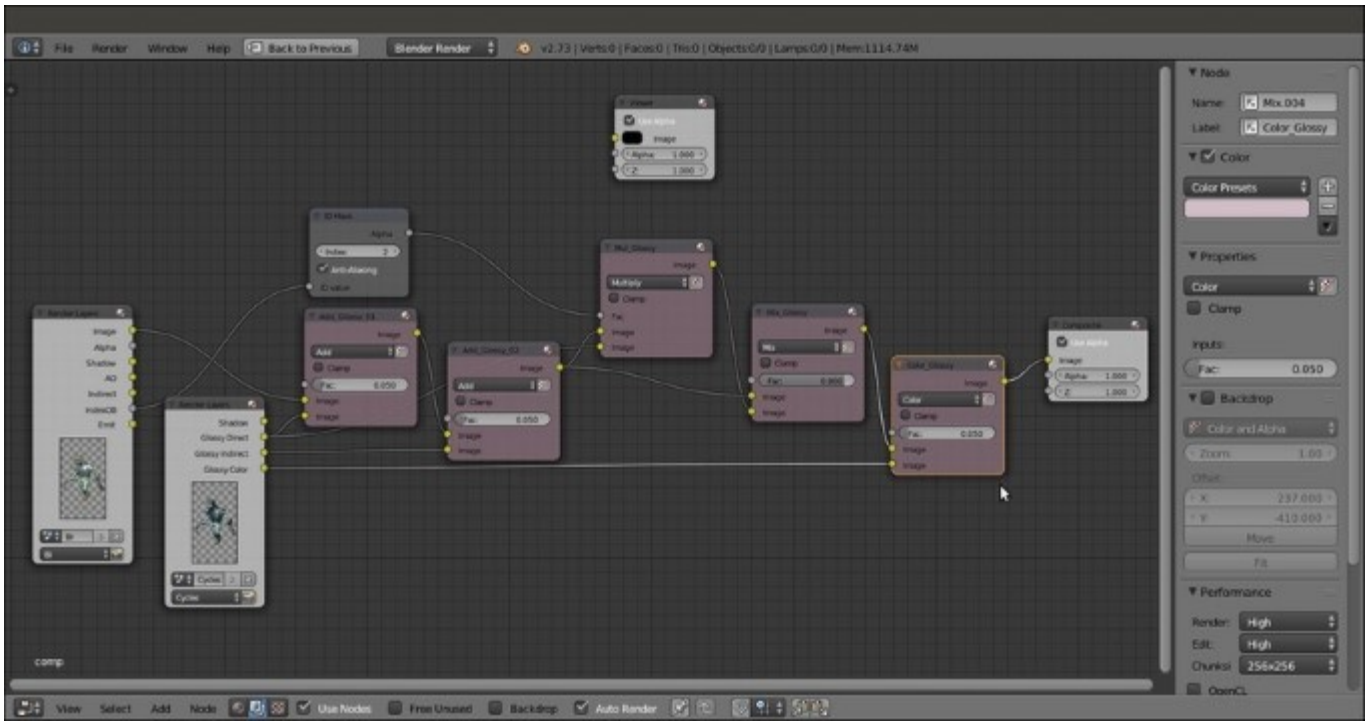
The Glossy pass, rendered in Cycles, added to the Image pass rendered in Blender Internal; note that now the armor looks too lightened

21. Press *Shift* + *D* to duplicate the **Add_GLOSSY_01** node, label it as **Add_GLOSSY_02** and paste it between the **Add_GLOSSY_01** and the **Composite** nodes; connect the **Glossy_Indirect** output of the **Cycles RenderLayers** node to the second **Image** input socket of the **Add_GLOSSY_02** node.
22. Press *Shift* + *D* to duplicate the **Add_GLOSSY_01** node again, label it as **Mul_GLOSSY**, change the **Blend Type** to **Multiply** and set the **Fac** value back to **1.000**; connect the output of the **Add_GLOSSY_02** node to its first **Image** input socket and the **Glossy_Direct** output of the **Cycles RenderLayers** node to its second **Image** input socket. Connect the output of the **Mul_GLOSSY** node to the **Composite** node.
23. Add an **ID Mask** node (*Shift* + *A* | **Converter** | **ID Mask**), set the **Index** value to **2** and connect the **IndexOB** output of the **BI RenderLayers** node to the **ID value** input socket, then connect the **Alpha** output to the **Fac** input socket of the **Mul_GLOSSY** node; enable the **Anti-Aliasing** item.
24. For the moment, disconnect the **Viewer** node.



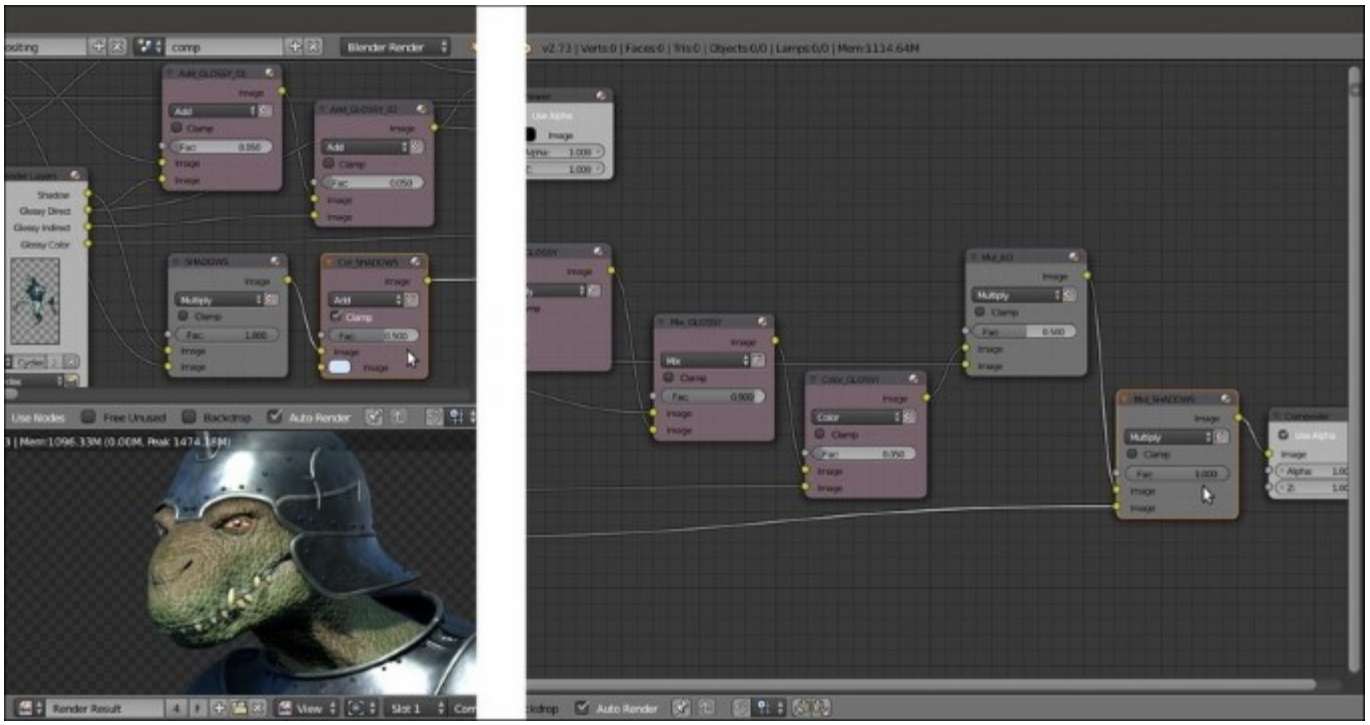
The IndexOB output used as a mask for the addition of the Glossy pass only on the character's skin

25. Press *Shift* + *D* to duplicate a new **Mix** node, label it as **Mix_GLOSSY** and set the **Blend Type** to **Mix** and the **Fac** value to **0.900**; connect the output of the **Add_GLOSSY_02** node to the first **Image** input socket and the output of the **Mul_GLOSSY** node to the second **Image** input socket.
26. Press *Shift* + *D* to duplicate another **Mix** node, label it as **Color_GLOSSY** and set the **Blend Type** to **Color** and the **Fac** to **0.050**; connect the output of the **Mix_GLOSSY** node to the first **Image** input socket and the **Glossy_color** output of the **Cycles RenderLayers** node to the second **Image** input socket:



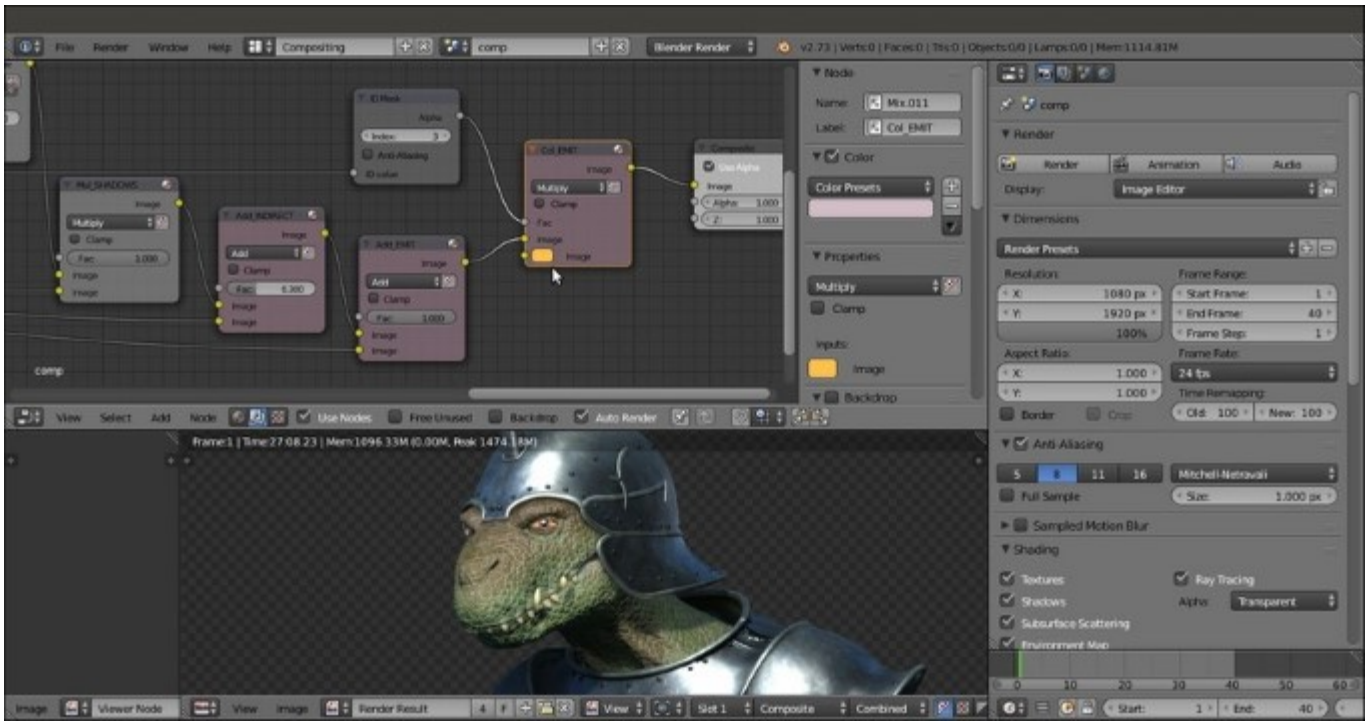
Adding more compositing nodes to re-build the separately rendered Glossy passes

27. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**) right after the **Color_GLOSSY** one, label it as **Mul_AO**, set the **Blend Type** to **Multiply** and connect the **AO** output of the **BI RenderLayers** node to the second **Image** input socket; set the **Fac** to **0.500**.
28. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **SHADOWS**, change the **Blend Type** to **Multiply** and set the **Fac** value to **1.000**; connect the **Shadow** output of the **BI RenderLayers** node to its first **Image** input socket and the **Shadow** output of the **Cycles RenderLayers** node to the second **Image** input socket.
29. Press *Shift* + *D* to duplicate the **Mul_AO** node and label it as **Mul_SHADOWS**; paste it right behind the **Mul_AO** node and set the **Fac** value slider to **1.000**.
30. Connect the output of the **SHADOWS** node to the second **Image** input socket of the **Mul_SHADOWS** node.
31. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **Color_SHADOWS** and paste it right behind the **SHADOWS** node; set the **Blend Type** to **Add**, the **Fac** to **0.500** and enable the **Clamp** item: set the color of the second **Image** socket to **R 0.640**, **G 0.780**, **B 1.000**.



Multiplying and coloring the shadow pass

32. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**) right behind the **Mul_SHADOWS** one, label it as **Add_INDIRECT**; set the **Blend Type** to **Add**, the **Fac** to **0.300** and connect the **Indirect** output of the **BI RenderLayers** to the second **Image** input socket.
33. Repeat the previous step but label the node as **Add EMIT**, set the **Fac** value to **1.000** and connect the **Emit** output of the **BI RenderLayers** node to the second **Image** input socket.
34. Add one more **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **Col_EMIT**, set the **Blend Type** to **Multiply** and paste it behind the **Add_EMIT** node; set the color of the second **Image** socket to **R 1.000**, **G 0.542**, **B 0.073**.
35. Add a new **ID Mask** node (*Shift* + *A* | **Converter** | **ID Mask**), connect the **IndexOB** output of the **BI RenderLayers** node to the **ID value** socket, set the **Index** value to **3** and connect its **Alpha** output to the **Fac** input socket of the **Col_EMIT** node.



Coloring and adding the eyes

36. Connect the output of the **Col_EMIT** also to the **Image** input socket of the **Viewer** node:

The completed compositing network

37. Save the file.

How it works…

In the *Getting ready* section:

- From step 2 to step 5 we disabled the **env_refl_skin** and the **env_refl_armor** texture slots in all the material nodes of the **BI** shaders; in fact, because we are going to add the **Cycles** reflection on the **BI** diffuse, we don't need to fake the environment reflection on the character and on the armor surfaces anymore.
- In step 6 we assigned a different **Index Pass** number to each one of the objects; this is useful to later *separate* the objects in the **Compositor** for particular effects (in our case we only needed the armor **Index Pass** number, but it's a good habit to give index passes to all the objects for any eventuality).
- At step 7 we saved the file with a new name, and at steps 8 and 9 we changed, in the file to be rendered, the path to the new library file.

In the *How to do it…* section:

- From step 1 to step 3 we added a new empty scene, **comp**, to the blend file, linked to the **Compositing** screen layout and to be used for the compositing.
- In fact, in the **comp** scene, all the **compositing** nodes are connected together so as to recreate the best possible rendering look of the **Gidiosaurus**, using the different passes from the different **BI** and **Cycles** scenes through the **render layers**.
- From step 4 to step 7 we enabled the required passes in the **Render layers** windows of both the **BI** and **Cycles** scenes; note that basically we set an almost complete *only-diffuse* render in **Blender Internal** (besides the **Indirect**, **Ambient Occlusion** and **Emit** passes, subtracted from the total render and separately outputted), while we set the **Glossy** passes in the **Cycles** engine.
- From step 8 to step 15 we set up the **RenderLayers** nodes, the **Viewer** and the **Composite** nodes.

  The **RenderLayers** node outputs the rendering of a particular scene also delivering the enabled passes for that scene, through the **Render Layer** setup.

  The **Compositing** node is the mandatory final output node and must always be connected as the last step of the compositing chain.

  The **Viewer** node, instead, is optional but always used anyway to visualize the different steps of the compositing itself.
- At step 16 we started the rendering; Blender starts to render the **BI** and the **Cycles** scenes using the settings setup for each scene (and not the settings of the **comp** scene, which are true only for the compositing).
- From step 18 to step 34 we mixed the different passes together. Basically, we used the same approach that we have seen in the creation of the shaders, that is, by decomposing the final result into the different components; then, we obtained the diffuse from the **Blender Internal** engine because it's quite fast for that, and the glossy component from the **Cycles** render engine for the same reasons, and then we re-mixed them. The **Subsurface Scattering** pass is actually

rendered together at the diffuse component in **BI** and is delivered through the **Combined** (**Image**) pass.

- The **Mix_GLOSSY** node added at step 24 is to tweak the strength of the multiplied **Glossy Direct** pass; we couldn't use the **Fac** value, in this case, because it was already used by the **ID Mask** output to *isolate*, thanks to the **Object Index**, the **Armor** from the rest of the render.
- With the **Col_SHADOWS** node at step 30 we obtained two goals: first, we set the dark intensity of the shadows to **0.500** and, second, we gave them a bluish coloration.
- The emission pass for the eyes has been added to the rendering through the same technique used to multiply the glossy only on the **Armor**, that is, by an **ID Mask** node and the **Object Index**.

See also

- http://www.blender.org/manual/render/blender_render/layers.html
- http://www.blender.org/manual/render/blender_render/passes.html
- http://www.blender.org/manual/render/post_process/layers.html
- http://www.blender.org/manual/composite_nodes/index.html

Part 3. Module 3

Blender Cycles: Materials and Textures Cookbook - Third Edition

Over 40 practical recipes to create stunning materials and textures using the Cycles rendering engine with Blender

Chapter 1. Overview of Materials in Cycles

In this chapter, we will cover the following recipes:

- An overview of material nodes in Cycles
- An overview of procedural textures in Cycles
- How to set the World material
- Creating a mesh-light material
- Using volume materials
- Using displacement

Introduction

Cycles' materials work in a totally different way than in Blender Internal.

In Blender Internal, you can build a material by choosing a diffuse and a specular shader from the **Material** window, by setting several surface options, and then by assigning textures (both procedurals and image maps as well) in the provided slots. All of these steps make one complete material. After this, it's possible to combine two or more of these materials by a network of nodes, thereby obtaining a lot more flexibility in a shader's creation. However, the single materials themselves are the same as those set through the **Material** window—shaders made for a scan-line-rendering engine—and their result is just an approximation of the simulated absorption-reflection behavior of light on a surface.

In Cycles, the approach is quite different. All the names of the closures describing surface properties have a **Bidirectional Scattering Distribution Function** (**BSDF**), which is a general mathematical function that describes the way in which light is scattered by a surface in the real world. It's also the formula that path tracers such as Cycles use to calculate the rendering of an object in a virtual environment. Basically, light rays are shot from the camera. They bounce on the objects in the scene and keep on bouncing until they reach a light source or an empty background (which, in Cycles, can emit light as well). For this reason, a pure path tracer such as Cycles can render in reasonable times an object set in an open environment. The rendering times increase a lot for closed spaces, for example, furniture set inside a room, because light rays can bounce on the floor, the ceiling, and the walls many times before reaching one or more light sources.

In short, the main difference between the two rendering engines is due to the fact that, while in Blender Internal, the materials use all the traditional shader tricks of a scan-line rendering engine such as the simulated specular component, the Cycles rendering engine is a path tracer that tries to mimic the real behavior of a surface as closely as possible as if the surface were real. This is the reason we don't have an arbitrary Specular factor simulating the reflection point of light on the surface in Cycles, but instead have a glossy shader that actually mirrors the light source and the surroundings to be mixed with other components in different ratios. Thus the glossy shader behaves in a more realistic way.

Just for explanatory purposes, in this module, I will refer to the more or less blurred point of light created by the reflection of the light source on a mirroring glossy surface as *specularity*.

Be aware that the rendering speed in Cycles depends on the device you use to render your scenes—CPU or GPU. This means that basically, you can decide to use the power of the CPU (default option) or the power of the graphic card processor, the GPU.

To set the GPU for the rendering, perform the following steps:

1. Call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*) and go to the **System** tab, the last tab to the right of the panel.
2. Under the **Compute Device** tab to the bottom-left corner of the panel, select the option to be used for computation. To make this permanent, click on the **Save User Settings** button or press *Ctrl* + *U*. Now close the **Blender User Preferences** panel.
3. In the **Properties** panel to the right of the screen, go to the **Render** window and, under the **Render** tab, it's now possible to configure the GPU of the graphics card instead of the default CPU (this is possible only if your graphic card supports CUDA, that is, for NVIDIA graphic cards. OpenCL, which is intended to support rendering on AMD/ATI graphics cards, is still in a very incomplete and experimental stage, and therefore, not very usable yet).

A GPU-based rendering has the advantage of literally increasing the Cycles' rendering speed several times, albeit with the disadvantage of a small memory limit, so it's not always possible to render big complex scenes made up of a lot of geometry. In such cases, it's better to use the CPU instead.

There are other ways to reduce the rendering times and also to reduce or avoid the noise and the fireflies (white dots) produced in several cases by the glossy, transparent, and light-emitting materials. All of this doesn't strictly belong to shaders or materials. By the way, you can find more information related to these topics at the following addresses:

Information on *Cycles Render Engine* can be found at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles.

More information on *Reducing Noise* is available on the Cycles wiki page, at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Reducing_Noise.

A list of supported graphic cards for Cycles can be found at https://developer.nvidia.com/cuda-gpus.

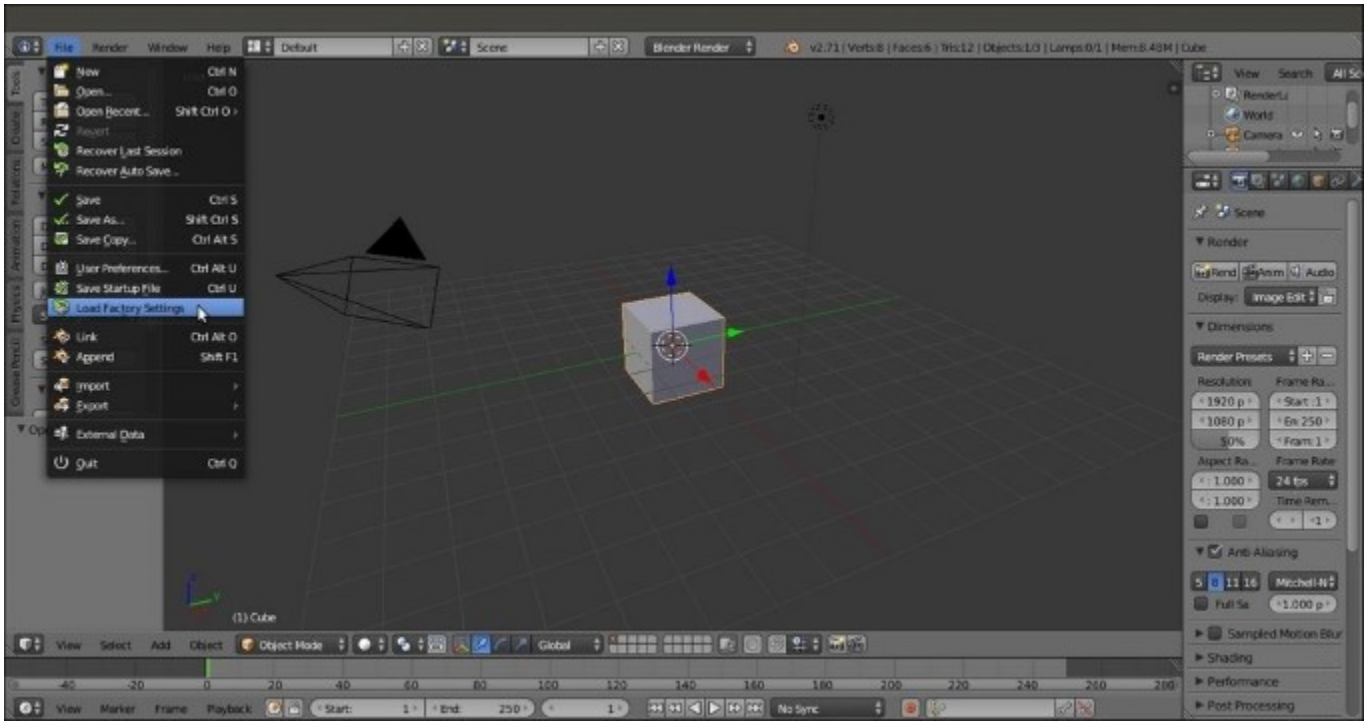
Material nodes in Cycles

A Cycles material is basically made up of distinct components named **shaders**. They can be combined to build even more complex surface or volume shaders.

In this recipe, we'll have a look at the basic, necessary steps required to build a basic surface Cycles material, to activate the rendered preview in the 3D window, and to finally render a simple scene.

Getting ready

In the description of the following steps, I'll assume that you are using Blender with the default factory settings. If you aren't, start Blender and just click on the **File** menu item in the top main header bar to select **Load Factory Settings** from the pop-up menu, as shown in the following screenshot:



The default Blender interface and the File pop-up menu with the Load Factory Settings item

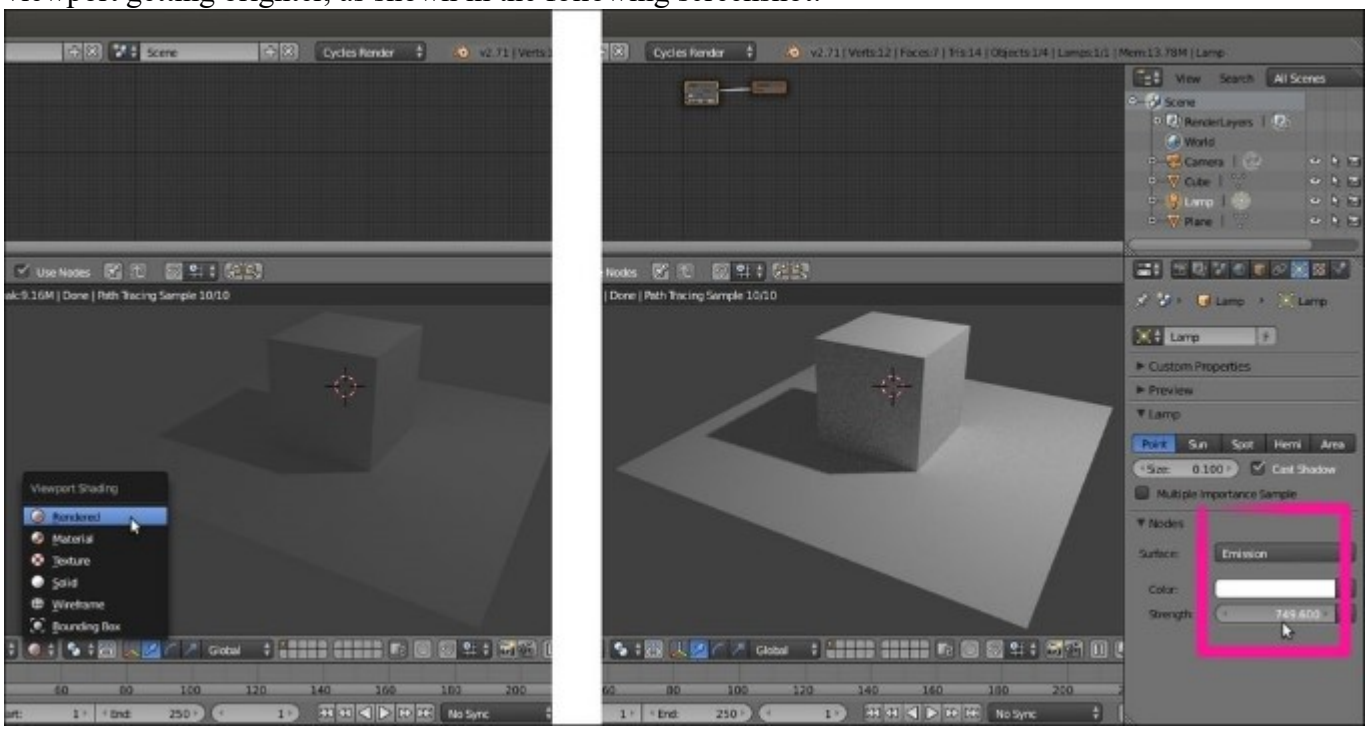
Now perform the following steps:

1. In the upper menu bar, switch from **Blender Render** to **Cycles Render** (hovering with the mouse on this button shows the engine to use to render a label).
2. Now split the 3D view into two horizontal rows, and change the upper row to the **Node Editor** window by selecting the menu item from the **Editor Type** button in the left corner of the bottom bar of the window. The **Node Editor** window is, in fact, the window we will use to build our shaders by mixing the nodes (actually, this is not the only way, but we'll see this later).

3. Put the mouse cursor in the 3D view and add a Plane under the Cube (press *Shift* + *A* and navigate to **Mesh** | **Plane**). Enter **Edit Mode** (press *Tab*), scale it 3.5 times bigger (press *S*, enter *3.5*, and then press *Enter*) and go out of **Edit Mode** (press *Tab* again). Now move the Plane one Blender unit down (press *G*, then *Z*, then enter *-1*, and finally, press *Enter*).
4. Go to the little icon (**Viewport Shading**) showing a sphere in the bottom bar of the 3D view and click on it. A menu showing different options appears (**Bounding Box**, **Wireframe**, **Solid**, **Texture**, **Material** and **Rendered**). Select **Rendered** from the top of the list (or press the *Shift* + *Z* shortcut) and watch your Cube being rendered in real time in the 3D viewport.
5. Now you can rotate and translate the view or the Cube itself, and the view gets updated in real time (the speed of the update is restricted only by the complexity of the scene and the computing power of your CPU or graphics card).

Let's learn more by performing the following steps:

1. Select the **Lamp** item in the **Outliner** window (by default, it's a Point lamp).
2. Go to the **Object data** window under the **Properties** panel on the right-hand side of the screen.
3. Under the **Nodes** tab, click on **Use Nodes** to activate a node system for the selected light in the scene. This node system is made by an **Emission** shader connected to a **Lamp Output** node.
4. Go to the **Strength** item, which is set to `100.000` by default, and start increasing the value. As the intensity of the Lamp increases, you will see the Cube and the Plane rendered in the viewport getting brighter, as shown in the following screenshot:



The Viewport Shading menu with the Rendered item and the Lamp Object data window with the Strength slider

Tip

Downloading the example code

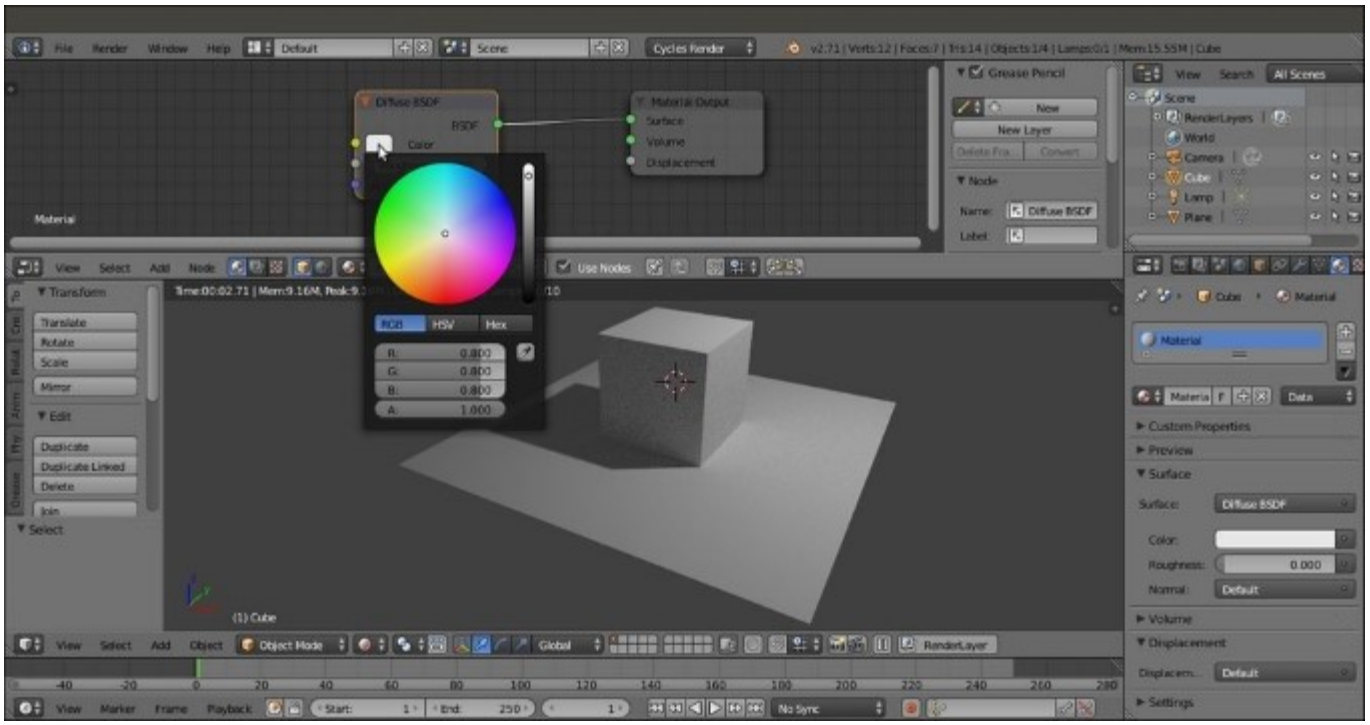
You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

How to do it...

We just prepared the scene and took the first look at one of the more appreciated features of Cycles since its first inclusion in Blender—the real-time-rendered preview (which, by the way, is now also available in Blender Internal but seems to work faster in Cycles).

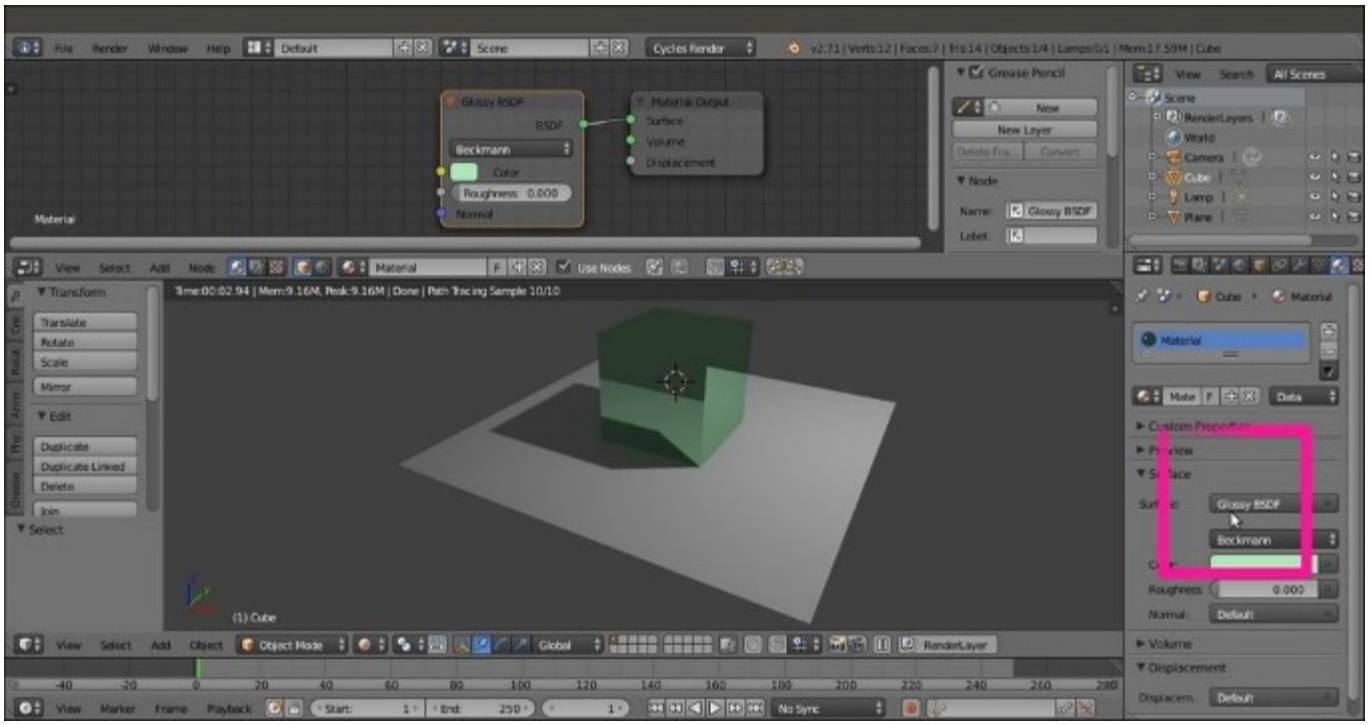
Now let's start with the object's materials:

1. Select the Cube to assign the shader to by clicking on the item in the **Outliner** window or right-clicking directly on the object in the **Rendered** viewport (but be aware that in the **Rendered** mode, the object selection outline usually around the mesh is not visible because it's obviously not renderable).
2. Go to the **Material** window under the **Properties** panel. The Cube already has a default material assigned (as you can precisely see under the **Surface** subpanel within the **Material** window). By the way, you need to click on the **Use Nodes** button under the **Surface** subpanel to activate the node system for the material. Instead of this, you can also check the **Use Nodes** box in the toolbar of the **Node Editor** window.
3. As you check the **Use Nodes** box, the content of the **Surface** tab changes, showing that a **Diffuse BSDF** shader has been assigned to the Cube and that, accordingly, two linked nodes have appeared inside the **Node Editor** window. The **Diffuse BSDF** shader is already connected to the **Surface** input socket of a **Material Output** node.
4. Put the mouse cursor in the **Node Editor** window, and by scrolling the mouse wheel, zoom in to the **Diffuse BSDF** node. Click on the **Color** rectangle. A color wheel appears, where you can select a new color to change the shader color by clicking on the wheel itself or by inserting the **RGB** values (note that there is also a color sampler and the alpha channel value, although the latter, in this case, doesn't have any visible effect on the object material's color).



The color wheel of a Diffuse shader node in the Node Editor window and the Rendered 3D viewport preview

5. The Cube rendered in the 3D preview changes its material's color in real time. You can even move the cursor in the color wheel and watch the rendered object switching the colors accordingly. Set the object's color to a greenish color by changing its **RGB** values to `0.430`, `0.800`, and `0.499`, respectively.
6. Go to the **Material** window, and under the **Surface** tab, click on the **Surface** button, which is showing the **Diffuse BSDF** item at the moment. From the pop-up menu that appears, select the **Glossy BSDF** shader item. Now the node changes in the **Node Editor** window, and so does the Cube's material in the **Rendered** preview, as shown in the following screenshot:



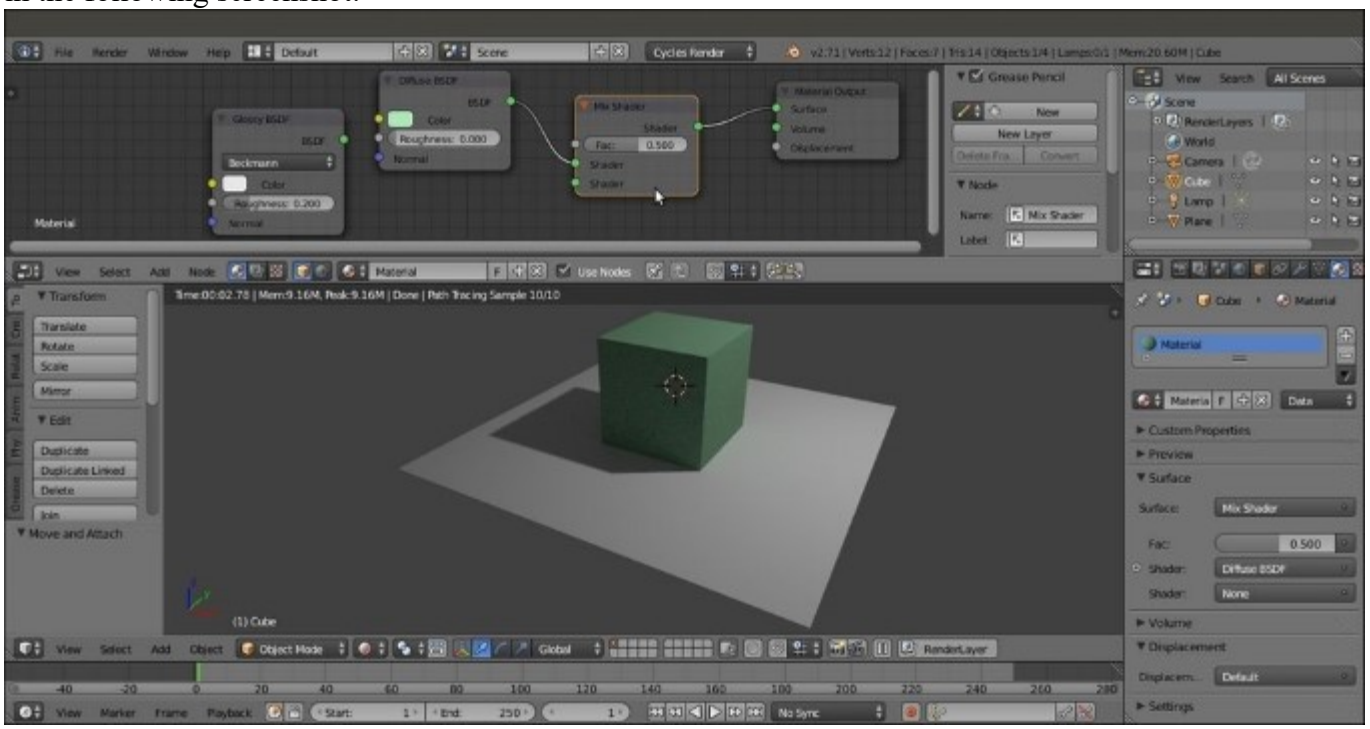
Real-time preview of the effect of the Glossy shader node and the Surface subpanel under the Material window

Note that although we just switched a shader node with a different node, the color we set in the former has been kept in the new one. Actually, this happens for all the values that can be kept from a node to a different one.

Now, because a material having a 100 percent matte or reflective surface could hardly exist in the real world, a more accurate basic Cycles material should be made by mixing the **Diffuse BSDF** and the **Glossy BSDF** shaders, blended together by a **Mix Shader** node, which in turn is connected to the **Material Output** node:

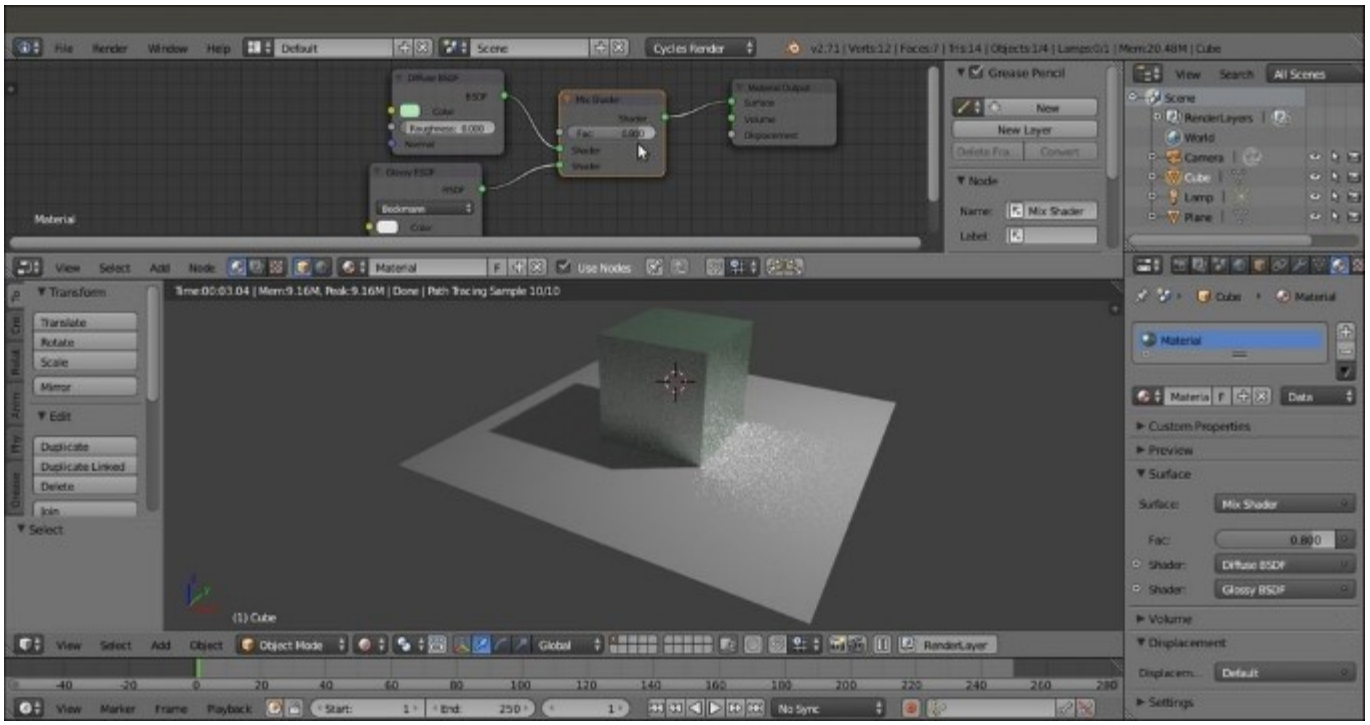
1. In the **Material** window, under the **Surface** tab, click again on the **Surface** button, which is now showing the **Glossy BSDF** item, and replace it with a **Diffuse BSDF** shader.
2. Put the mouse pointer on the **Node Editor** window, and by pressing *Shift* + *A*, make a pop-up menu appear with several items. Move the mouse pointer on the **Shader** item. It shows one more pop-up, where all the shader items are collected. Alternatively, press the *T* key to call the **Node Editor** tool shelf, where you can find the same shader items under the different tabs.
3. Select one of these items (in our case, the **Glossy BSDF** shader node again). The **Shader** node, which is already selected, is now added to the **Node Editor** window, although it is not connected to anything yet (in fact, it's not visible in the **Material** window but only in the **Node Editor** window).
4. Again press *Shift* + *A* in the **Node Editor** window, and this time, add a **Mix Shader** node.

5. Press *G* to move the node to the link connecting the **Diffuse BSDF** node to the **Surface** input socket of the **Material Output** node (you'll probably need to first adjust the position of the two nodes to make room between them). The **Mix Shader** node gets automatically pasted in between, and the **Diffuse** node output gets connected to the first **Shader** input socket, as shown in the following screenshot:



Mix Shader node pasted between a preexisting nodes connection inside the Node Editor window

6. Click on the green dot output of the **Glossy BSDF** shader node, and grab the link to the second input socket of the **Mix Shader** node. Release the mouse button now and see the nodes being connected.
7. Because the blending **Fac** (factor) value of the **Mix Shader** node is set by default to `0.500`, the two shader components, **Diffuse** and **Glossy**, are now showing on the Cube's surface in equal parts, that is, each component at 50 percent. Click on the **Fac** slider with the mouse and slide it to `0.000`. The Cube's surface now shows only the **Diffuse** component because the **Diffuse BSDF** shader is connected to the first **Shader** input socket, which is corresponding to a value of `0`.
8. Slide the **Fac** slider value to `1.000` and the surface now shows only the **Glossy BSDF** shader component, which is, in fact, connected to the second **Shader** input socket corresponding to a value of `1`.
9. Set the **Fac** value to `0.800` (keep *Ctrl* pressed while you are sliding the **Fac** value to constrain it to `0.100` intervals). The Cube is now reflecting the white Plane on its sides, even though it is blurred, because we have a material that is reflective at 80 percent and matte at 20 percent (the white noise you see in the rendered preview is due to the low sampling we are using at the moment. You will learn more about this later). This is shown in the following screenshot:



The Rendered preview of the effect of the mixed Diffuse and Glossy shader nodes

10. Lastly, select the Plane, go to the **Material** window, and click on the **New** button to assign a diffuse whitish material.

How it works...

In its minimal form, a Cycles material is made by any one of the node shaders connected to the **Surface** or the **Volume** input sockets of the **Material Output** node. For a new material, the node shader is **Diffuse BSDF** by default, with the **RGB** color set to `0.800` and connected to the **Surface** socket, and the result is a matte whitish material (with the **Roughness** value at `0.000`, actually corresponding to a **Lambert** shader).

Then the **Diffuse BSDF** node can be replaced by any other node of the available shader list, for example, by the **Glossy BSDF** shader as in the former Cube scene, which produced a totally mirrored surface material.

As we have seen, the **Node Editor** window is not the only way to build the materials. In the **Properties** panel on the right-hand side of the UI, we have access to the **Material** window, which is usually divided as follows:

- The material name, user, and the **datablock** subpanel.
- The **Preview** window.
- The **Surface** subpanel, including only the shader nodes added in a vertically ordered column in the **Node Editor** window, and already connected to each other.
- The **Volume** subpanel, with the similar feature as that of the **Surface** subpanel.

- The **Displacement** subpanel.
- The **Settings** subpanel, where we can set the object color, the alpha intensity, the specularity color, and the hardness as seen in the viewport in non-rendered mode (**Viewport Color, Alpha**, **Viewport Specular**, and **Hardness**). It also contains the **Pass Index** value of the material, a **Multiple Importance Sample** checkbox, the Volume sampling methods, the Interpolation, the Homogeneous item to be activated to accelerate the rendering of volumes, and an option to disable the rendering of the transparent shadows to accelerate the total rendering.

The **Material** window not only reflects what we do in the **Node Editor** window and changes accordingly (and vice versa), but can also be used to change the values to easily switch the shaders themselves, and to some extent, to connect them to the other nodes.

The **Material** and the **Node Editor** windows are so mutual that there is no prevalence in which window to use to build a material. Both can be used individually or combined, depending on preferences or practical utility. In some cases, it can be very handy to switch a shader from the **Surface** tab under **Material** on the right (or a texture from the **Texture** window as well, but we'll see textures later), leaving all the settings and the links in the node's network untouched.

There is no question, by the way, that the **Material** window can become pretty complex and confusing as a material network grows more and more in complexity, while the graphic appearance of the **Node Editor** window shows the same network in a clearer and much more readable way.

There's more...

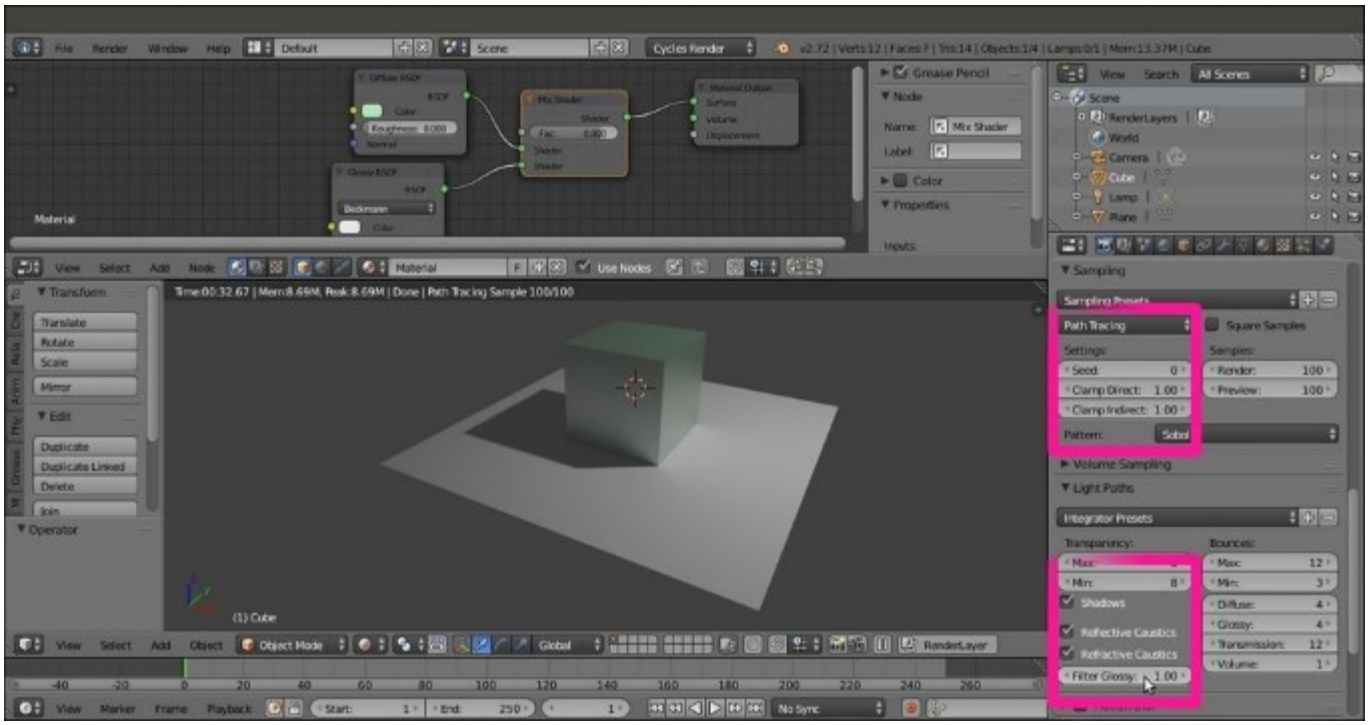
Looking at the **Rendered** viewport, you'll notice that the image is now quite noisy and that there are white dots in certain areas of the image. These are the infamous fireflies, caused mainly by transparent, luminescent, or glossy surfaces. Actually, they have been introduced in the rendering of our Cube by the glossy component.

Here is one way to eliminate the fireflies:

1. Go to the **Render** window under the **Properties** panel.
2. Uncheck both the **Reflective** and **Refractive Caustics** items under the **Light Path** subpanel.
3. This will immediately eliminate the white noise, but alas! It also eliminates all the caustics (which we would like to keep in the rendering in most cases).

Therefore, a different approach is as follows:

1. Go to the **Render** window under the **Properties** panel. In the **Sampling** tab, set **Samples** to `100` for both **Preview** and **Render** (they are set to `10` by default).
2. Set the **Clamp Direct** and **Clamp Indirect** values to `1.00` (they are set to `0.00` by default).
3. Go to the **Light Paths** tab, re-enable the **Reflective** and **Refractive Caustics** items, and then set the **Filter Glossy** value to `1.00`.
4. The resulting rendered image, as shown in the following screenshot, is now a lot smoother and noise-free, and also keeps the reflected caustics on the Plane:



Noise-free Rendered preview and settings under the Render window

5. Save the blend file in an appropriate location on your hard drive with a name such as `start_01.blend`.
6. The **Samples** set to `10` by default are obviously not enough to give a noiseless image, but are good for a fast preview. We could also let the **Preview** samples remain at the default value and increase only the **Render** value, to have longer rendering times but a clean image only for the final render (which can be started, as in Blender Internal, by pressing the *F12* key).

Using the **Clamp** value, we can reduce the energy of the light. Internally, Blender converts the image color space to linear, which is from `0` to `1`, and then reconverts it to **RGB**, which is from `0` to `255`, for the output. A value of `1.00` in linear space means that all the image values are now included inside a range starting from `0` and arriving to a maximum value of `1`, and that values greater than `1` are not possible, thus avoiding the fireflies problem in most cases. Be aware that **Clamp** values higher than `1.00` might also lower the general lighting intensity of the scene.

The **Filter Glossy** value is exactly what the name says, a filter that blurs the glossy reflections on the surface to reduce noise.

Remember that even with the same samples, the **Rendered** preview does not always have a total correspondence to the final render with regards to both noise and the fireflies. This is mainly due to the fact that the preview-rendered 3D window and the final rendered image usually have very different sizes, and artifacts visible in the final rendered image may not show in a smaller preview-rendered window.

See also

As you have seen, the several nodes that can be used to build Cycles shaders have both input and output sockets to the left and to the right of the node interface, respectively, and the color of these sockets is actually indicative of their purpose; green sockets are for shaders, yellow sockets are for colors, gray sockets for values, and blue sockets for vectors.

Each color output socket of one node should be connected with the same color input socket of another node. By the way, connecting differently colored sockets also works quite often; for example, a yellow color output can be connected to a gray value input socket and to a blue vector input.

A general overview of all the Cycles nodes can be found at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes.

Procedural textures in Cycles

In this recipe, we'll see several kinds of textures available in Cycles, and learn how to use them with the shaders.

Similar to Blender Internal, we can use both procedural textures and image textures in Cycles. However, the Cycles procedural textures are not exactly the same as in Blender Internal. Some textures are missing because they have been replaced by an improved version (for example, the **Clouds** procedural texture has been replaced by particular settings of the **Noise** procedural texture), and a few textures are new and exclusive to Cycles.

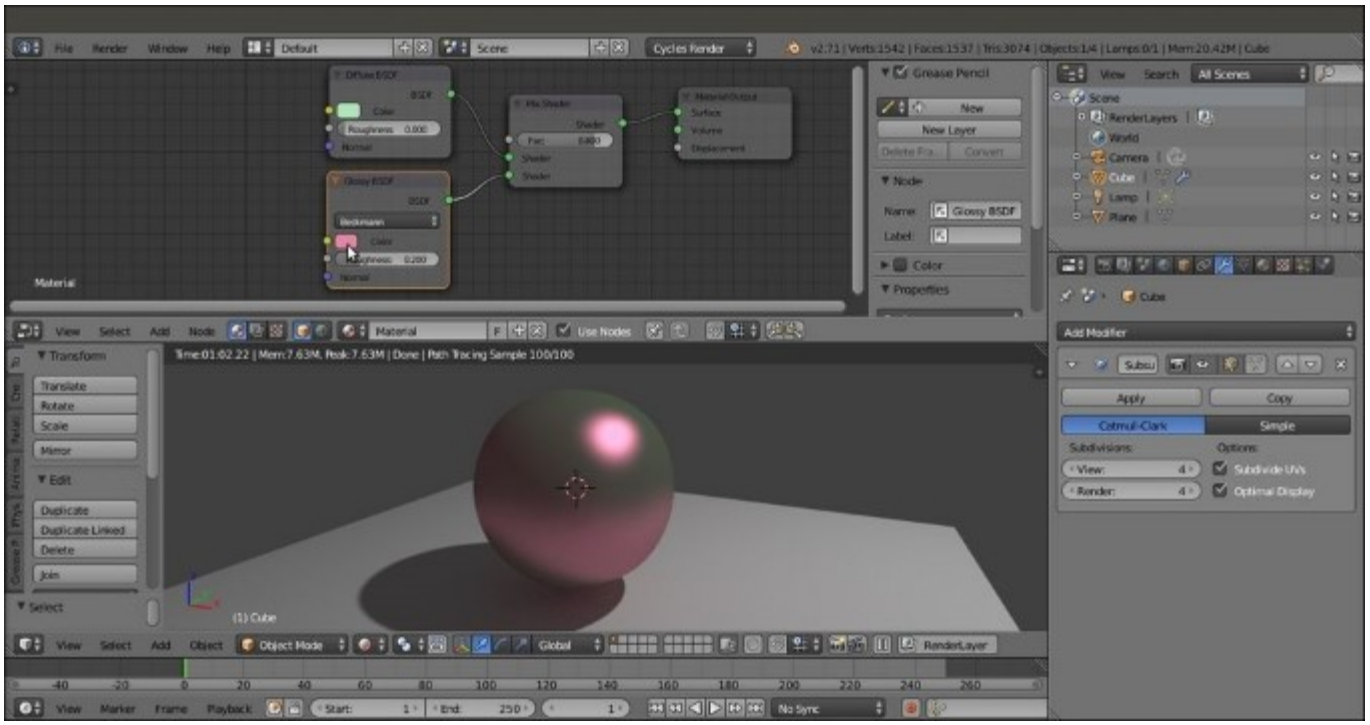
Getting ready

We have already seen a simple construction of a basic Cycles material by mixing the diffuse and the glossy (specular) components of a surface. Now let's take a look at the textures we can use in Cycles to further refine a material.

Because Cycles has a node-based system for materials, textures are not added in their slot under a tab as they are in Blender Internal. They get added in the **Node Editor** window, and are directly connected to the input socket of the shaders or other kinds of nodes. This gives a lot more flexibility to the material creation process because a texture can be used to drive several options inside the material network.

Let's see how they work:

1. Starting from the previously saved `start_01.blend` blend file, where we already set a simple scene with a Cube on a Plane and a basic material, select the Cube and go to the **Object modifiers** window inside the **Properties** panel to the right of the UI.
2. Assign to the Cube a **Subdivision Surface** modifier, set the **Subdivisions** level to `4` for both **View** and **Render**, and check the **Optimal Display** item.
3. Go to the **Tool** tab at the left of the 3D window, navigate to **Edit** | **Shading**, and set the subdivided Cube (let's call it Spheroid from now on) to **Smooth**.
4. Just to make things clearer, click on the color box of the **Glossy BSDF** shader to change it to a purple color (RGB set to `0.800`, `0.233`, and `0.388`, respectively). Note that only the glossy reflection part on the Spheroid is now purple, whereas the rest of the surface, which is the diffuse component, is still greenish.
5. Save the blend file and name it `start_02.blend`. The effect visible in the real-time **Rendered** preview is as follows:

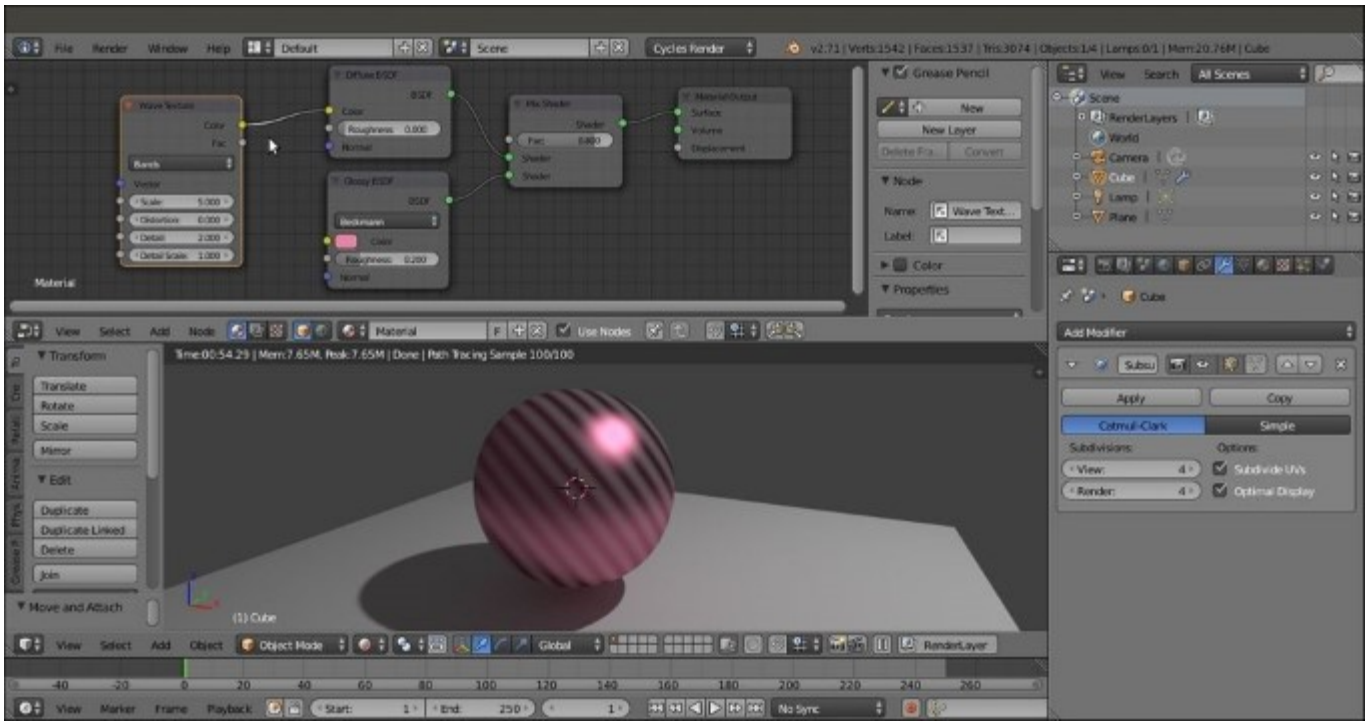


The Rendered preview of the effect of two differently colored Diffuse and Glossy components on the Spheroid

How to do it...

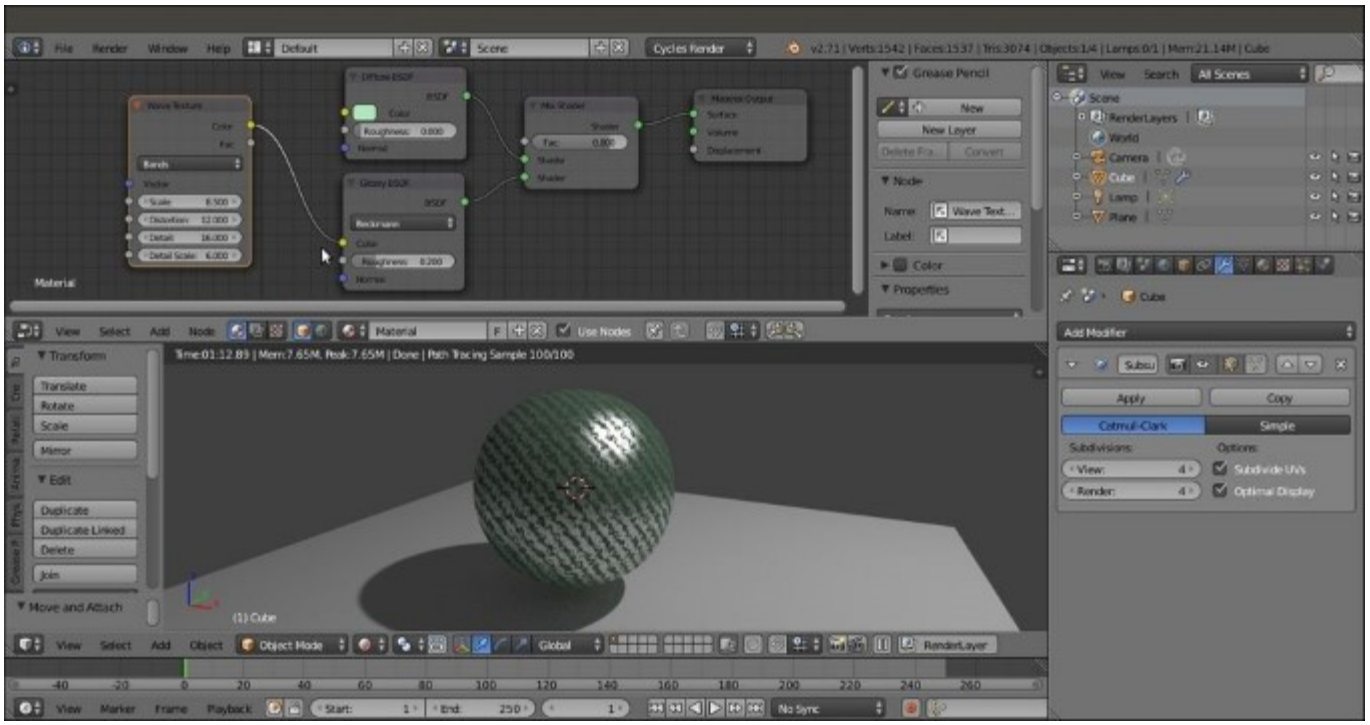
Perform the following steps to add a procedural texture to the object:

1. Put the mouse pointer in the **Node Editor** window and press *Shift* + *A*.
2. In the contextual pop-up menu, go to the **Texture** item, just under **Shader**, and click on **Wave Texture** to add the texture node to the **Node Editor** window.
3. Grab and connect the yellow **Color** output socket of the texture to the yellow input socket of the **Diffuse** shader, the socket close to the **Color** rectangle that we formerly set as a greenish color, as shown in this screenshot:



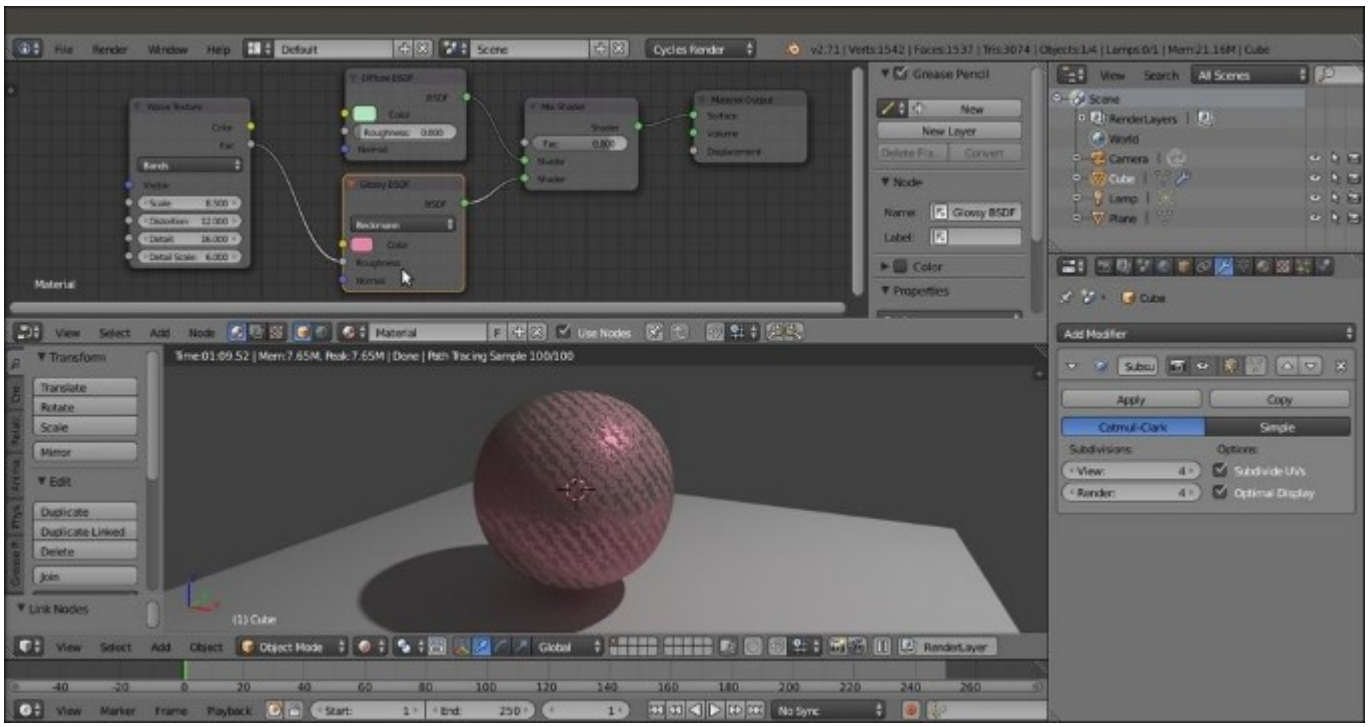
The Rendered preview of the effect of a Wave texture assigned as color to the diffuse component of the material

4. In the **Wave Texture** node, change the **Scale** value to `8.500`, **Distortion** to `12.000`, **Detail** to a maximum value of `16.000`, and the **Detail Scale** value to `6.000`.
5. Now disconnect the texture color output from the **Diffuse** node and connect it to the color input socket of the **Glossy** shader, as shown in the following screenshot:



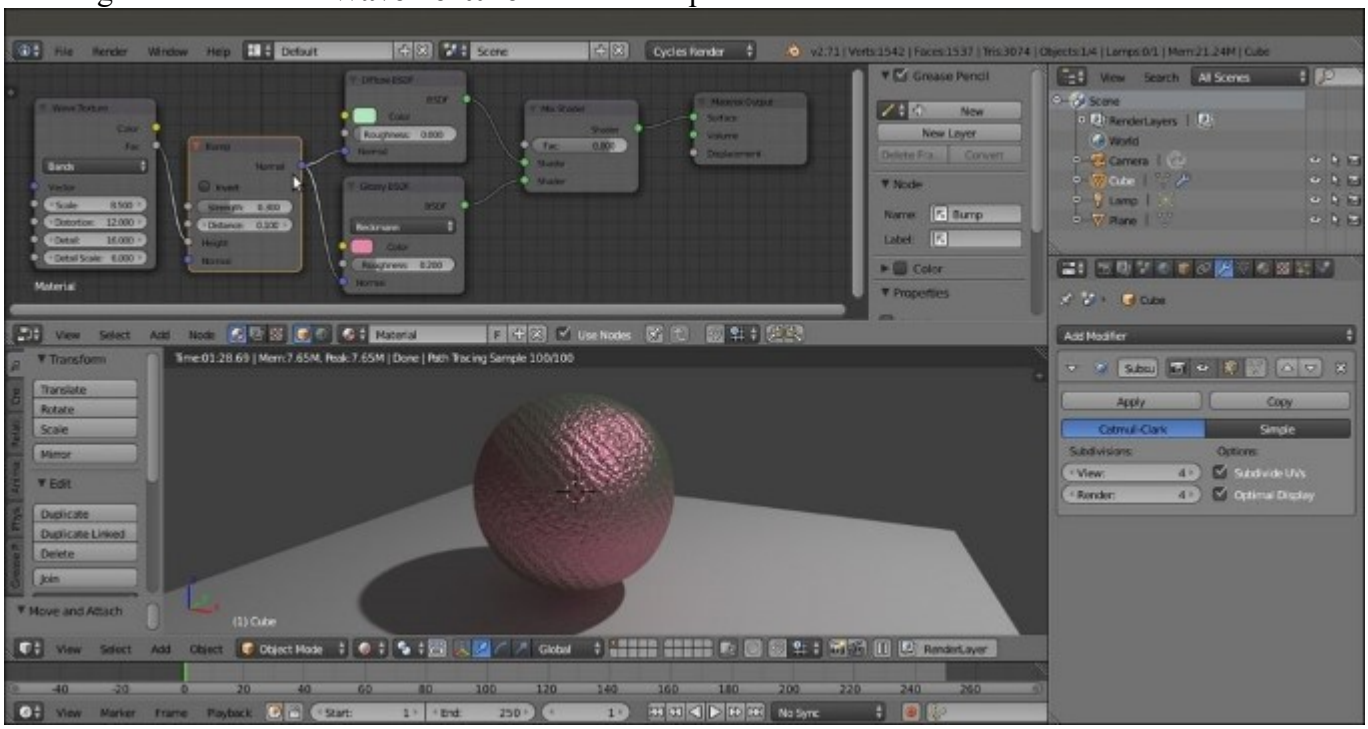
The effect of the Wave Texture assigned as color to the Glossy component of the material

6. Disconnect the texture color output from the **Glossy** shader. Grab and connect the texture node's **Fac** output to the **Roughness** input socket of the **Glossy BSDF** shader, as shown in this screenshot:



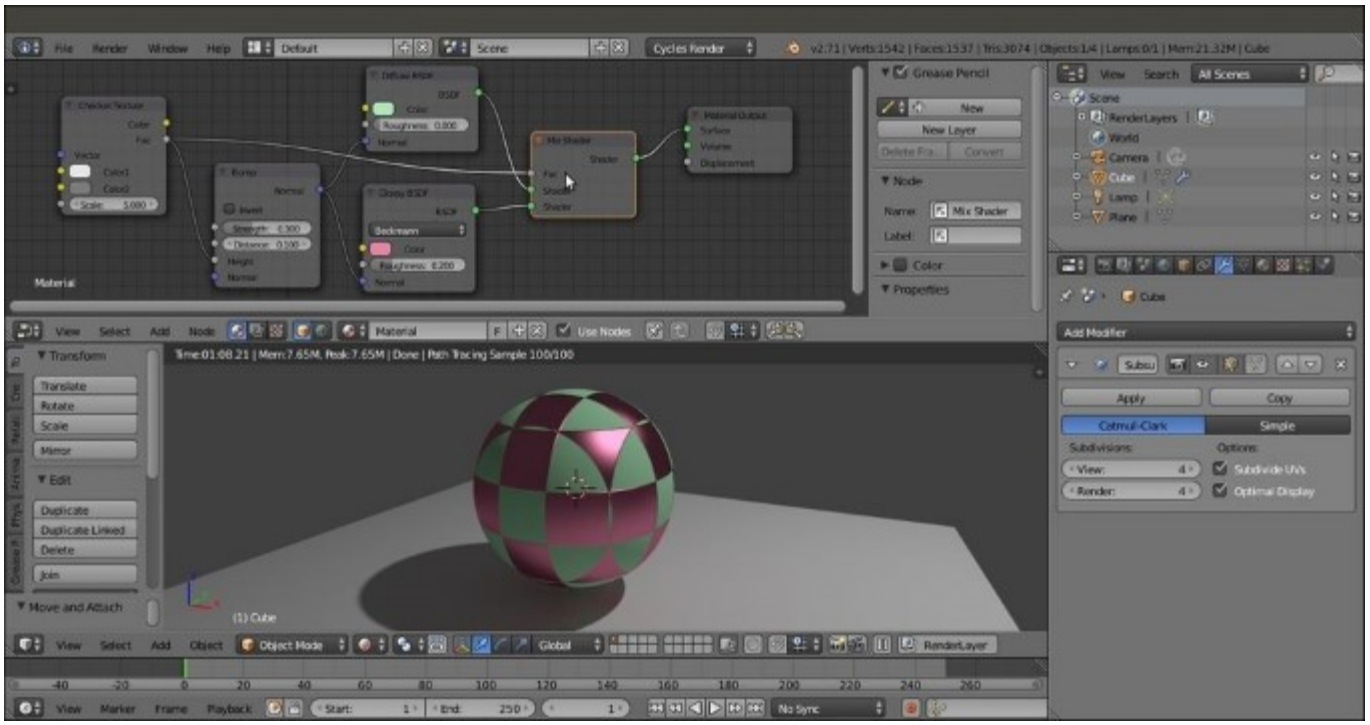
The effect of the Wave Texture assigned as Roughness factor to the Glossy component of the material

7. Disconnect the texture color output from the **Roughness** input socket of the **Glossy BSDF** shader. Move the **Wave Texture** node to the left and add a **Bump** node (*Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Fac** output of the **Wave Texture** node to the **Height** input node of the **Bump** node, and the **Normal** output of the **Bump** node to the **Normal** input socket of both the **Diffuse** and the **Glossy** nodes. Set the **Strength** to `0.300`. Here is a screenshot showing the effect of the **Wave Texture** node as bump:



The effect of the Wave Texture Fac output as Bump for both the components of the material

8. Save the file.
9. Delete the **Wave Texture** node (*X* key), press *Shift* + *A* with the mouse pointer in the **Node Editor** window, and add a **Checker Texture** node.
10. Connect the **Fac** output of the **Checker Texture** node to the **Fac** input socket of the **Mix Shader** node and to the **Height** input socket of the **Bump** node, as shown in the following screenshot:



The effect of a Checker Texture used as bump and especially as blending factor to mix the two components of the material

11. Save the file as `start_03.blend`.

How it works...

From step 1 to 3, the changes are immediately visible in the **Rendered** viewport. At the moment, the **Wave Texture** node color output is connected to the color input of the **Diffuse BSDF** shader node, and the Spheroid looks as if it's painted in a series of black and white bands. Actually, the black and white bands of the texture node override the green color of the diffuse component of the shader, while keeping the material's pink glossy component unaltered.

In step 5, we did exactly the opposite. We disconnected the texture output from the **Diffuse** shader to connect it to the **Glossy** shader color input. Now we have the diffuse greenish color back and the pink has been overridden, while the reflection component is visible only inside the white bands of the wave texture.

In step 6, in addition to the color output, every texture node also has a **Fac** (factor) output socket, outputting gray-scale linear values. When connected to the **Roughness** input socket of the **Glossy** shader, the texture output works as a factor for its reflectivity. The Spheroid keeps its colors and gets the specular component only in the white areas on the surface (that is, white bands represent total reflection and black bands represent no reflection).

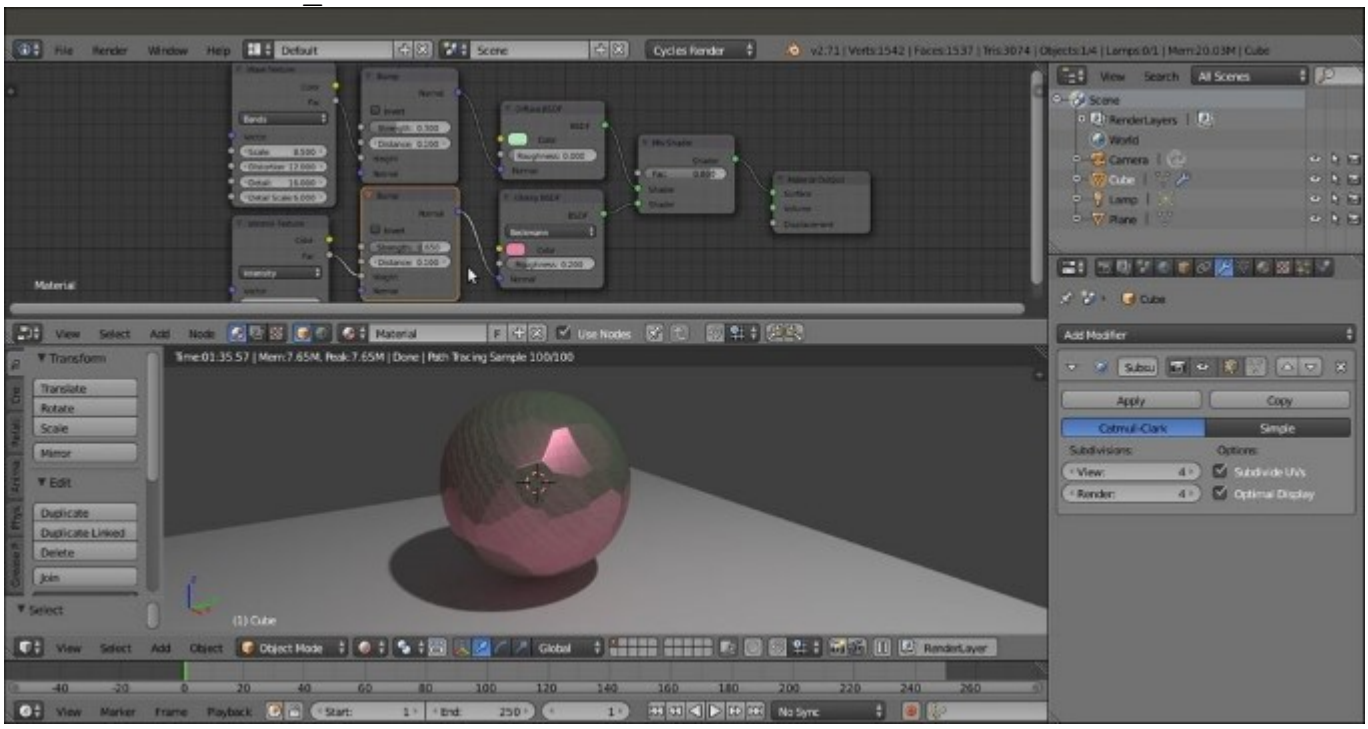
In step 10, the **Checker Texture** node's **Fac** output connected to the **Fac** input socket of the **Mix Shader** node works as a mask, or a stencil, based on the black and white values of the output. The numeric slider for the mixing factor on the **Mix Shader** node has disappeared because now we are using the black and white linear values of the **Checker Texture** output as a factor to mixing the **Diffuse** and **Glossy** components. Therefore, these components appear on the Spheroid surface according to the black and white quads of the checker.

Every texture node has several setting options. All of them have in common the **Scale** value to set the size of the procedural. The other settings change according to the type of texture.

The **Fac** output of the texture node can be used to feed the **Height** input socket of the **Bump** node (actually, the **Color** output also works quite well here). Hence, the **Normal** output of the **Bump** node can be connected to the **Normal** input sockets of each shader node, giving a per node bump effect. So, the bump can have an effect only on the diffuse component, or only on the glossy component, or on both, and so on.

Let's create an example of **Wave** and **Voronoi** textures:

1. Re-open the `start_02.blend` file.
2. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a new **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**).
3. Connect the **Fac** output of the **Voronoi Texture** node to the **Height** socket of the new **Bump** node, and connect the latter to the **Normal** input socket of the **Glossy BSDF** shader node. Set its **Strength** value to `0.650` and the **Voronoi** scale to `6.000`.
4. Save the file as `start_02bis.blend`.



Two different procedural textures, a Wave and a Voronoi, used as bumps for the two components to have a per shader effect

In this case, we have two different bump types, affecting the diffuse and the glossy components independently, and building an effect of a layered bump.

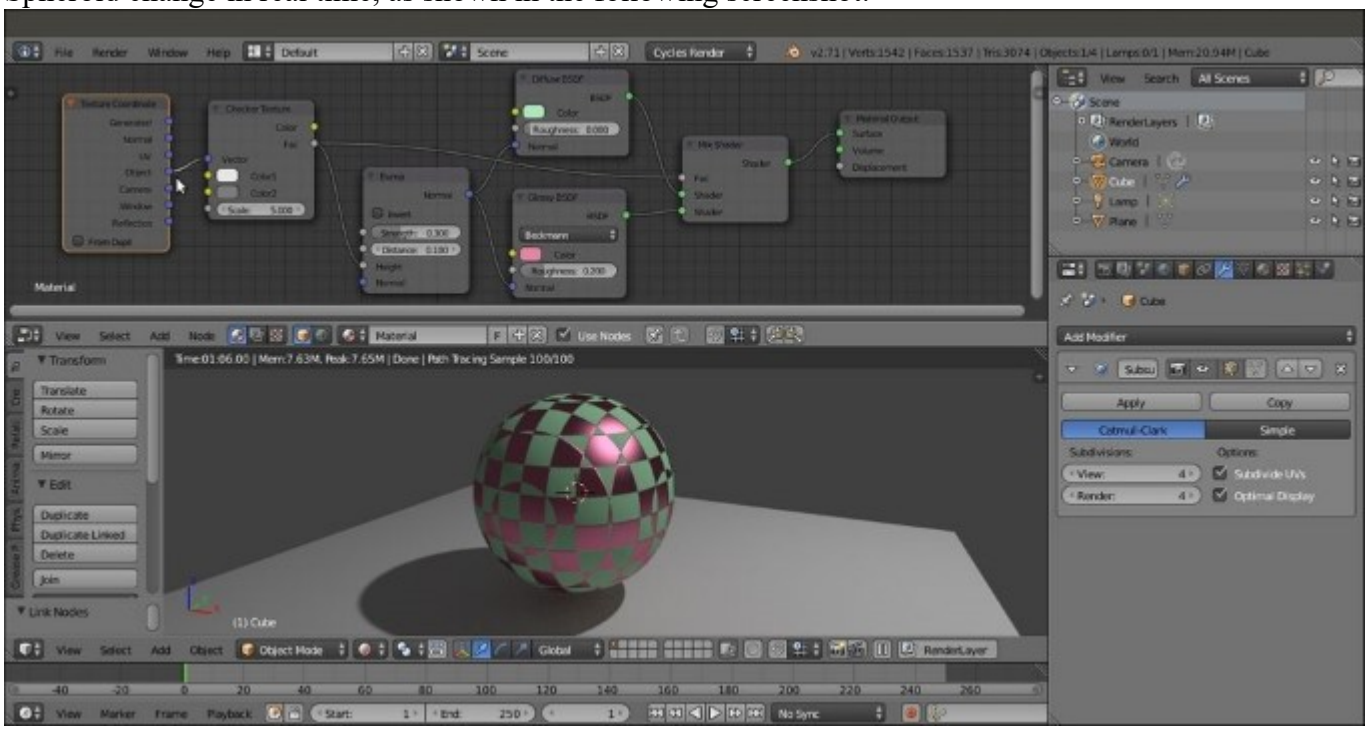
There's more...

At this point, you could wonder: "Okay, we just mapped textures on the Spheroid, but what's the projection mode of these mappings?"

Good question! By default, if the projection mode is not specified and if the object doesn't have any UV coordinates, the mapping is **Generated**, which is the equivalent of the **Original Coordinates** mode (now renamed **Generated** as well) in Blender Internal.

But what if you want to specify a mapping method? Then follow these steps:

1. Press *Shift* + *A* with the mouse pointer in the **Node Editor** window again, go to the **Input** item, and select the **Texture Coordinate** item, which is a node with several mapping modes and their respective output sockets.
2. Try to connect the several outputs to the **Vector** input (the blue socket on the left-hand side of the node), which can be found from **Checker Texture**, to see the texture mapping on the Spheroid change in real time, as shown in the following screenshot:



The Object output of the Texture Coordinate node connected to the Vector input of the Texture node

By the way, I'd like to point your attention to the UV coordinates output. Connect the link to the texture's vector socket, and you will see the mapping on the Spheroid disappear. Why is this so? Because we haven't assigned any UV coordinates to our Spheroid yet.

Go to the **UV Maps** tab in the **Object data** window, under the **Properties** panel on the right, and click on the **+** sign. This just adds a one-to-one **Reset UV projection UV** layer to the object, which means that every face of the mesh is covering the whole area of the **UV/Image Editor** window. Remember that although the Cube looks like a Spheroid now, this is only due to the effect of the assigned **Subdivision Surface** modifier. The UV coordinates work at the lowest level of subdivision, which is still a six-faced Cube.

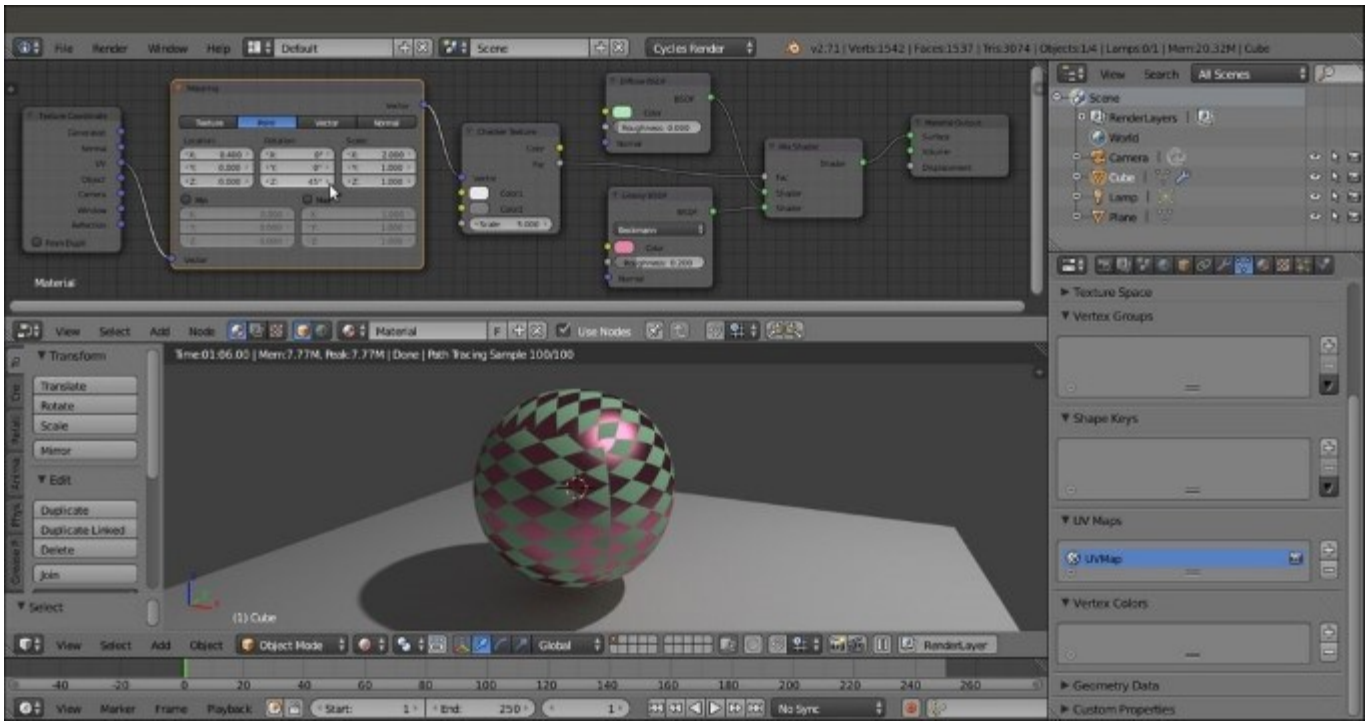
A second option is to place the proper seams on the Cube's edges and directly unwrap the object in the **UV/Image Editor** window, as demonstrated in the following steps:

1. Press *Tab* to go to **Edit Mode**, select the appropriate edges, press *Ctrl* + *E*, and in the **Edges** pop-up menu, select the **Mark Seam** item.
2. Now press *A* to select all the vertices (if deselected), press *U*, and choose an unwrapping method from the **UV Mapping** pop-up menu (**Smart UV Project** and **Cube Projection** don't even need the seams). Then go out of **Edit Mode** to update the **Rendered** preview.

The **Texture Coordinate** node is not mandatory to map an image texture on an unwrapped object; in such a case, Cycles will automatically use the (first) available UV coordinates to map the image map anyway.

Often, the only **Texture Coordinate** node is not enough. What we need now is a way to offset, rotate, and scale this texture on the surface:

1. First delete the **Bump** node, then select the **Texture Coordinate** node, and drag it to the left of the window as far as suffices to make room for a new node. In the **Add** menu, go to **Vector** and choose **Mapping**.
2. Grab the **Mapping** node in the middle of the link that connects the **Texture Coordinate** node to the **Checker Texture** node. It will be automatically pasted between them, as shown in the following screenshot:



The Mapping node pasted between Texture Coordinate and the Texture nodes

3. Now start playing with the values inside the **Mapping** node. For example, set the **Z Rotation** value to `45°`, set the **X Scale** value to `2.000`, and then slide the **X Location** value, while seeing, in the **Rendered** viewport, how the texture changes orientation and dimension and actually slide along the *x* axis.
4. Save the blend file as `start_04.blend`.

The **Min** and **Max** buttons on the bottom of the **Mapping** node are used to clip the extension of the texture mapping. Check both **Min** and **Max** to prevent the texture from being repeated *n* times on the surface, and it will be shown only once. A minimum value of `0.000` and a maximum value of `1.000` give a correspondence of one-to-one to the mapped image. You can tweak these values to limit or extend the clipping. This is useful to map decals, logos, or labels, for example, on an object and avoid repetition.

See also

In Cycles, it is possible to use normal maps by adding the **Normal Map** node (by navigating to **Add** | **Vector** | **Normal Map**) and connecting its output to the **Normal** input socket of the shader nodes.

To see an example of a **Normal Map** node used in a Cycles material, go to Chapter 8, *Creating Organic Materials*, of this cookbook and look at the `bark_seamless` material of the *Creating trees shaders – the bark* recipe.

Here is a link to the official documentation talking about the **Normal Map** node:

http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes/More

Setting the World material

In this recipe, we'll see the properties and the settings of the World in Cycles.

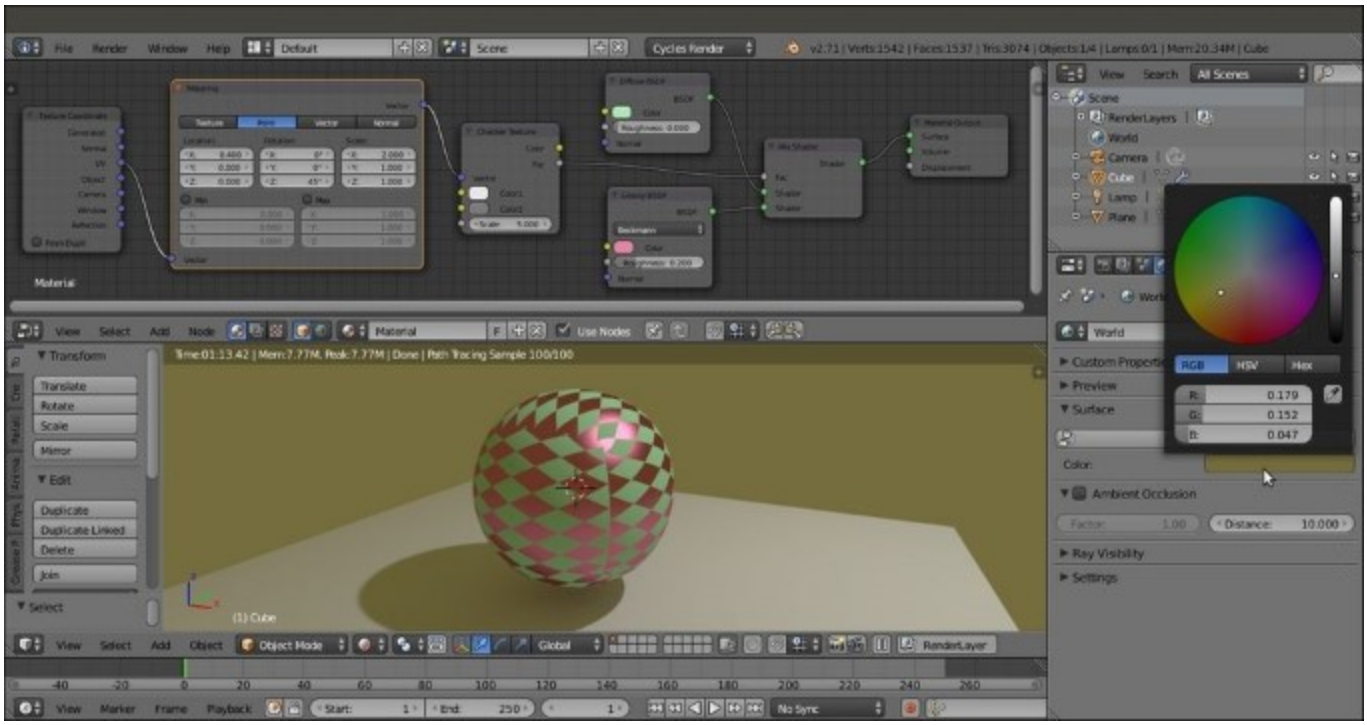
The main characteristic of the Cycles World is that it can emit light, so it practically behaves as a light source. Actually, its effect is the famous Global Illumination effect.

As in Blender Internal, the World is considered as a virtual dome at a large distance from the camera, never touching the scene's objects. Nothing in the 3D scene can affect the World. Actually, only the World can emit light on the scene and the objects.

Getting ready

1. Open the `start_04.blend` file and go to the **World** window under the **Properties** panel to the right of the screen. This is where we see the usual **Use Nodes** button under the **Surface** tab.
2. Although no node system for the **World** window is set by default, the **World** window already has a dark, medium gray color slightly lighting the scene. Delete the default Lamp or put it in a different and disabled layer to see that the Spheroid in the scene is dark but still visible in the rendered 3D viewport.
3. It's already possible to change this gray color to some other color by clicking on the **Color** button right under **Use Nodes** (the color at the horizon). This brings up the same color wheel that we saw for the shader colors. Set the color to `R 0.179`, `G 0.152`, and `B 0.047`, and save the file as `start_05.blend`.

Note that both the intensity and the general color graduation of the World are driven by this color. To have more light, just move the **Value** slider (the vertical slider) to a whiter hue. To give a general color mood to the scene, pick a color from inside the wheel. This will affect all of the scene's illumination but will show the effect mainly in the shadows, as shown in the following screenshot:

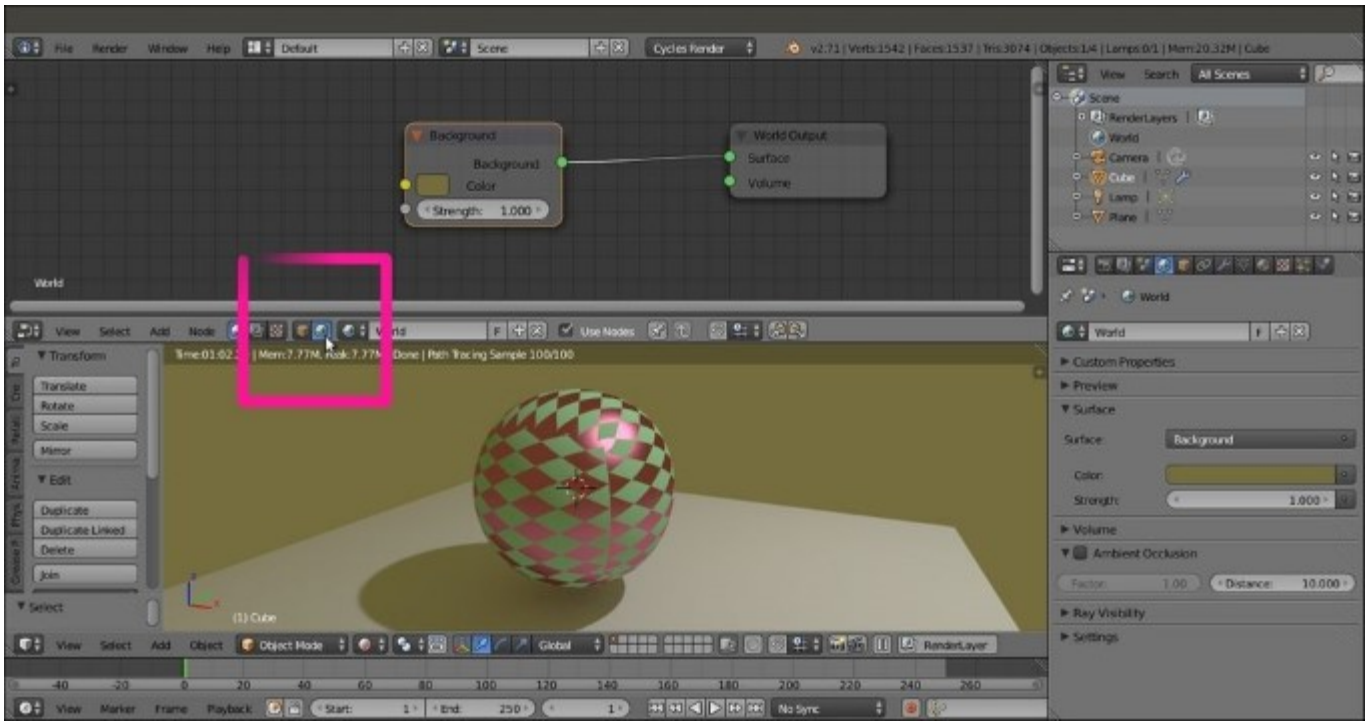


To the right is the color wheel to set the World's color, inside the World window, under the main Properties panel

How to do it...

However, to get access to all the options for the World, we have to initialize it as a node system, which is shown in the following steps:

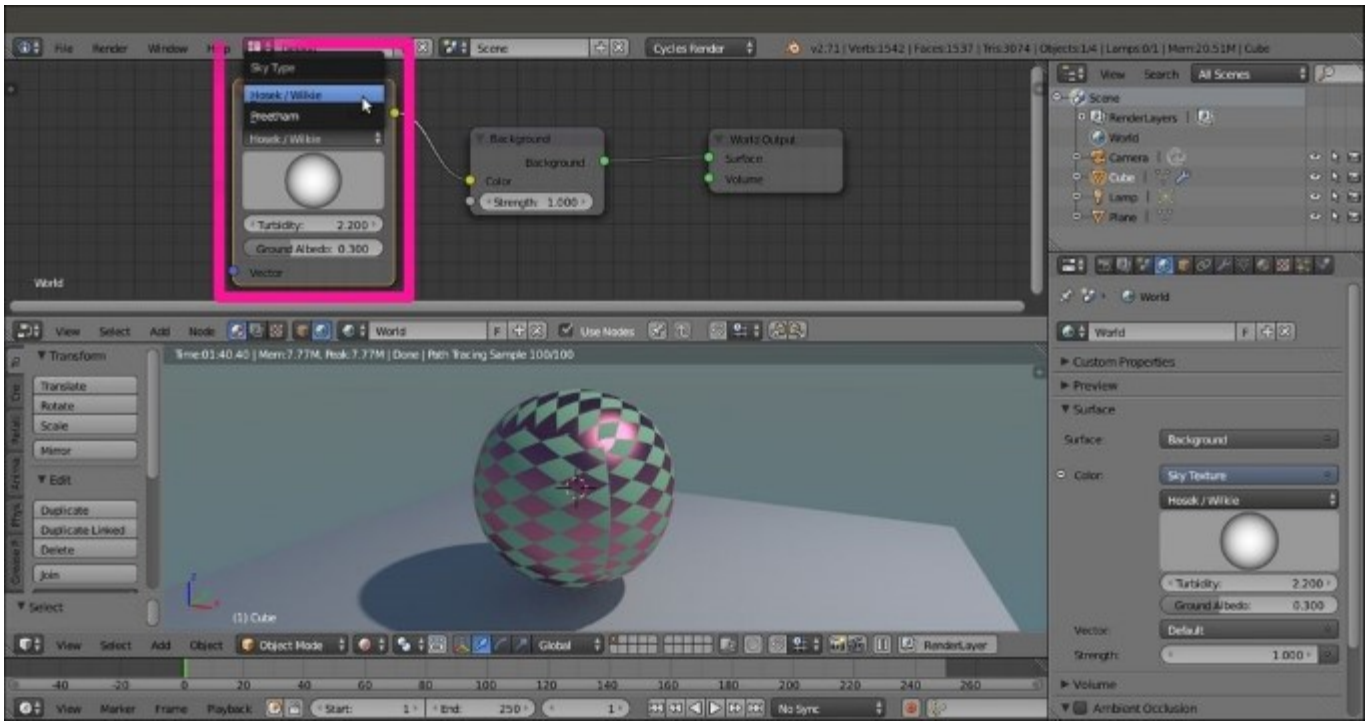
1. Look at the bottom header of the **Node Editor** window. On the left-hand side of the material data block, there are two little icons: a little cube and a little world. The cube icon is used to create materials, while the world icon is for the World. At the moment, because we were working on the Spheroid material, the cube icon is the one selected.
2. Click on the little world icon. The material's node disappears, and the **Node Editor** window is empty now because we entered the World mode. Check the little **Use Nodes** box on the right of the data block to make a default world material appear. Alternatively, go to the **World** window under the **Properties** panel and click on the **Use Nodes** button under the **Surface** tab. This is shown in the following screenshot:



The World button to be switched in the Node Editor toolbar

Just like the materials, the default material for the World is simply made up of two nodes. A **Background** node is connected to a **World Output** node. In the **Background** node, there are two setting options: the **Color** box and the **Strength** slider. Both of them are quite self-explanatory. Now, perform the following steps:

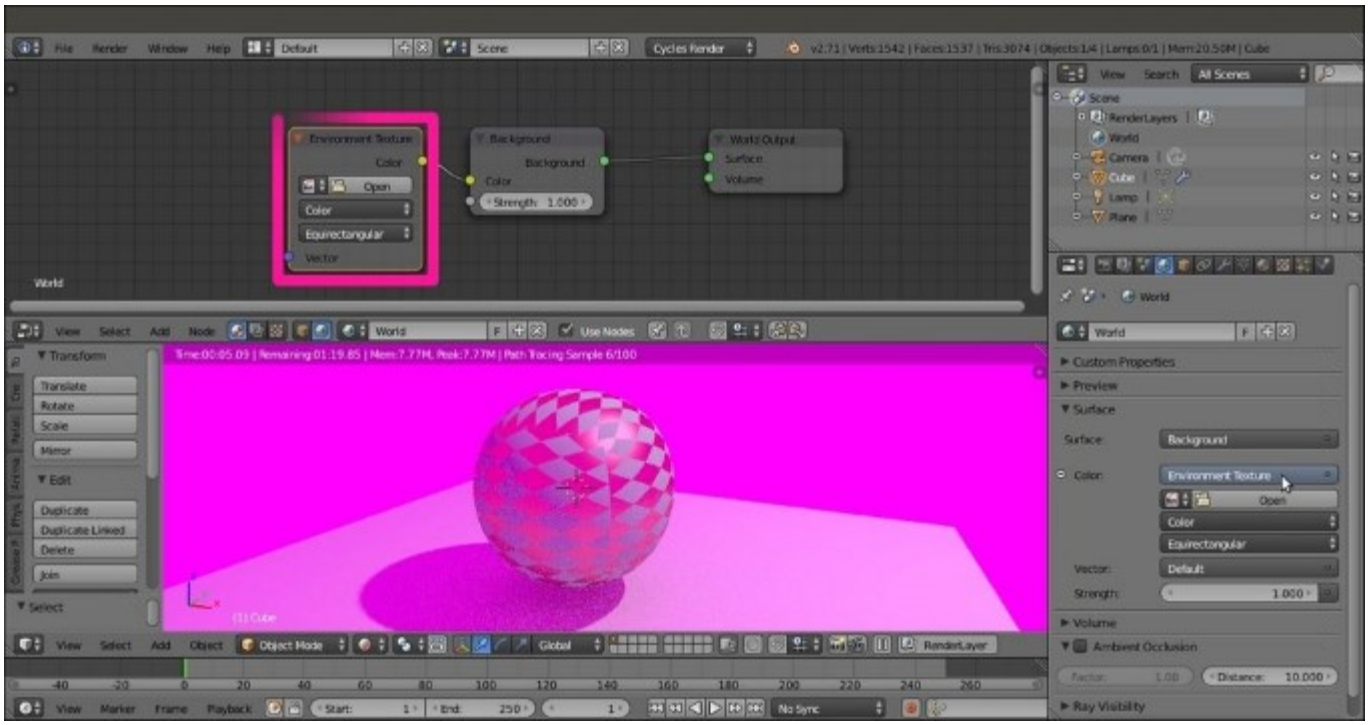
1. Go to the **World** window under the **Properties** panel, and click on the little square with a dot to the right side of the **Color** slot.
2. From the resulting menu, select the **Sky Texture** node item. This replicates a physical sky model with two **Sky** types, an atmospheric **Turbidity** value slider, a **Ground Albedo** value slider, and a **Strength** slider, as shown in this screenshot:



The Sky Texture node with options connected as Color to the Background node

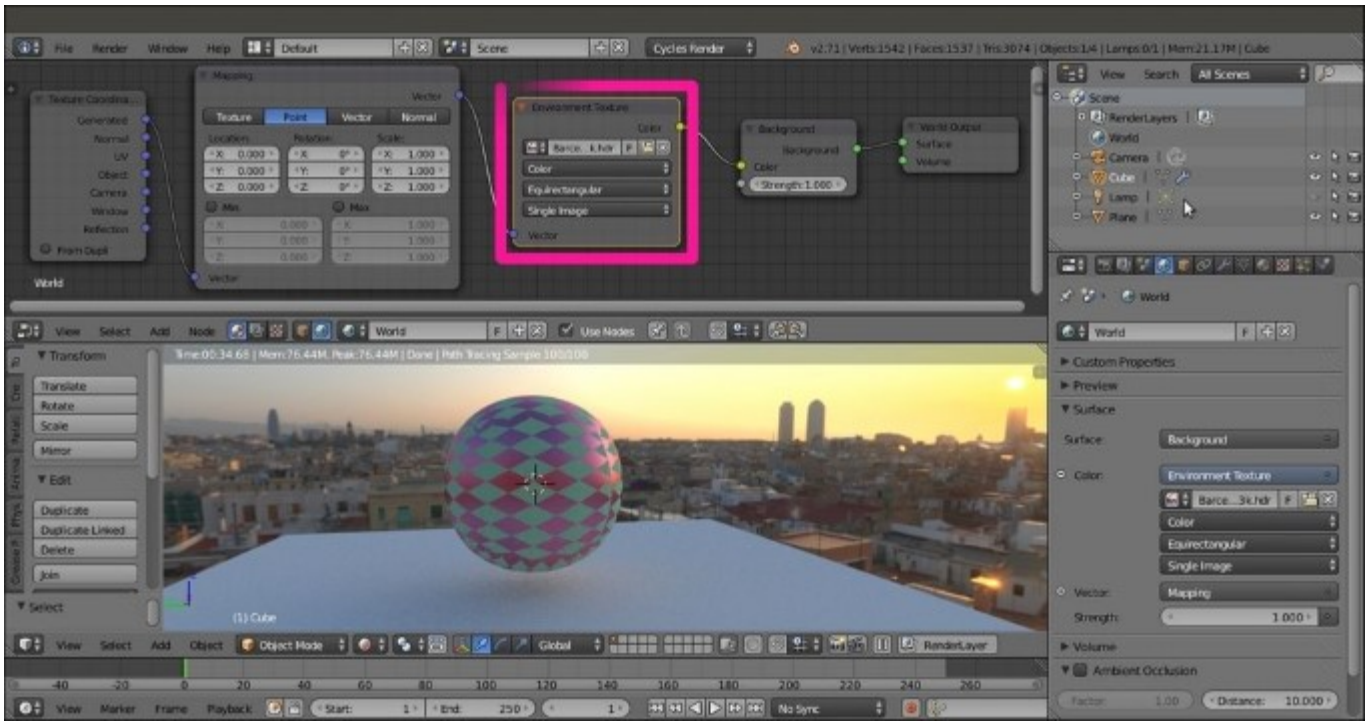
Note that you can also modify the incoming direction of the light, that is, the location of the sun, by rotating the sphere icon inside the node interface. This control isn't that much precise, by the way, and will hopefully improve in the future. The next steps are as follows:

1. Save the file as `start_06.blend`.
2. Click on the **Color** button, which is now labeled **Sky Texture**, under the **Surface** tab in the **Properties** panel, and select the **Environment Texture** node to replace it, as shown in the following screenshot:



The pink warning effect of a missing texture in the Environment Texture node of the World setting

3. Look in the **Rendered** view. You'll see that the general lighting has changed to a pink color. This is to show that the World material is now using an image texture to light the scene, but that there is no texture yet.
4. Click on the **Open** button in the **World** window, either under the **Properties** panel or in the recently added node inside the **Node Editor** window. Browse to the `textures` folder and load the `Barce_Rooftop_C_3k.hdr` image (a free, **High-dynamic-range** (**HDR**) image licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License from the sIBL Archive, at http://www.hdrlabs.com/sibl/archive.html).
5. To appreciate the effect, click on the little eye icon on the side of the **Lamp** item in the **Outliner** to disable its lighting. The Spheroid is now exclusively lit by the HDR image assigned to the World material. Actually, you can see the image as a background in the **Rendered** preview. You can also rotate the viewport and watch the background texture, pinned to the World coordinates, rotate accordingly in real time.
6. As for the object's materials, the mapping of any texture you are going to use for the World can be driven by the usual **Mapping** and **Texture Coordinates** nodes we have already seen. Generally, for the World materials, only the **Generated** coordinates output should be used, and actually, the **Generated** coordinates output is used by default if no mapping method is specified. Add the **Mapping** and **Texture Coordinates** nodes and connect them to the **Vector** input socket of the **Environment Texture** node, as shown in the following screenshot:

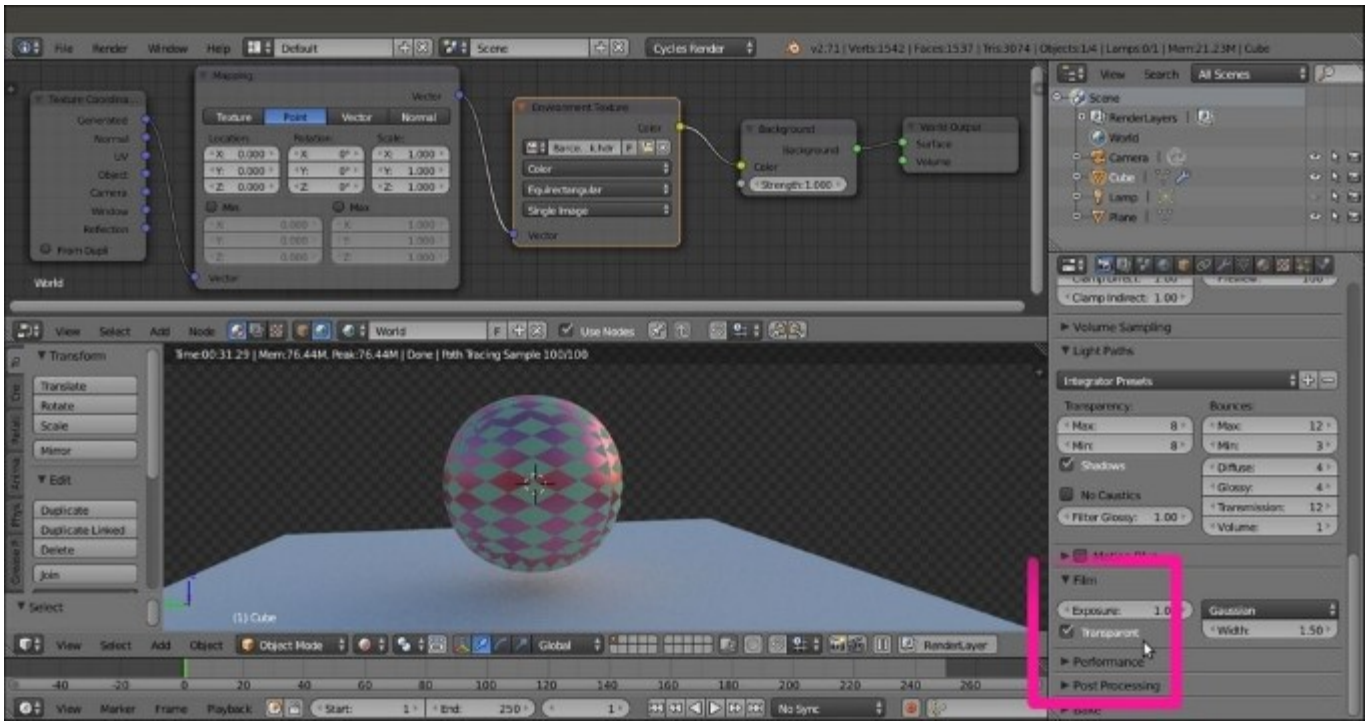


The Rendered preview of an HDR image assigned as a background to the World through the Environment Texture node

7. Save the file as `start_07.blend`.

Now let's imagine a case in which we want to assign a texture to the World material and use it for the general lighting of the scene, but we don't want it to show in the background of the render. In other words, we are using the HDR image to light the Spheroid and the Plane, but we want the two objects rendered on a uniform blue background; so how do we do it? This is how:

1. One way is to go to the **Render** window and check the **Transparent** option under the **Film** tab. This will show our Spheroid and Plane rendered in both the 3D viewport and the effective final rendered image on a transparent background, with a premultiplied alpha channel, as shown in the following screenshot:

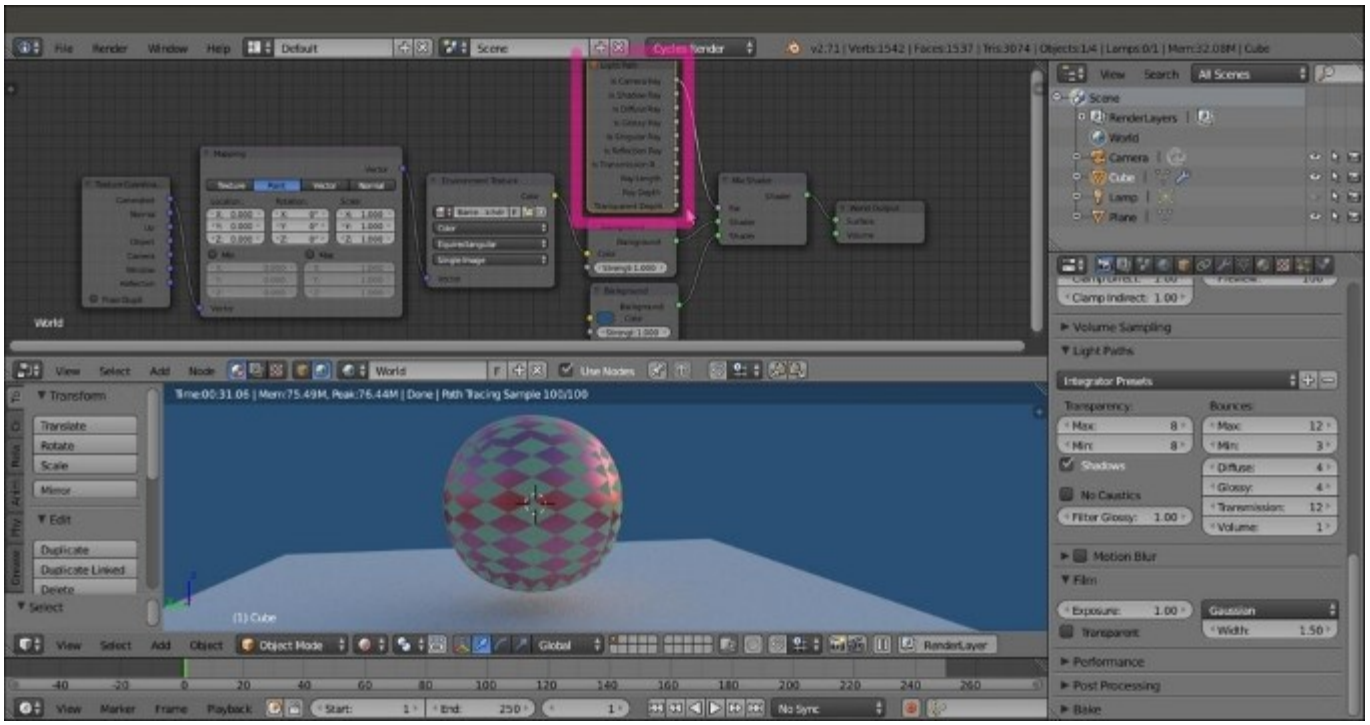


The previous World setting rendered with a transparent background

2. Now we can compose the rendered image with a blue background, both in external image editing software (such as GIMP, to stay inside FOSS) or directly in the Blender compositor.

A different way to render the two objects on a uniform blue background is to use a **Light Path** node:

1. If this is the case, deselect the **Transparent** item checkbox in the **Render** window to restore the sky background in the preview and in the rendering.
2. Click on the **World Output** node in the **Node Editor** window, press *G*, and move it to the right.
3. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** |**Mix Shader**) and move it to the link connecting the **Background** node to the **World Output** node, to paste it automatically between the two nodes.
4. Select the **Background** node in the **Node Editor** window. Press *Shift* + *D* to duplicate it and move it down.
5. Connect its output to the second input socket of the **Mix Shader** node. Click on its **Color** box to change the color to `R 0.023`, `G 0.083`, and `B 0.179`.
6. Now, add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**).
7. Connect the **Is Camera Ray** output of the **Light Path** node to the **Fac** input socket of the **Mix Shader** node, and voilà! The objects in the scene are lit by the HDR image connected to the first **Background** node, but they appear in a sky that is colored as set in the **Color** box of the second **Background** node. This is shown in the following screenshot:



The use of the Path Light node as a factor to have a different background than the HDR image still illuminating the scene

8. Save the file as `start_08.blend`.

How it works...

To explain this trick better, let's say we just created two different world materials: the first material with the texture and the second material with a plain blue color (this is not literally true; actually, the material is just one, containing the nodes of two ideally different worlds).

We mixed these two materials using the **Mix Shader** node. The upper green socket of the **Mix Shader** node is considered equal to a value of `0.000`, while the bottom green socket is considered equal to a value of `1.000`. As the name suggests, the **Light Path** node can set the path for the rays of light that are shot from the camera, if you remember. **Is Camera Ray** means that only the rays directly shot from the camera have a value of `1.000`, that is, not the reflected ones, or the transmitted ones, or whatever, which have a value of `0.000`.

Thus, because the textured world is connected to a socket equal to the value of `0.000`, we don't see it directly as a background, but only see its effect on the objects lit from the reflected light or from the HDR image. The World of the blue sky, which is connected to the input socket of value `1.000` instead, is seen as a background because the light rays shot from the camera directly hit the sky.

There's more...

Just after the **Surface** subpanel, in the **World** window, there is the **Ambient Occlusion** subpanel. **Ambient occlusion** is a lighting method used to emphasize the shapes or the details of a surface, based on how much a point on that surface is occluded by the nearby surfaces. Ambient occlusion can replace the Global Illumination effect in some cases, though not the same. For example, to render interiors with fast and noise-free results, ambient occlusion is a cheap way to get an effect that looks a bit like indirect lighting.

There is a checkbox to enable **Ambient Occlusion**, along with the following sliders:

- **Factor**: This is used for the strength of the ambient occlusion. A value of `1.00` is equivalent to a white World.
- **Distance**: This is the distance from a shading point to the trace rays. A shorter distance emphasizes nearby features, while a longer distance takes into account objects that are further away.

The **Ambient Occlusion** feature is only applied to the **Diffuse BSDF** component of a material. The **Glossy** or **Transmission BSDF** components are not affected. Instead, the transparency of a surface is taken into account. For example, a half-transparent surface will only half-occlude other surfaces.

Creating a mesh-light material

In this recipe, we will see how to create a mesh-light material to be assigned to any mesh object and used as a source to light the scene.

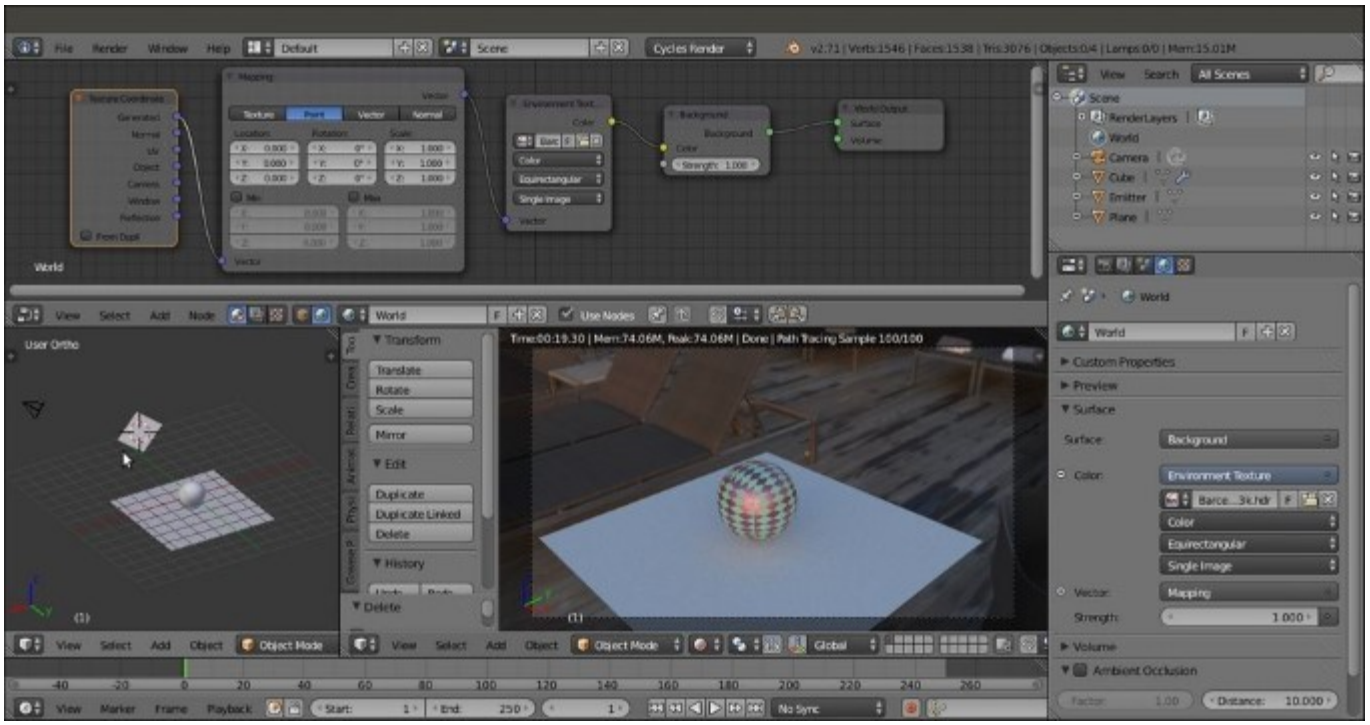
Getting ready

Until now, we have used the default Lamp (a Point light) already present in the scene to light the scene. By enabling the node system for the Lamp, we have seen that it uses a material created by connecting an **Emission** node to the **Lamp Output** node.

The good news is that just because it's a material node, we can assign an **Emission** shader to a mesh, for example, to a Plane conveniently located, scaled, and rotated to point to the scene that is the center of interest. Such a light-emitting mesh is called a mesh-light. Being a mesh, the **Emission** shader node output must be connected to the **Surface** (or the **Volume**) input socket of a **Material Output** node instead of the **Lamp Output** node.

Light emission coming from a surface and not from a point is a lot more diffused and softer than the light from a Lamp. A mesh-light can be any mesh of any shape, so it can be used as an object taking part in the scene and be the real light source of the rendering at the same time, for example, a table lamp, or a neon sign, or a television screen. As a pure light-emitting Plane, it's usually used as a sort of photographic diffuser. Two or three strategically placed mesh-lights can realistically simulate a photo studio situation. To replace the Lamp with a mesh-light, Plane perform the following steps:

1. Call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*), navigate to the **Addons** tab, and click on **3D View** under **Categories** on the left. Check the **Copy Attributes Menu** box to the right-hand side of the **3D View** option, and click on the **Save User Settings** button in the bottom-left corner of the panel. Then close the panel.
2. Starting from the `start_07.blend` file, click on the eye icon of **Lamp** in the **Outliner** to enable its visibility again.
3. Right-click on the **Lamp** in the 3D view and press *Shift* + *S* to bring up the **Snap** menu. Click on the **Cursor to Selected** item.
4. Press *Shift* + *A* with the mouse pointer in the 3D view and add a Plane to the scene at the 3D Cursor's location.
5. Press *Shift* and select the Lamp. Now you have both the recently added Plane and the Lamp selected, and the latter is the active object.
6. Press *Ctrl* + *C* to open the **Copy Attributes** menu and select the **Copy Rotation** item.
7. Rename this Plane as `Emitter`.
8. Right-click on the Lamp in the 3D view and press *X* to delete it.
9. Put the mouse pointer on the 3D view and press *0* from the numeric keypad to go to Camera view.
10. From the **Viewport Shading** menu in the window's header, select the **Rendered** mode (or put the mouse cursor on the **Camera** view and press *Shift* + *Z*):



A Plane set as a mesh-light to replace the Lamp, and the previous HDR image as the background

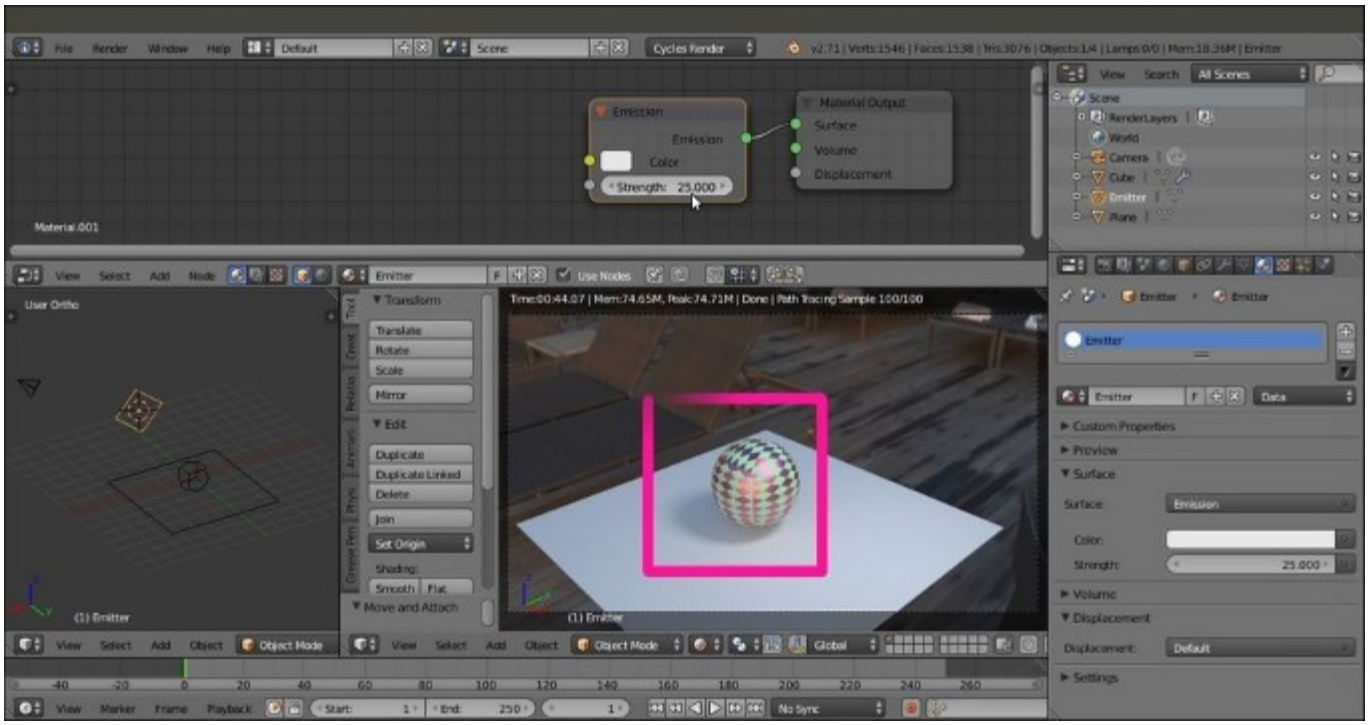
11. Save the file as `start_09.blend`.

How to do it...

Now let's create the emission material and also take a look at the setup for the softness of the projected shadows:

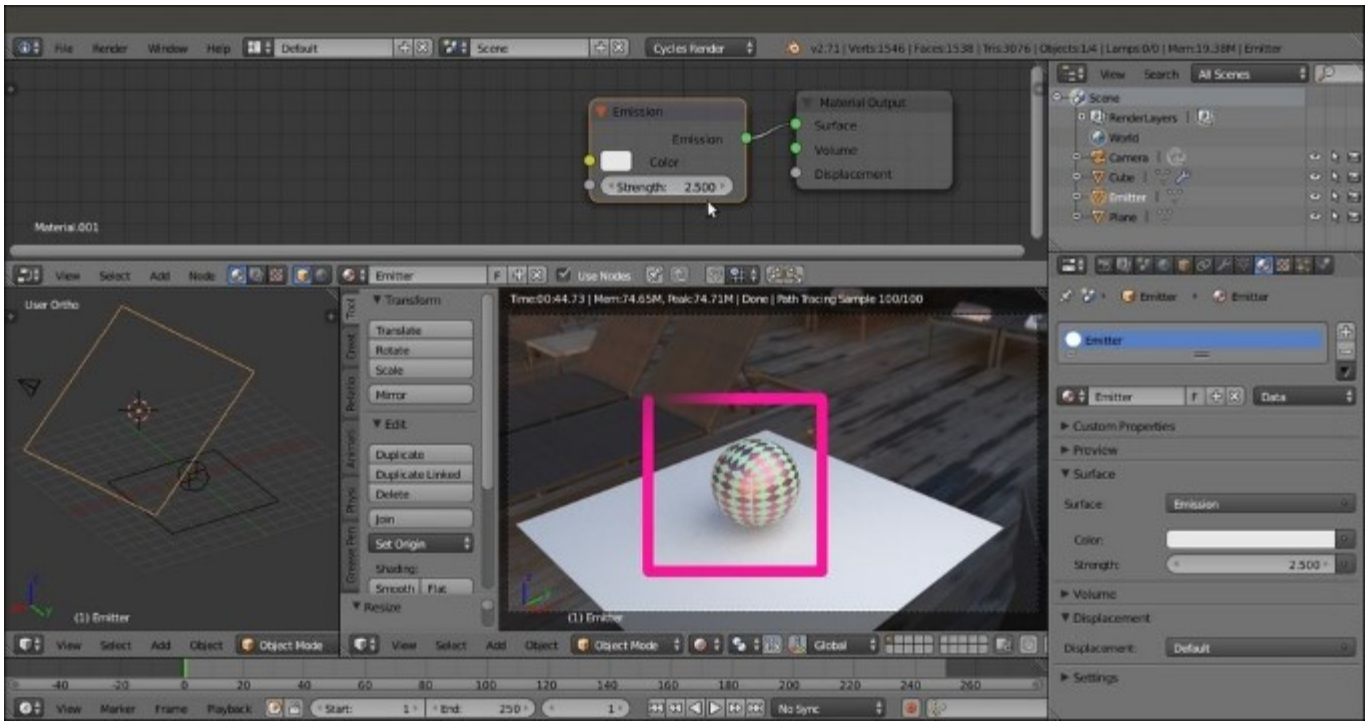
1. Select the **Emitter** plane and click on the little cube icon on the header of the **Node Editor** window.
2. Click on the **New** button in the header and rename the material as `Emitter`.
3. In the **Properties** panel, go to the **Material** window, and under the **Surface** tab, click on the **Surface** button to switch the **Diffuse BSDF** shader with an **Emission** shader. Leave the default color unchanged (**RGB** `0.800`) and set the **Strength** slider to `25.000`.
4. Save the file.

   The situation so far is as follows:



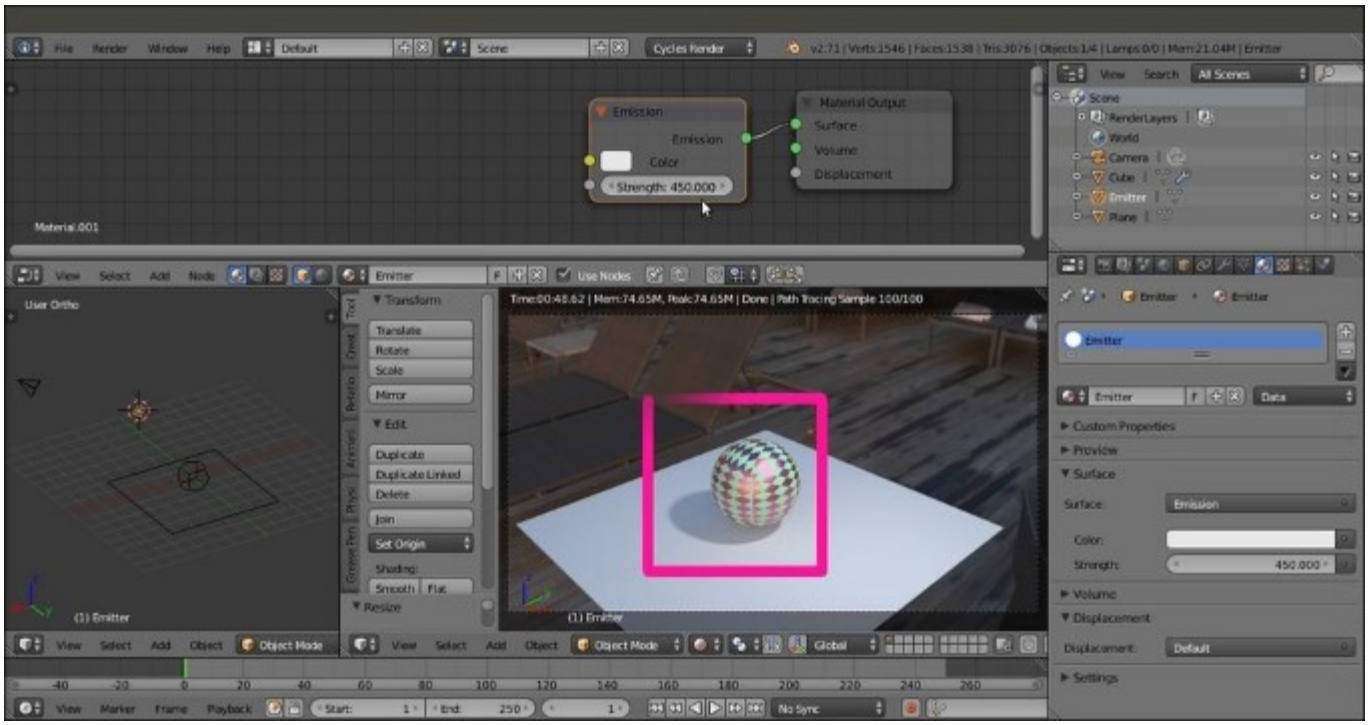
The mesh-light emission material with increased strength

5. In the 3D view, scale the **Emitter** plane five times bigger (press *S*, then enter *5*, and press *Enter*), and then set the **Strength** slider to `2.500`.
6. Save the file as `start_10.blend`. Now look at the softer shadow, as shown in the following screenshot:



Scaling the mesh-light bigger and decreasing the emission strength to have softer shadows

7. Now let's scale the **Emitter** plane a lot smaller (press *S*, then type *0.05*, and press *Enter*) and set the **Strength** slider to `450.000`.
8. Save the file as `start_11.blend`. Look at the crisper shadow in the **Rendered** preview, as shown in this screenshot:



Scaling the mesh-light smaller and increasing the emission strength to have crisper shadows

How it works...

From steps 5 to 7, we saw how a mesh-light can be scaled bigger or smaller to obtain a softer (in the first case) or a sharper (in the second case) shadow, respectively. The **Strength** value must be adjusted for the light intensity to remain consistent, or the mesh-light must be moved closer or more distant from the scene.

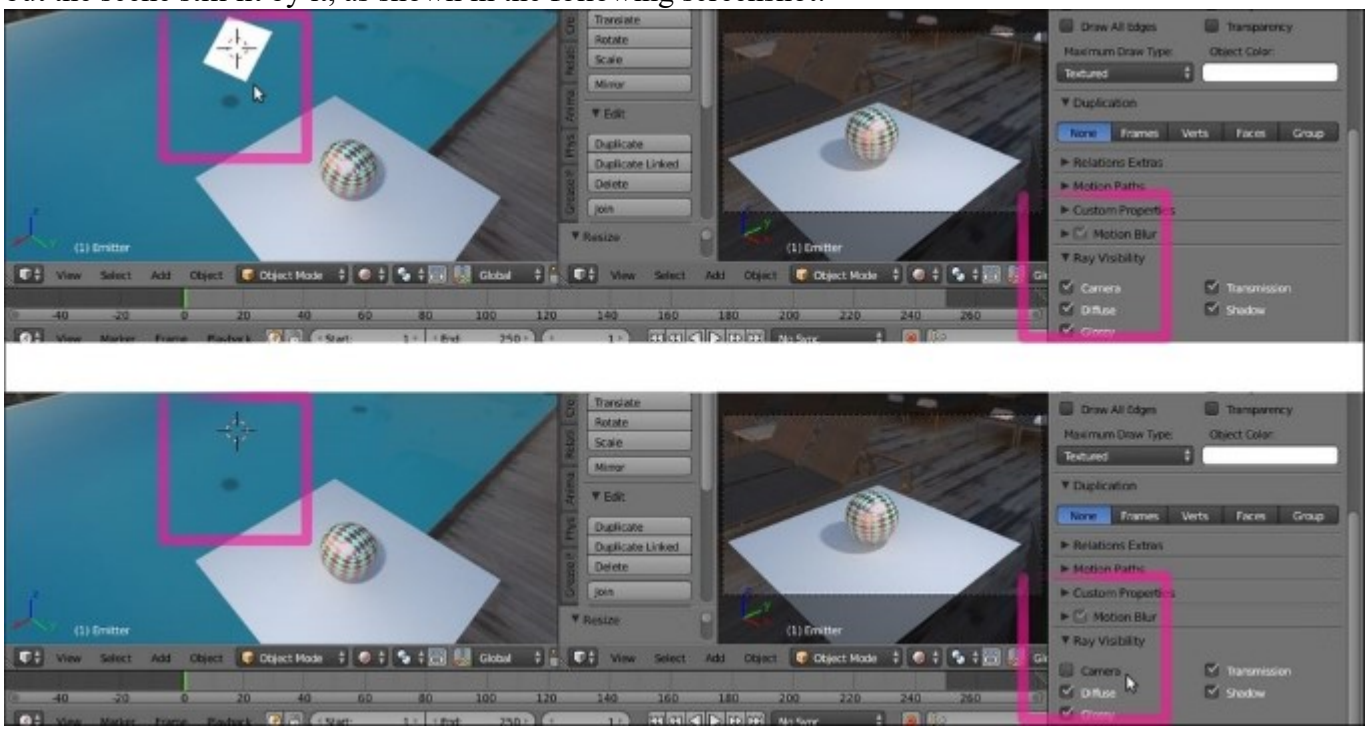
Scaling the mesh-light is basically the same as setting the size value for a Lamp. For Lamps, the softness of shadows can be set by the **Size** value to the left of the **Cast Shadow** option in the **Lamp** window, under the **Properties** panel (by default, the **Size** value is set to `1.000`). At a value of `0.000`, the shadow is at its maximum crispness, or sharpness. If the **Size** value is increased, the softness of the shadow increases too.

Unlike the mesh-light, varying the **Size** value of a Lamp doesn't require us to adjust the **Strength** value to keep the same light intensity.

There's more...

In several cases, you might not want the emitters to appear in your rendering. There are node arrangements to accomplish this (such as using the **Light Path** node in a way quite similar to the *Setting the World material* recipe we have seen before), but the easiest way to do this is as follows:

1. Start with the last saved blend (`start_11.blend`) and put the mouse cursor on the orthogonal 3D view to the left of the screen. Press the *3* key to navigate to the **Side** view. Then press *Shift* + *Z* to go in the **Rendered** mode to also see the **Emitter** plane rendered (be warned that if your computer can't easily render two windows at the same time, you must temporarily turn off the rendering for the **Camera** view).
2. With the Emitter plane still selected, navigate to the **Object** window under the **Properties** panel.
3. Look at the **Ray Visibility** tab (usually at the bottom of the **Properties** panel), where there are five items: **Camera**, **Diffuse**, **Glossy**, **Transmission** and **Shadows**, with the corresponding checked boxes.
4. Uncheck the **Camera** item and watch the **Emitter** plane disappear in the rendered 3D window, but the scene still lit by it, as shown in the following screenshot:

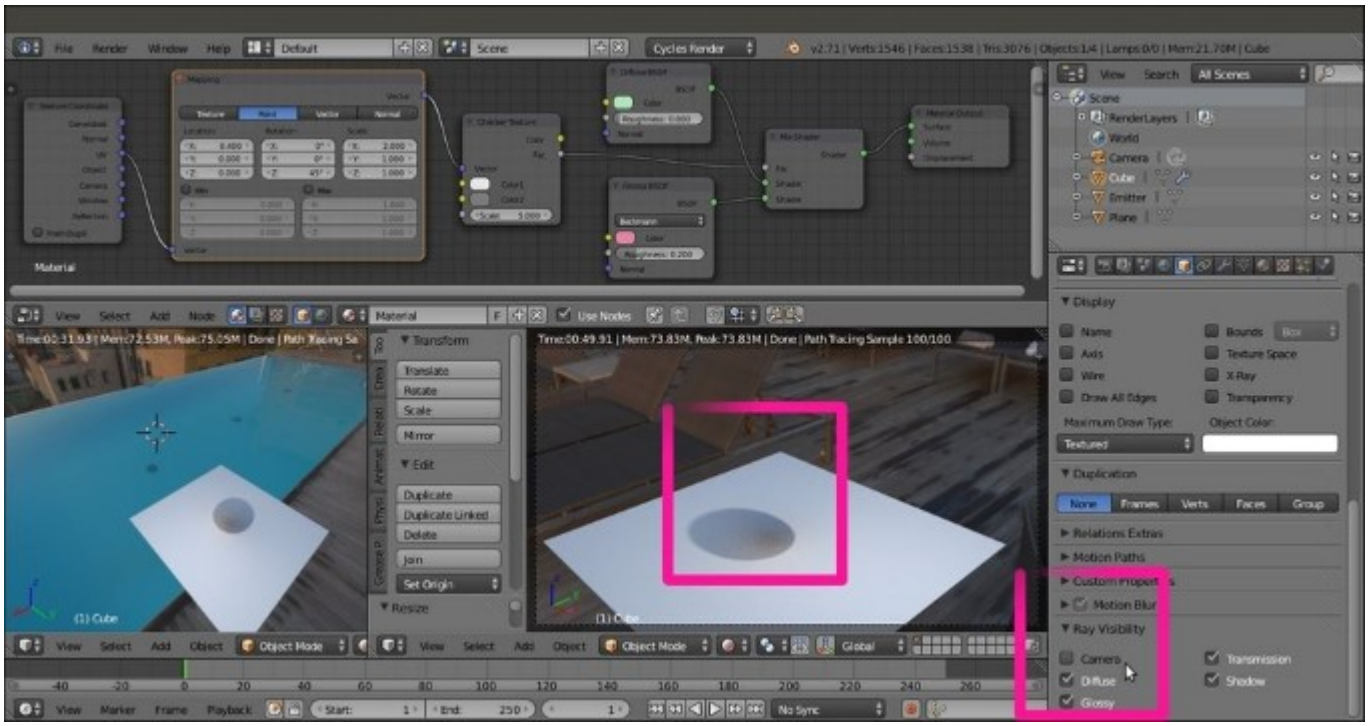


Disabling the Camera item in the Ray Visibility subpanel to hide the mesh-light Plane from the rendering

When you disable any one of the items, the corresponding property won't take part in the rendering. In our case, when the **Camera** box is unchecked, the mesh-light won't be rendered but it will still emit light. Be careful that the **Emitter** plane is not renderable at this moment, but because all the other items in the tab are still checked, it can be reflected and could cast its own shadow on other objects.

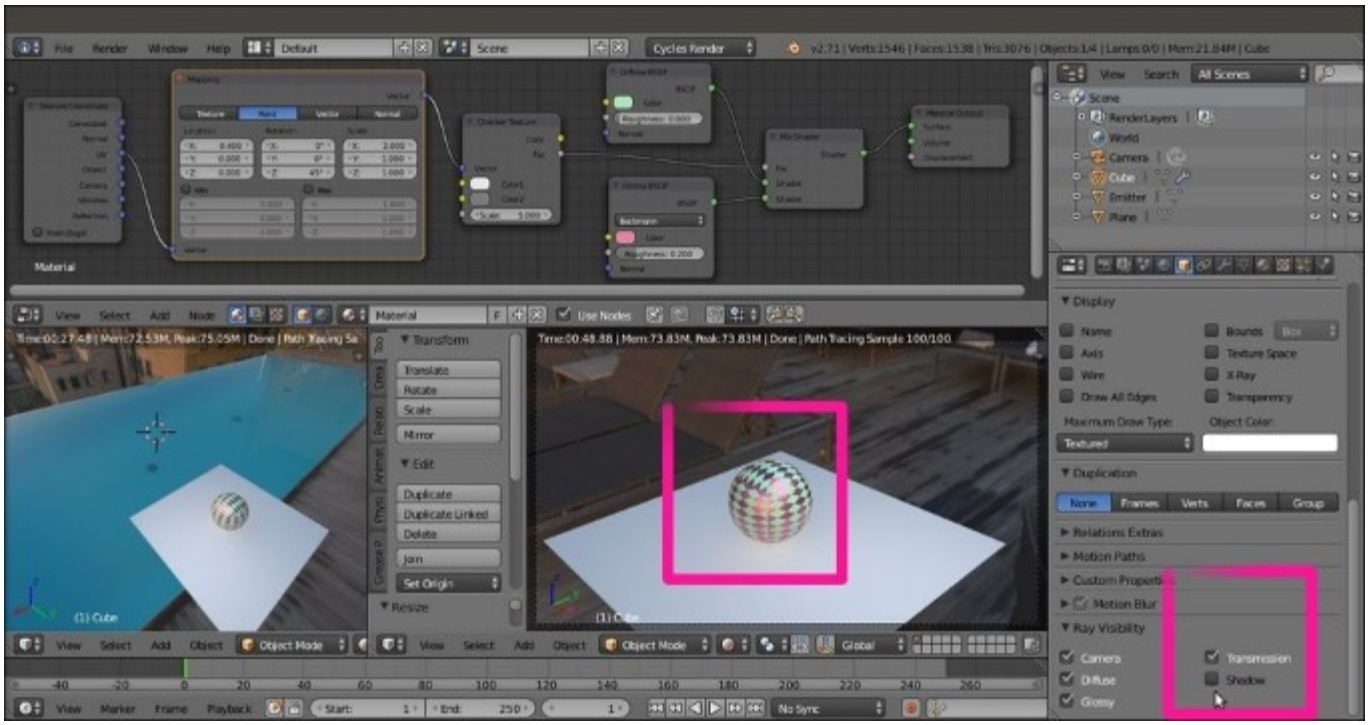
5. Now reselect the Spheroid (remember that unless you have renamed it, its name in the **Outliner** remains as Cube). Next, from the **Ray Visibility** tab in the **Object** window under the **Properties** panel, uncheck the **Camera** item.

Now the Spheroid has disappeared, but it's still casting its shadow on the floor Plane, as shown in this screenshot:



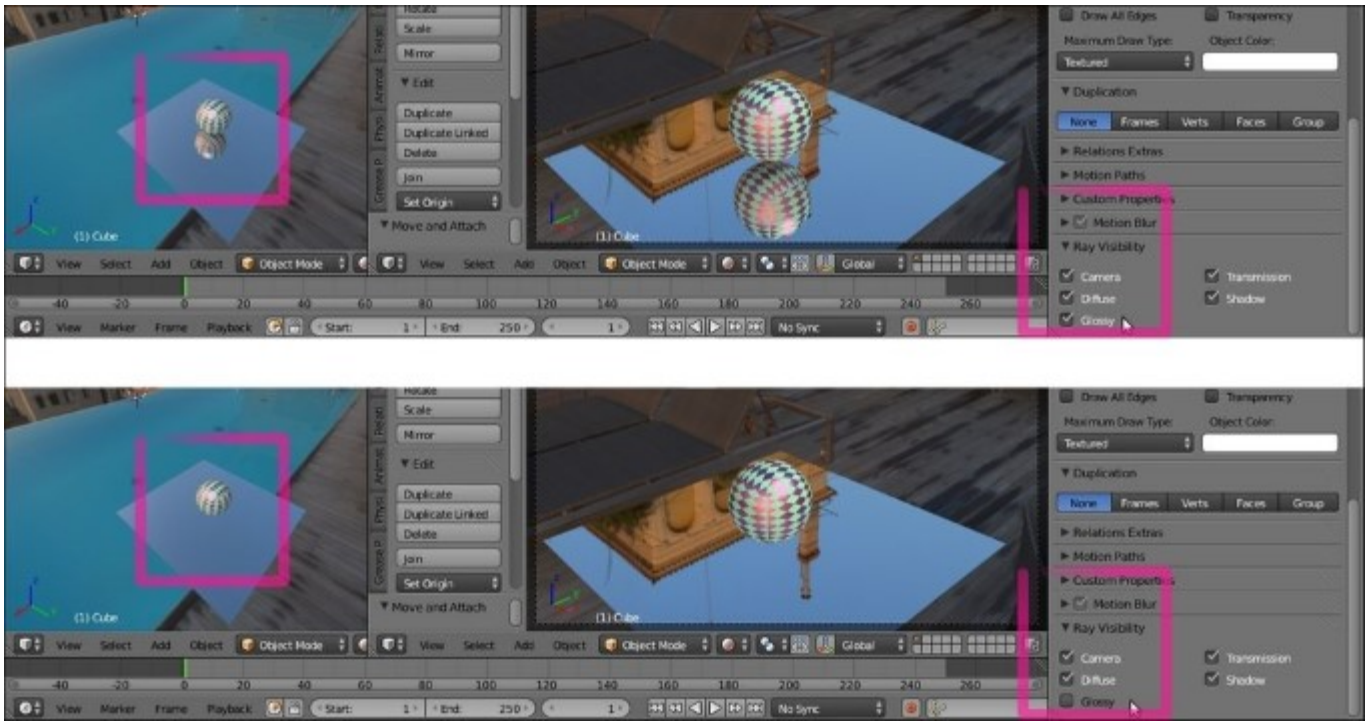
Disabling the Camera item to hide the Spheroid object from the rendering (but keeping the shadows on the floor)

6. Now check the **Camera** item again and uncheck the **Shadow** box. In this case, the Spheroid is visible again but doesn't cast a shadow, as shown in the following screenshot:



Disabling the Shadow item to have the Spheroid object rendered but without the shadows on the floor Plane

7. Save the file as `start_12.blend`. Let's try tweaking this a little.
8. Check the **Shadow** box for the Spheroid again, and select the floor Plane. Go to the **Material** window under the **Properties** panel, and click on the **New** button to assign a new material (**Material.001**).
9. Still in the **Material** window under the **Properties** panel, switch the **Diffuse BSDF** shader with a **Glossy BSDF** shader. The floor Plane is now acting as a perfect mirror, reflecting the Spheroid and the HDR image we formerly set in the World material.
10. Go back to the **Object** window and reselect the Spheroid. In the **Ray Visibility** tab, uncheck the **Glossy** item and watch the Spheroid, which is still rendered but not reflected by the mirror floor Plane, as shown in the following screenshot:



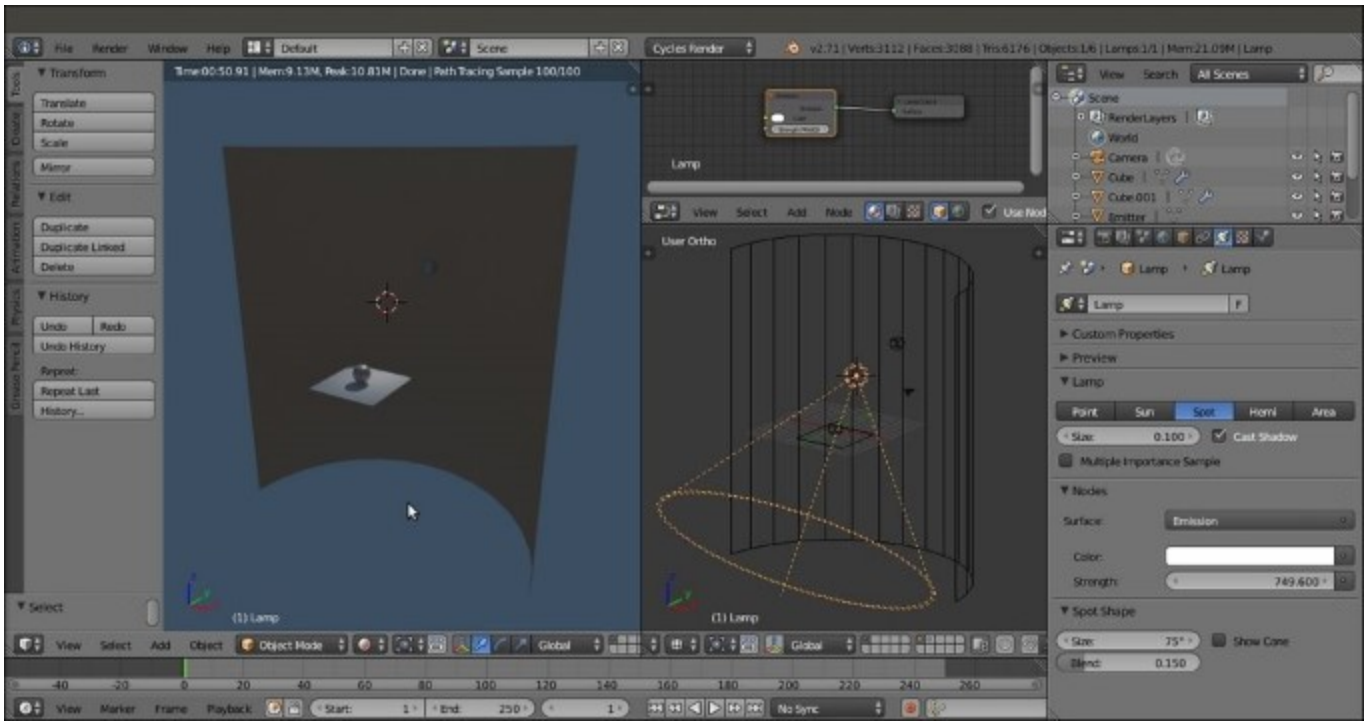
By disabling the Glossy item, we have the Spheroid object not mirrored by the glossy floor Plane

11. Save the file as `start_13.blend`.

Of course, the **Ray Visibility** trick we've just seen is not needed for Lamps because a Lamp cannot be rendered at all. At the moment, only **Point**, **Spot**, **Area**, and **Sun** lamps are supported inside Cycles. **Hemi** lamps are rendered as **Sun** lamps.

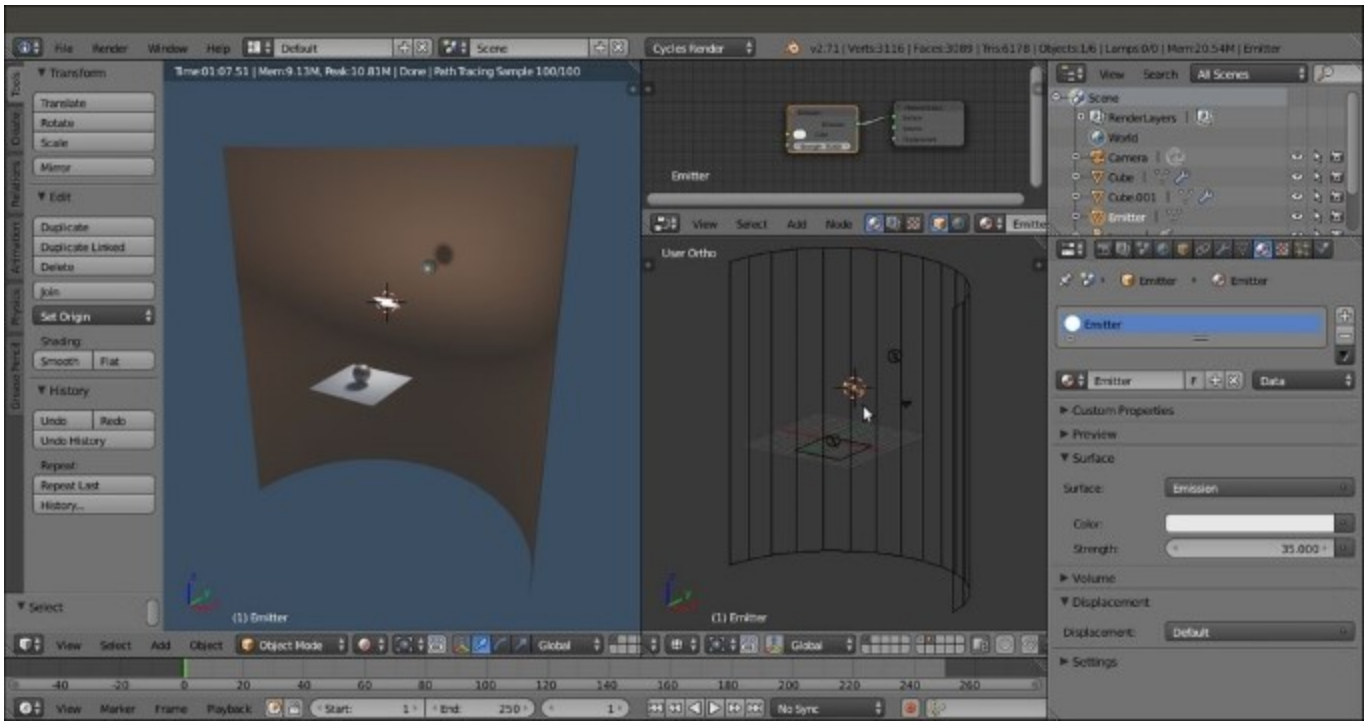
Both Lamps and mesh-lights can use textures too, for example, project colored lights on the scene, but only a mesh-light can be unwrapped and UV-mapped with an image map.

One advantage Lamps have over mesh-lights is that they can be made unidirectional easily, that is, apart from **Point** lamps, they cast light in only one direction. The following screenshot shows the casting of light with a Spot Lamp:



A Spot Lamp allows light to point in just one direction

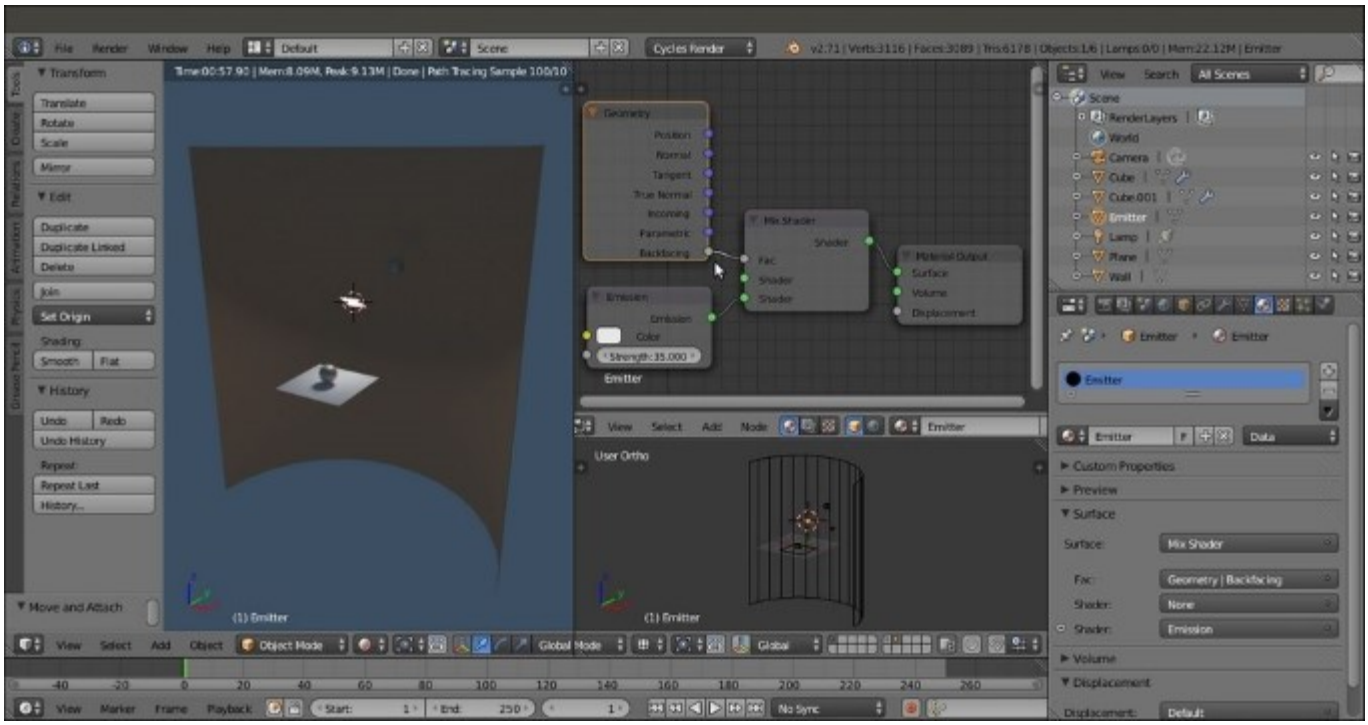
In the preceding screenshot, you can see that only the Plane and the Spheroid in front of the Spot lamp receive light. With a mesh-light plane replacing the Spot lamp, objects in both the front and the back (the half-cylindrical **Wall** and the second Spheroid) receive light.



A mesh-light emitter illuminates the region backward and forward by default

What if we want to light the object in only one direction (Plane and Spheroid in front) with a mesh-light? Is there a way to make a light-emitting plane emit light only from one side and not the opposite side? Yes, there is; follow these steps:

1. Open the `01_meshlight.blend` file, which has prepared the scene used for the preceding screenshots, and be sure to enable only the first and the seventh layer.
2. Put the mouse cursor on the left vertical 3D view, and press *Shift* + *Z* to navigate in **Rendered** view mode.
3. Click on the **Emitter** item in the **Outliner** to select it (if not already selected), and put the mouse pointer in the **Node Editor** window. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and move it to the link connecting the **Emission** node to the **Material Output** node to paste it in between them.
4. Add a **Geometry** node (press *Shift* + *A* and navigate to **Input** | **Geometry**) and connect its **Backfacing** output to the **Fac** input socket of the **Mix Shader** node.
5. Switch the **Emission** node output from the first **Shader** input socket of the **Mix Shader** node to the second node, as shown in the following screenshot:



Thanks to the Backfacing output of a Geometry node as Factor, a mesh-light can illuminate in only one direction

6. Save the file as `01_meshlight_final.blend`.

We have already seen that in a **Mix Shader** node, the first (upper) green **Shader** input socket is considered equal to a `0` value, while the second socket is considered equal to a `1` value. So, the **Backfacing** output of the **Geometry** node is telling Cycles to make the mesh-light plane emit light only in the face-normal direction, and to keep the opposite back-facing side of the plane black and non-emitting (just like a blank shader).

By switching the **Emission** node connection to the first **Mix Shader** input socket, it's obviously possible to invert the direction of the light emission.

Using volume materials

Very briefly (because there are dedicated recipes in the last chapter of this Cookbook), let's take a look at how volumetric materials work in Cycles.

Volumetric materials are exactly what they sound like. Instead of the surface of an object, Cycles renders the inner volume of that object, and this gives space to a lot of interesting possibilities—not only can elusive materials such as smoke, fire, clouds, or light transmission effects through the medium be realized, but peculiar shapes can also be obtained from the volume itself by Boolean operations made through material nodes.

The drawback is that volume materials are slow—a lot slower compared to the surface materials, but hopefully, this is an issue that will be fixed in some way in the future (be aware that from Version 2.72, volume materials are available on GPUs too).

Getting ready

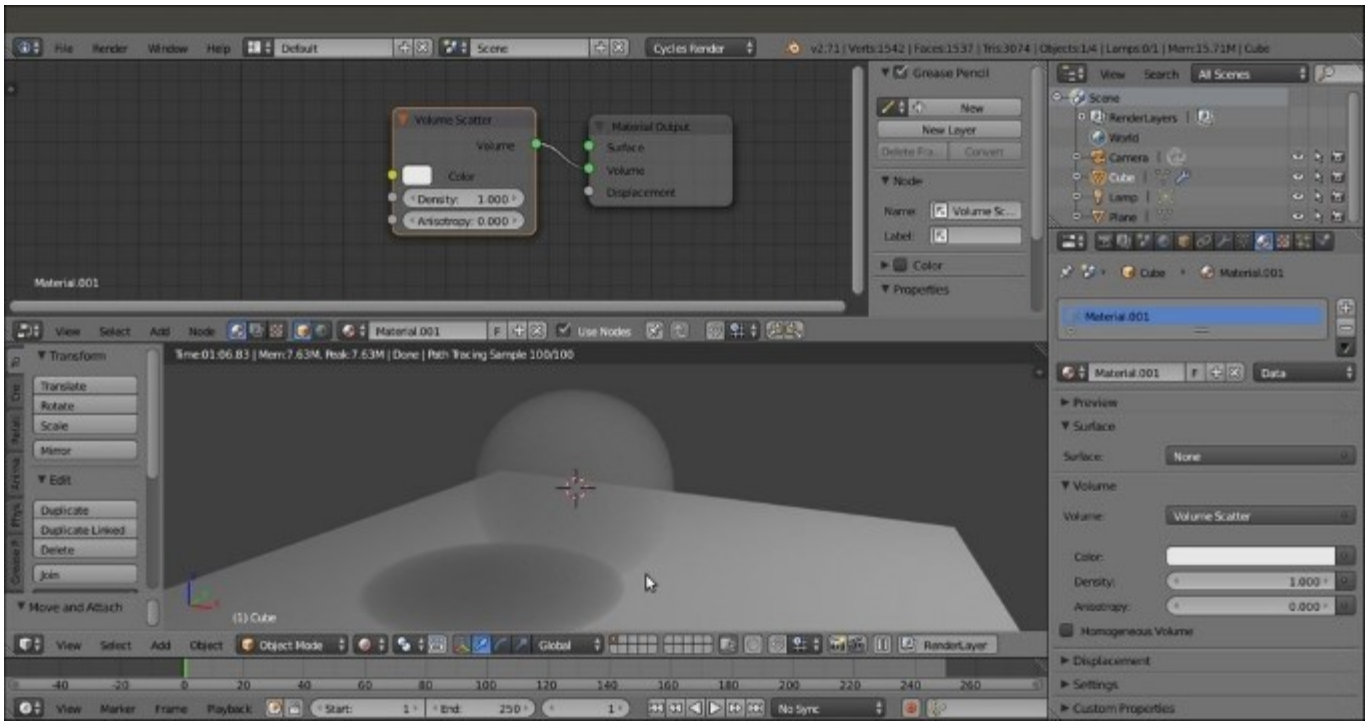
Let's start with our usual Spheroid blend file:

1. Open the `start_02.blend` file and delete the material assigned to the Spheroid.
2. Put the mouse cursor in the 3D view and press *Shift* + *Z* to navigate to the **Rendered** view.
3. Click on the **New** button to add a new material, and then switch the **Diffuse** node link from the **Surface** input socket to the **Volume** input socket of the **Material Output** node.

How to do it...

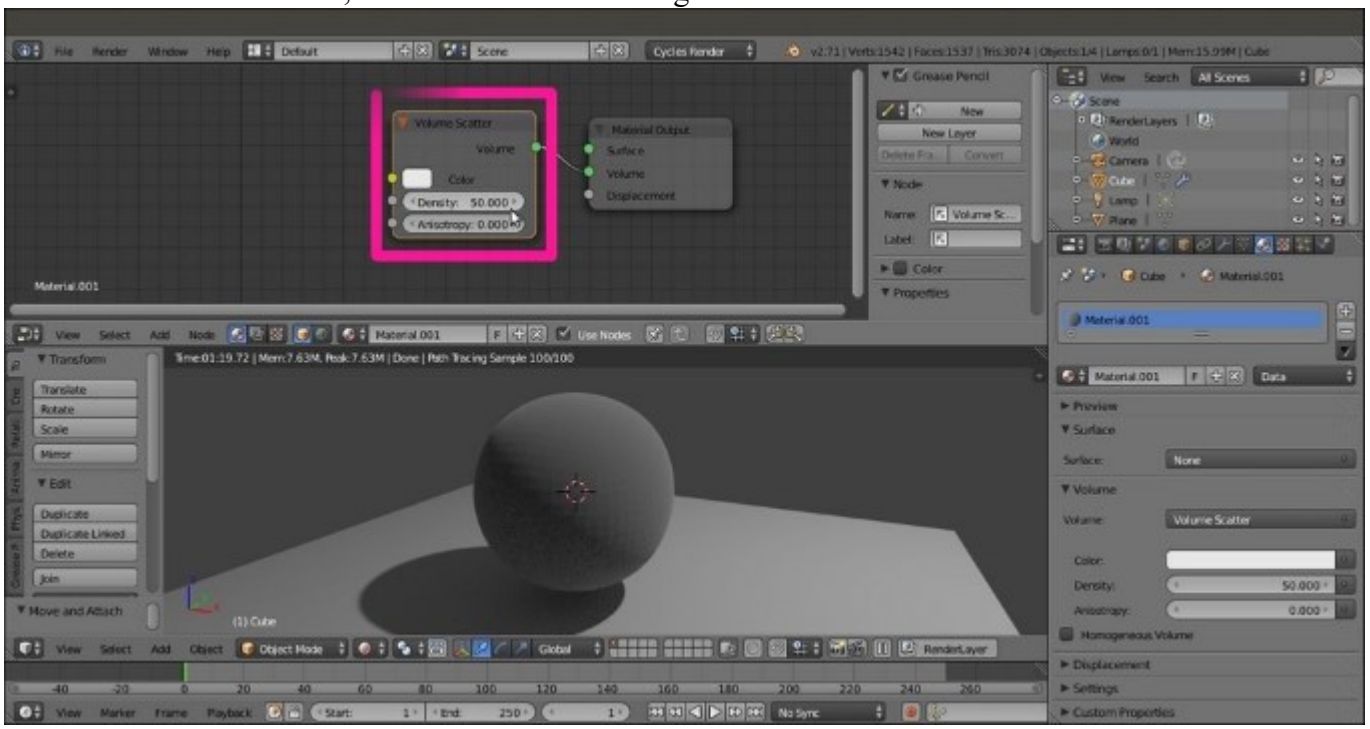
Now let's go to the volume section of the **Material** window with the following steps:

1. Go to the **Material** window under the **Properties** panel, and click on the **Diffuse BSDF** labeled button to the side of the **Volume** item. In the pop-up menu, select the **Volume Scatter** node as shown in this screenshot:



The Rendered preview of a Volume Scatter node assigned to the Spheroid

2. Change the **Density** value of the **Volume Scatter** node from `1.000` to `50.000`. The Spheroid looks a lot more solid now, as shown in the following screenshot:

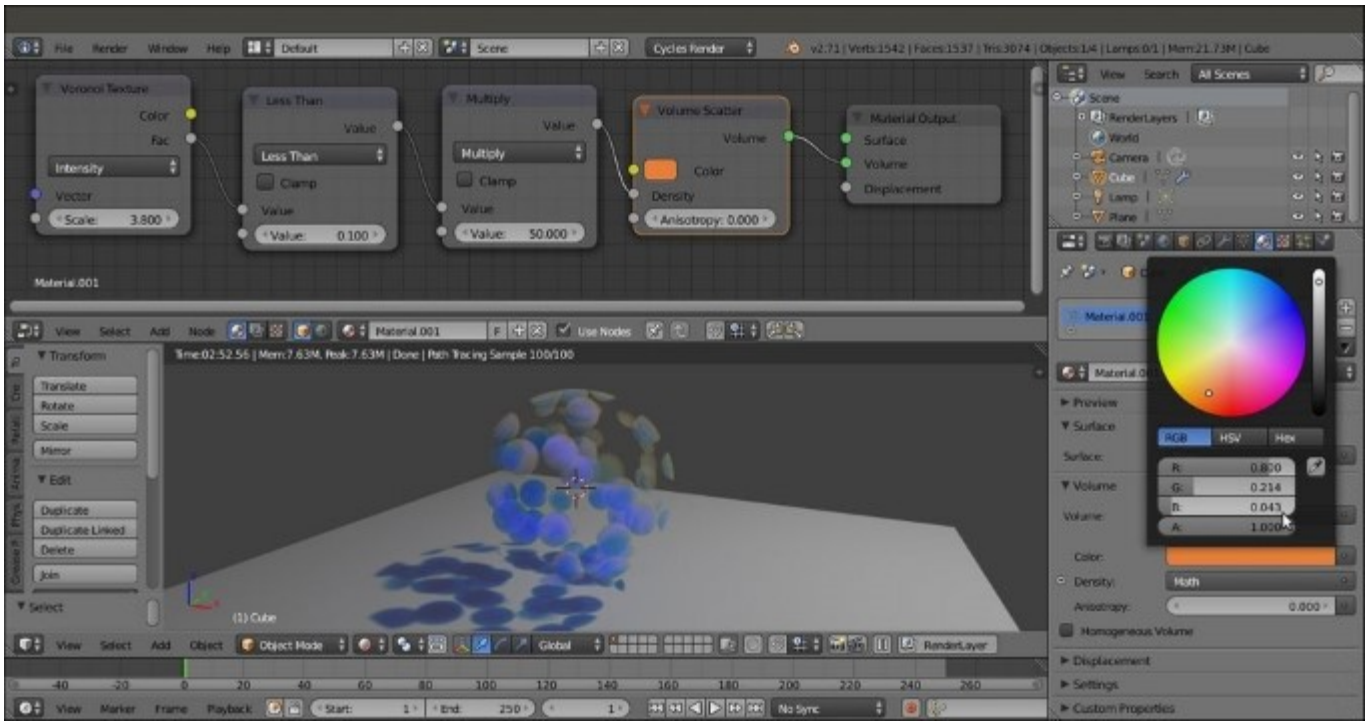


The effect of the Volume Scatter node with increased density

3. Add a **Voronoi Texture** node (Press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**). Connect the **Fac** output to the **Density** input socket of the **Volume Scatter** node. Set the **Voronoi** scale to `3.800`.
4. Add a **Math** node (Press *Shift* + *A* and navigate to **Converter** | **Math**) and paste it in the link between the **Voronoi Texture** and the **Volume Scatter** nodes. Set **Operation** to **Less Than** and second **Value** to `0.100`.
5. Add a second **Math** node and paste it right after the first node. Set the **Operation** to **Multiply** and second **Value** to `50.000`. Here is a screenshot of the output of a **Voronoi Texture** node for your reference:

The output of a Voronoi Texture node used as Factor for the density of the Volume Scatter node

6. Click on the **Color** button of the **Volume Scatter** node. Set the **RGB** values to `0.800`, `0.214`, and `0.043`, respectively.



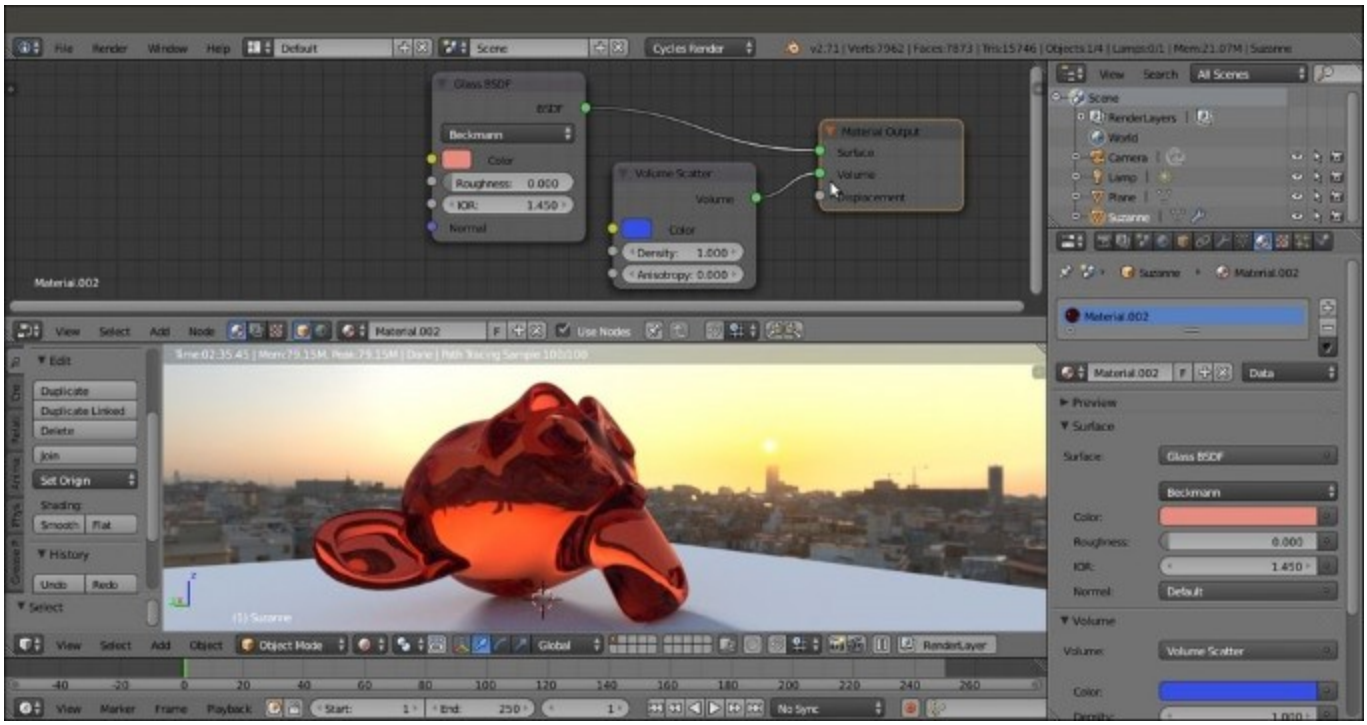
The scattered light is obviously of a hue complementary to the color assigned to the volume

7. Save the file as `01_volumetric.blend`.

How it works...

We have seen that when we increase the **Density** value of the **Volume Scatter** node, the Spheroid starts to look more and more solid. So, we used the output of a **Voronoi Texture** node and clamped it with a **Less Than** node to show only the values that are not beyond the `0.100` limit. Then we multiplied the value by `50.000`, thus increasing the density of the Voronoi spheres and making them appear as solid objects inside the Spheroid volume.

Remember that in this case, we rendered only the inside of the object and not the surface. Anyway, a combination of **Surface** and **Volume** is possible and can give interesting results, as shown in the following screenshot:



Combining a Glass shader for the surface with a Volume Scatter node for the inside of the mesh

There's more...

Volumes also work in the World. In fact, the **World Output** node now has a **Volume** input socket. By connecting a **Volume Scatter** or **Volume Absorption** node to the **World Output** node, it is possible to obtain several special effects, for example, fog, mist, atmospheric perspective, atmospheric scattering effects, and a body of water for an underwater scene. Clearly, it's also possible to fill this environment volume with textures.

In any case, you won't usually fill the entire World with a volumetric material because the World in Blender is considered as going to an infinite distance, and this would make the volume calculation too heavy. It's better to use a scaled Cube, properly placed and filled with the volume material.

To know more about volume materials, go to the last chapter of this Cookbook or to the documentation on the wiki at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Materials/Volume.

Using displacement

The last input socket of the **Material Output** node is **Displacement**. Sadly, it seems that at the moment, its use is limited.

Getting ready

By enabling **Experimental** in the **Feature Set** option under the **Render** tab in the **Render** window, it's possible to have access to an incomplete displacement feature:

1. Open the `start_03.blend` file, select the Spheroid, and delete its material.
2. Go to the **Render** window under the **Properties** panel. In the **Render** tab, click on the **Feature Set** button, labeled with **Supported** by default, and select **Experimental**.
3. Go to the **Object data** window to find a new tab named **Displacement**, where we can choose between three options: **Bump**, **True**, and **Both** (the **Use Subdivision** and **Dicing Rate** buttons don't seem to work yet).

   ### Note

   Bump will give us the average bump effect, which is the same as connecting the texture output in the **Displacement** input of the **Material Output** node (this is a different way to have an overall bump effect, and it works without the need to set the **Feature Set** option to **Experimental**).

   By setting the method to **True**, we can have a displacement effect that is not different from the **Displace Modifier** output, and the mesh must be subdivided.

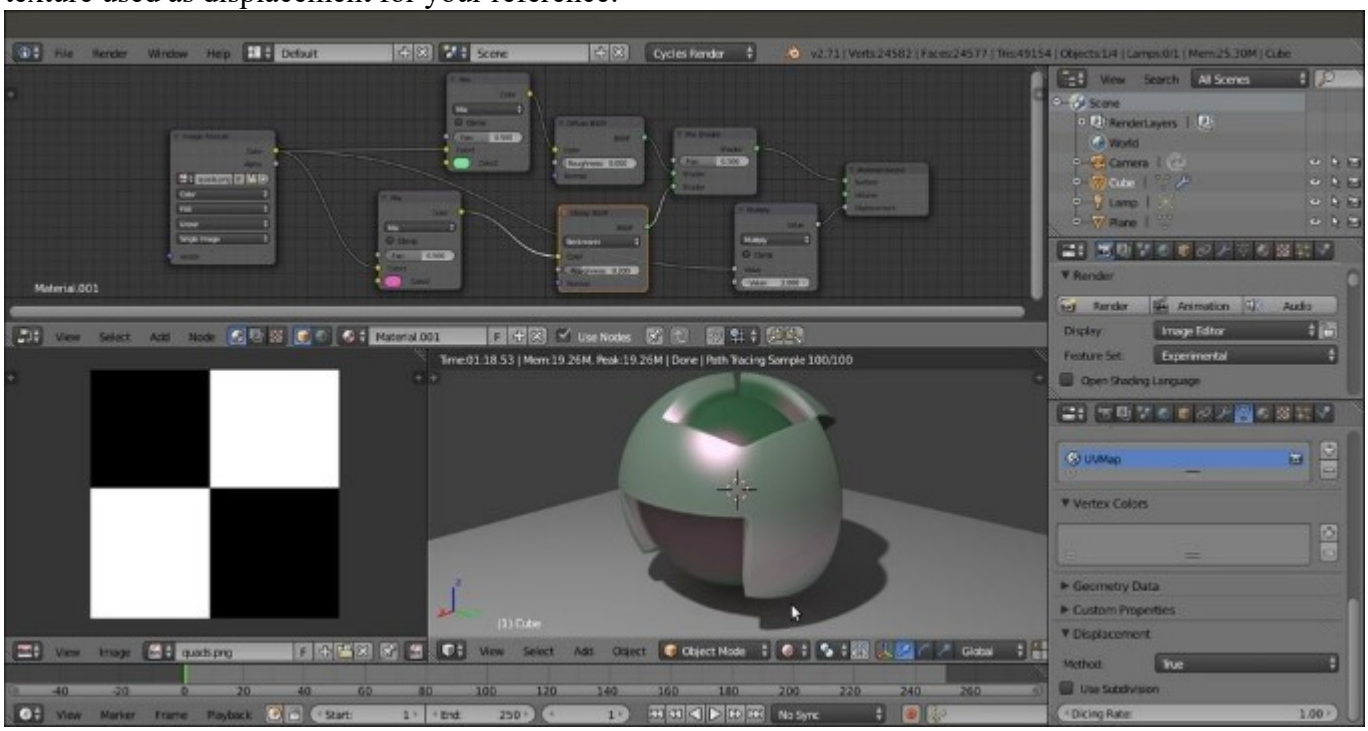
   Both will use the texture gray-scale values' information for a displacement and the bump effect together.

4. Select **True**.

How to do it...

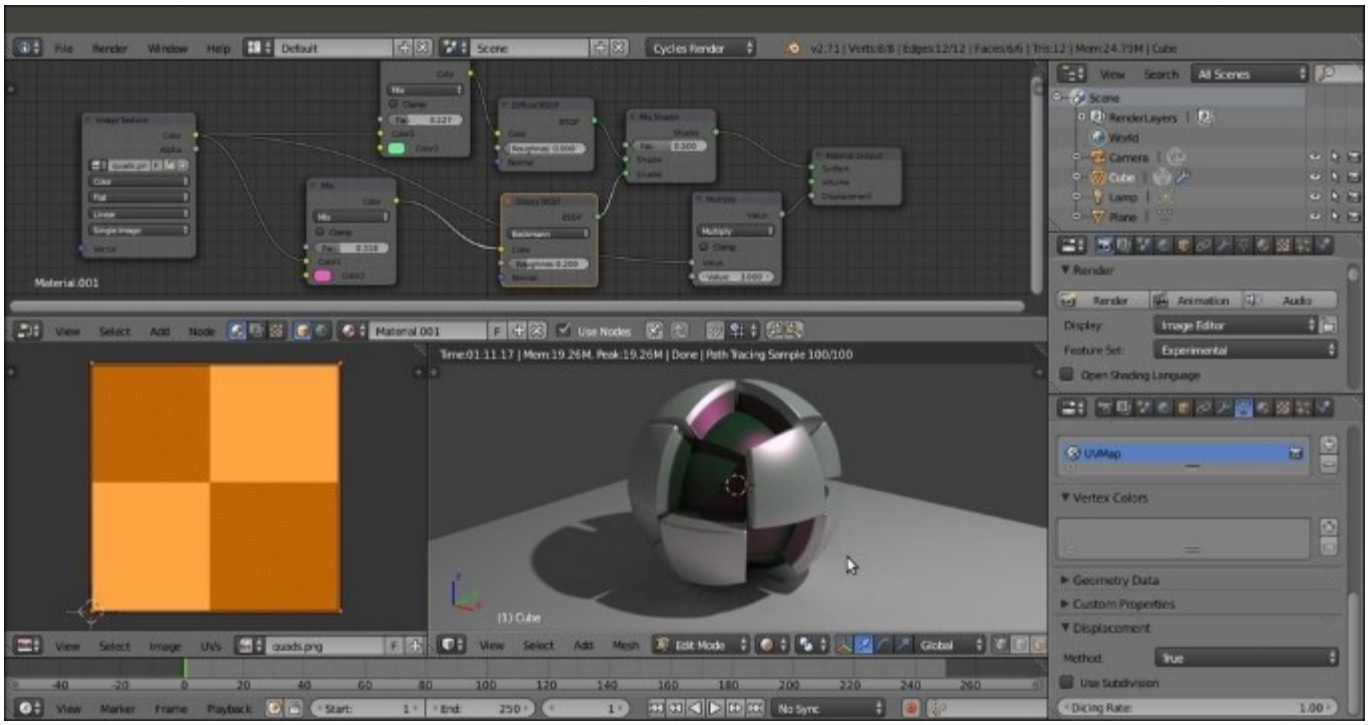
1. Go to the **Material** window under the **Properties** panel and click on the **New** button. In the **Displacement** tab, click on the **Default** button, and in the pop-up menu, select the **Image Texture** node.
2. Click on the **Open** button, browse to the `textures` folder, and load the `quads.png` image.
3. Split the bottom 3D window to open a **UV/Image Editor** window.
4. Press *Tab* to go to **Edit Mode**. Then press *U* with the mouse pointer in the 3D window. In the **UV Mapping** menu, select **Smart UV Project**, then load the `quads.png` image in the `UV/Image Editor`, and press *Tab* again to go out of **Edit Mode**. Note that this is the quicker way to unwrap the Spheroid, which is still a Cube at its lower level of subdivision. If you want, you can do a better unwrapping by placing seams to unfold it and by selecting a normal **Unwrap** option from the pop-up menu.
5. Go to the **Object modifiers** window and raise the **Subdivisions** levels for both **View** and **Render** to `6`.

6. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and paste it between the **Image Texture** node and the **Material Output**. Set **Operation** to **Multiply**, and the second option, **Value**, to `2.000` (if you don't see any modification in the rendered preview, it's an update issue, which can be solved by pressing *Tab* twice to go in and out of **Edit Mode**).
7. Add a **Glossy** node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**) and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), and connect them to build the average basic material we already know.
8. Add two **MixRGB** nodes (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect them to the color input sockets of the **Diffuse** and the **Glossy** nodes.
9. Finally, connect the color output of the **Image Texture** node to the **Color1** input sockets of the **MixRGB** nodes, and set colors for the **Color2** sockets. Here is a screenshot of a checker image texture used as displacement for your reference:



A checker image texture used as a color and output for the Rendered displacement of the Spheroid

Instead of the **Smart UV Project** option to unwrap the Spheroid, try the default `1:1` UV Mapping (the **Reset** item in the menu, which gives the whole image mapped on each face). The following screenshot shows the checker image texture used with the different unwrap:



The checker image texture used as a color and output for the rendered displacement of a Spheroid with a different unwrap

10. Save the file as `9931OS_01_displacement.blend`.

In any case, this is just for a temporary demonstration; the feature is still incomplete. At the moment, it seems to work quite well only if the texture is mapped with UV coordinates. This is definitely going to change in the future.

How it works...

Simply put, the gray-scale values of the texture are multiplied by the value we put in the second slider of the **Math** node. For example, if we set a value of `0.500`, the intensity of the effect will be the half of the default value (*1.000 x 0.500 = 0.500*). With a value of `3.000`, the effect would be three times the default value, and so on. Similar to Blender Internal, the value can also be set as negative, thereby inverting the direction of the displacement.

Chapter 2. Managing Cycles Materials

In this chapter, we will be covering the following recipes:

- Preparing an ideal Cycles interface for material creation
- Naming materials and textures
- Creating node groups
- Grouping nodes under frames for easier reading
- Linking materials and node groups

Introduction

As with Blender Internal materials, Cycles materials can (and should) be organized to optimize your workflow.

Material nodes in Cycles can easily grow quite complex, and it's sometimes a good idea to split and label the different parts of a shader's network, just to make the meaning of the different sections clearer (even to yourself because maybe at a certain point of your workflow, you will forget how exactly that 120-node material you made a couple of months ago works). Moreover, organized materials can be easily reused in other files and projects or as parts of bigger and different materials.

Organization of materials is basically done by grouping them or giving them proper names and defined locations so that they can be easily found in the hard disk.

Preparing an ideal Cycles interface for material creation

Before starting with the actual organization, it's a good idea to prepare a material creation screen to be saved in your Blender preferences.

It is possible, in fact, to prepare a basic scene setup that includes the elements and the settings we need to do the job best.

In any case, just take this recipe with a grain of salt, that is, take it as more of a suggestion or as a starting point that you can eventually modify to better suit your needs.

Getting ready

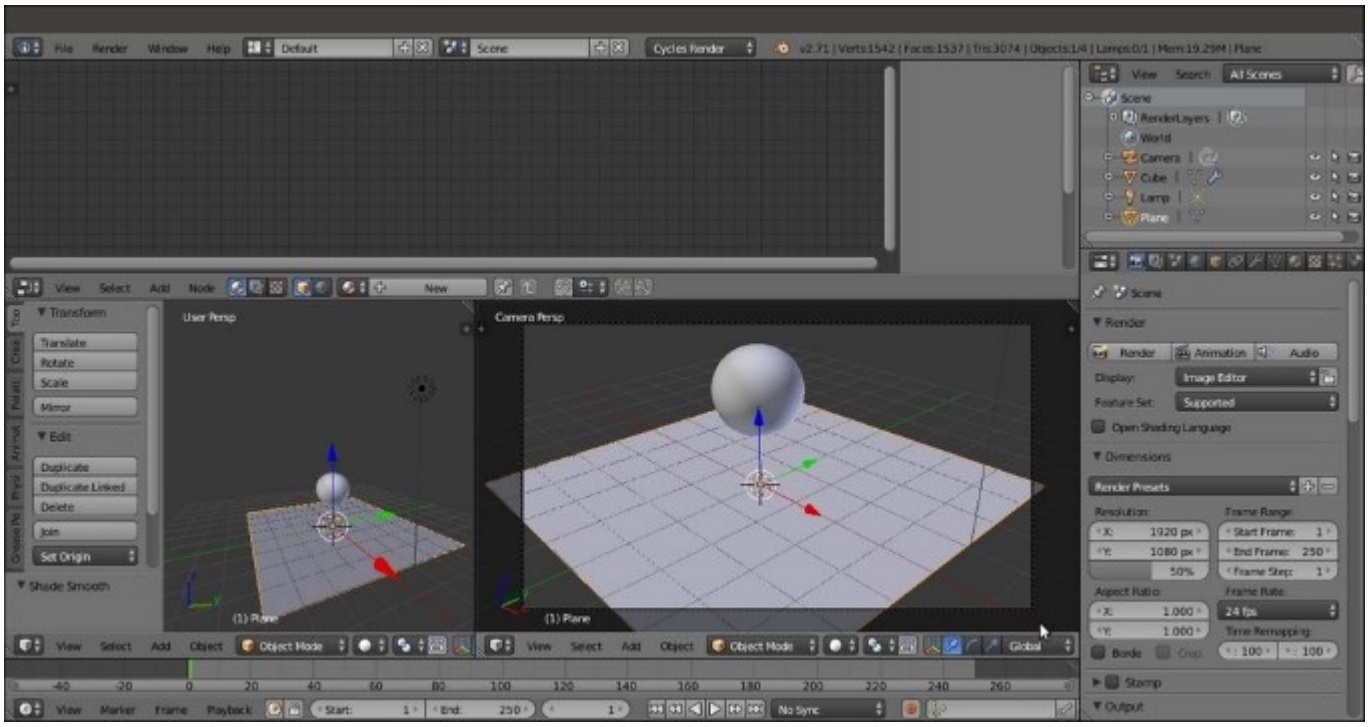
Start Blender and in the upper menu (the **Engine to use for rendering** button), switch to **Cycles Render**.

How to do it...

We are now going to customize the **Default** screen:

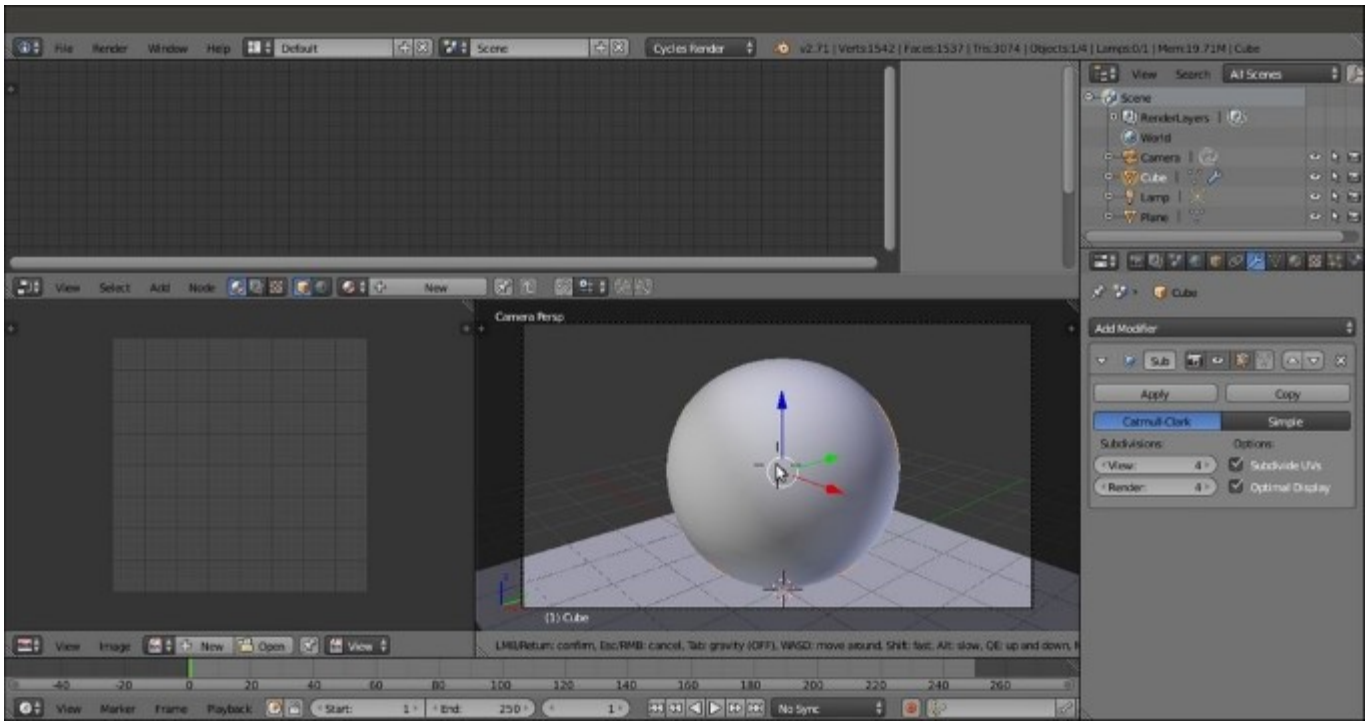
1. Split the 3D view into two horizontal rows. To do this, move the mouse cursor onto the lateral edge of the window. The cursor changes to a double arrow icon. Right-click on the edge, and from the little pop-up menu that appears, select **Split Area** (or click in the top-right corner of the window and move it).
2. Change the upper window to a **Node Editor** window by selecting the item from the **Editor Type** button in the left corner of the bottom bar (or by putting the mouse cursor in the window and pressing the *Shift* + *F3* keyboard shortcut).
3. Select the default Cube in the scene if it is not selected already, and go to the **Object modifier** window under the **Properties** panel to the right of the screen. Assign a **Subdivision Surface** modifier to the Cube, which is now a Spheroid, and set the **Subdivisions** levels for both **View** and **Render** to `4`. Check the **Optimal Display** item.
4. Set the Spheroid shading mode to **Smooth** by clicking on the appropriate button under the **Shading** subpanel in the **Tools** tab on the left.
5. Move the Spheroid upward by 2 units on the *z* axis (press *G*, then press *Z*, enter digit *2*, and finally, press *Enter*).
6. Ensuring that the cursor is still at the center of the scene (if not, press *Shift* + *C* to center it), press *Shift* + *A* and navigate to **Mesh** | **Plane** to add a Plane.
7. Press *Tab* to go to **Edit Mode** and scale the Plane four times bigger (press *Tab*, then press *S*, enter digit *4*, and finally, press *Enter*). Exit **Edit Mode**.
8. Split the bottom row into two parts, put the mouse cursor in the 3D window on the right, and press *0* in the numeric keypad of the keyboard to go to the **Camera** view. Then press *T* to close the **Tool Shelf** panel with the tabs on the left. Scroll the mouse wheel to fit the **Camera** view field into the window (or for a finer control, press *Ctrl* + the middle button of the mouse and move the mouse).

This screenshot shows where we are now:



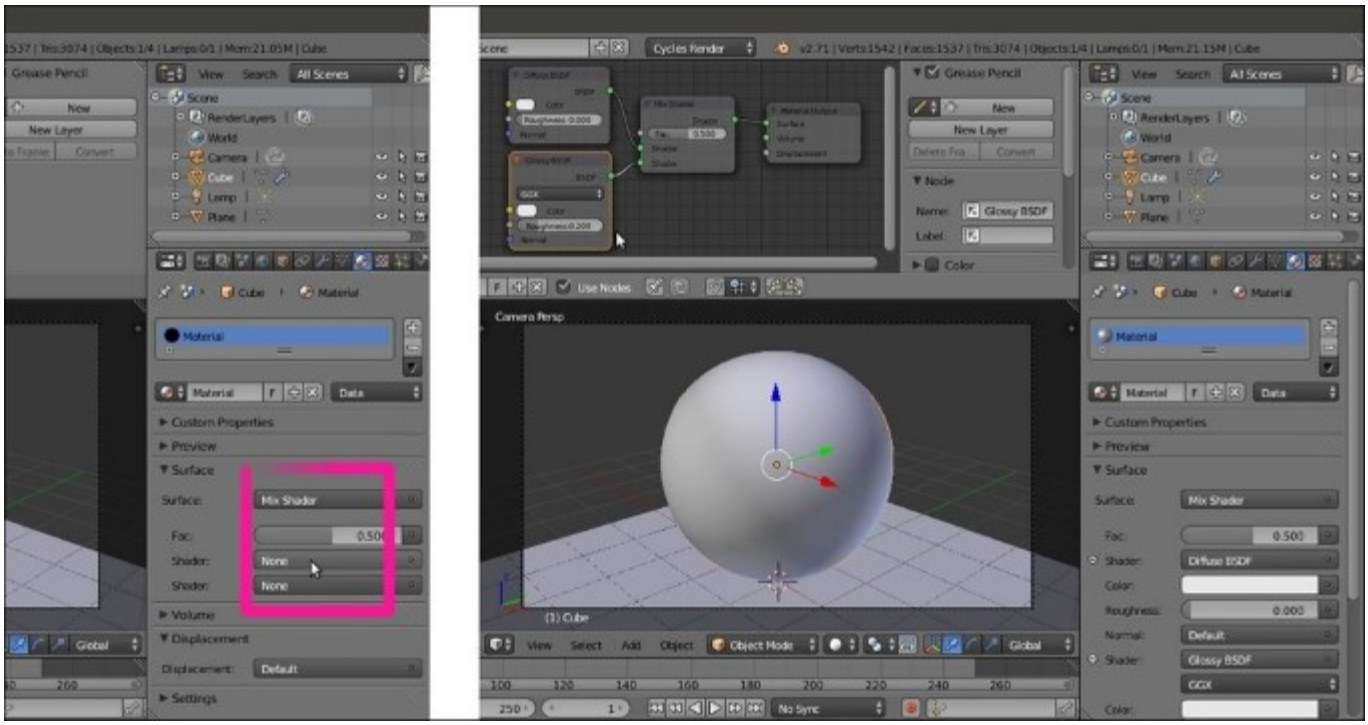
The first steps of the Default screen customization for Cycles material creation

9. In the **Editor Type** button in the left corner of the bottom bar of the left 3D window, select **UV/ Image Editor**.
10. Select the Spheroid, go to **Edit Mode**, and scale it to twice the current size (press *Tab*, then press *S*, enter digit *2*, and finally, press *Enter*). Exit **Edit Mode**.
11. Move the mouse to the **Camera** view and press *Shift* + *F* to enter Walk Navigation mode (a viewfinder appears to show the center of the camera field). By moving the mouse to pan and by pressing the *W* or the *S* key to go forward or backward, respectively, adjust the **Camera** view to fit the Spheroid better. Then press *Enter* or click to confirm, as shown in the following screenshot:



Centering the Camera view on the Spheroid

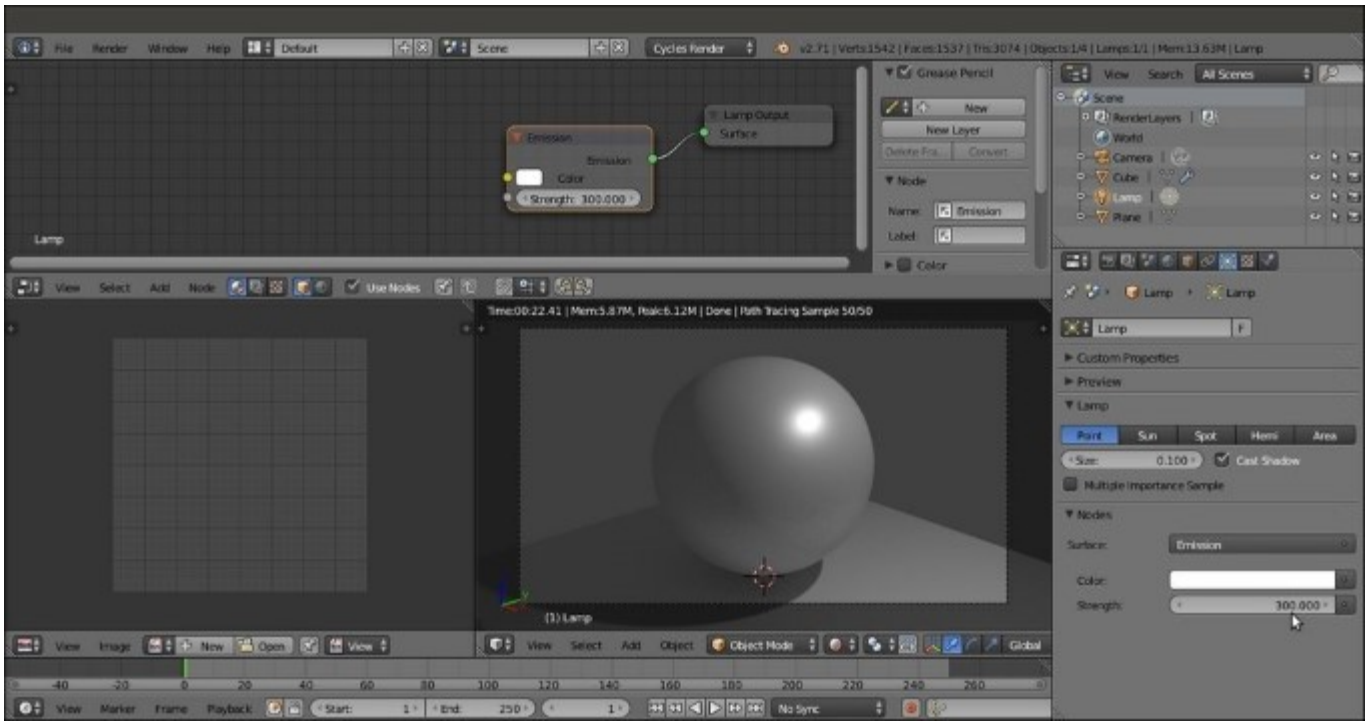
12. Select the Plane. Click on **New** in the **Node Editor** toolbar to assign a new material (**Material.001**). Rename it `Plane` and leave all the settings as they are.
13. Select the Spheroid and click on **Use Nodes** in the **Material** window under the **Properties** panel to the right of the screen or in the **Node Editor** toolbar.
14. Put the mouse in the **Node Editor** window and scroll the wheel to zoom to the nodes.
15. Under the **Surface** subpanel in the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. Then click on the first **Shader** slot to select a **Diffuse BSDF** shader and on the second slot for a **Glossy BSDF** shader (the two **Shader** slots I'm referring to are highlighted in the next screenshot).
16. In the **Node Editor** window, adjust the position of the nodes to make them more readable, as shown in this screenshot:



Preparing a basic average material with the Diffuse and the Glossy components

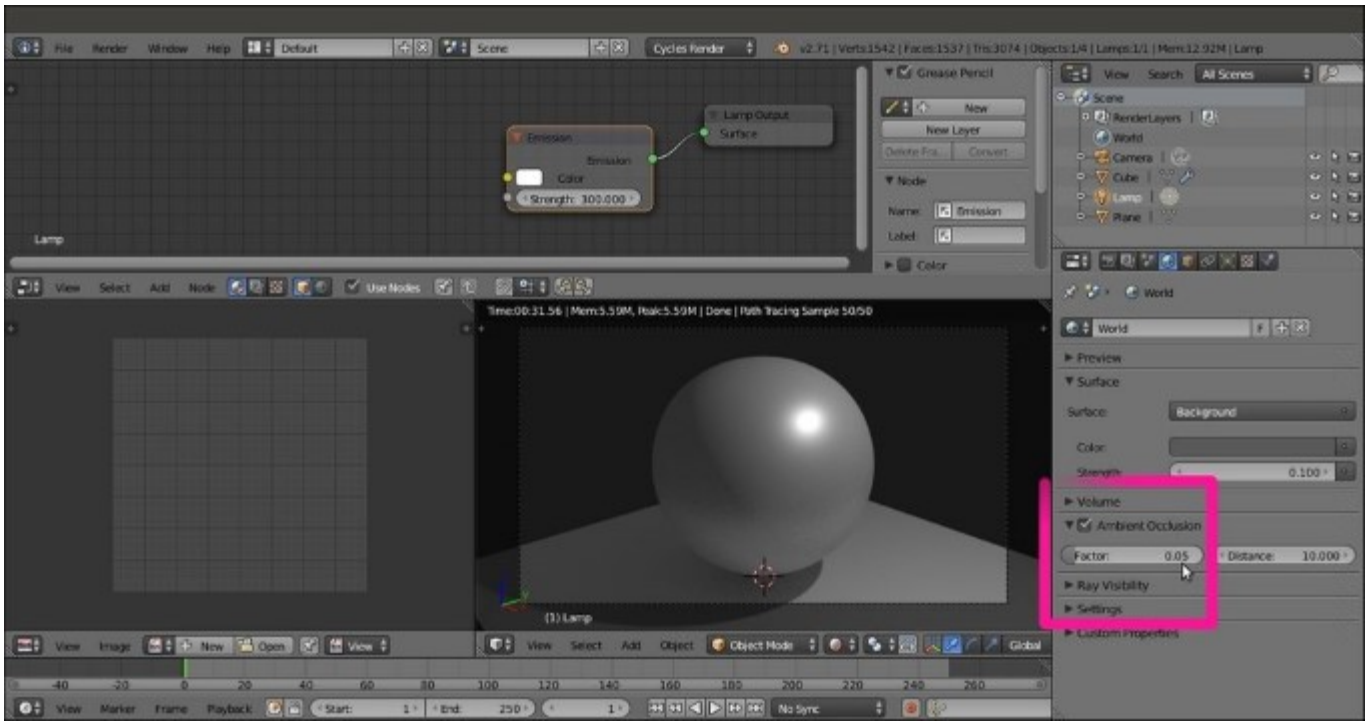
17. Set the **Camera** view mode to **Rendered** by clicking on the **Viewport Shading** button on the window toolbar and selecting the top item or by pressing *Shift* + *Z* with the mouse cursor inside the 3D view.
18. Go to the **Render** window under the **Properties** panel on the right, and under the **Sampling** subpanel, set both **Clamp Direct** and **Clamp Indirect** to `1.00`, and both **Render** and **Preview** to `50` samples.
19. Under the **Light Paths** subpanel, set **Filter Glossy** to `1.00`.
20. Go to the **Outliner** window and select the **Lamp** item. Go to the **Object data** window under the **Properties** panel and click on the **Use Nodes** button under the **Nodes** subpanel. Increase the **Strength** value to `300.000`.

    Now the output will look like what is shown in this screenshot:



Setting the Camera view to Rendered and increasing the Lamp strength

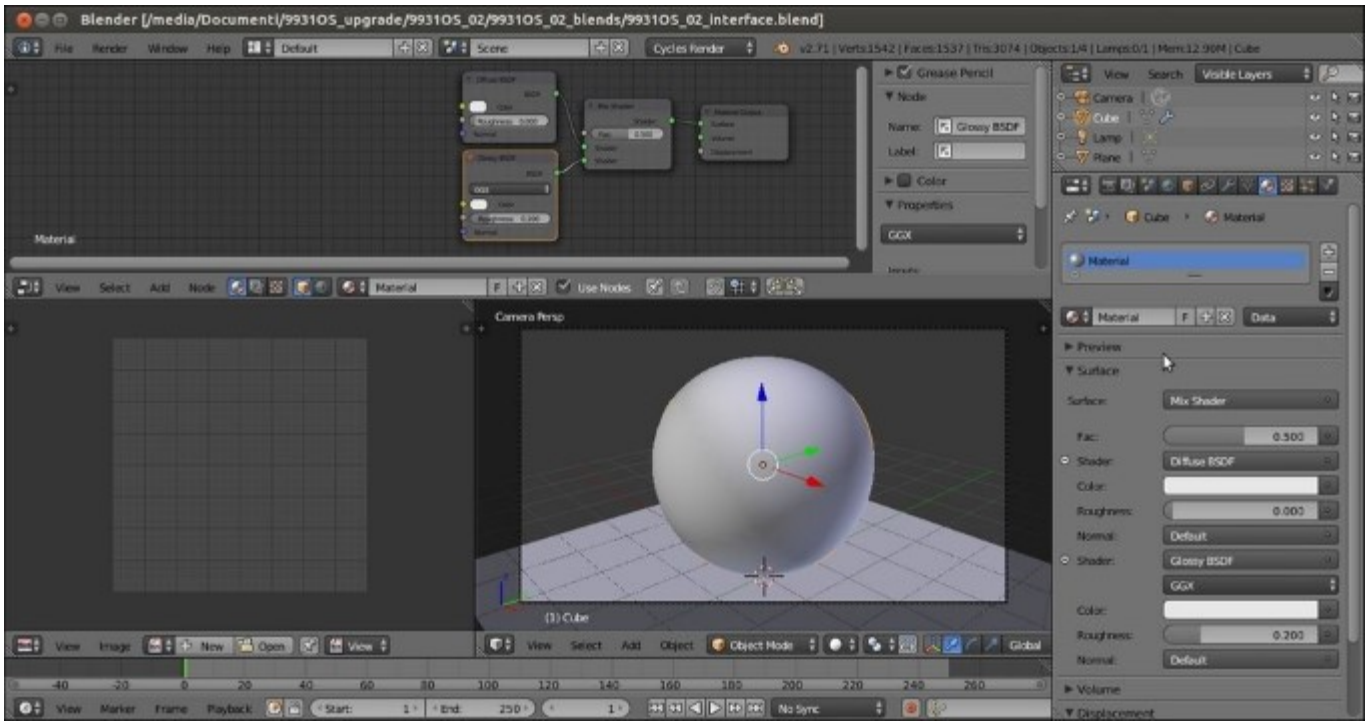
21. Go back to the **Render** window under the **Properties** panel and set the **Percentage** scale for render resolution under **Dimensions** to `25%` to have smaller but faster rendering.
22. Under the **Performance** subpanel, set **Viewport BVH Type** to **Static BVH** and check **Use Spatial Splits**, **Cache BVH**, and **Persistent Images** (these are probably not really useful for a simple Spheroid, but they are useful if you want to render a more complex object).
23. Go to the **World** window and click on the **Use Nodes** button under the **Surface** subpanel. Click on the **Color** slot, set the **RGB** values to `0.100`, and set the **Strength** value to `0.100`.
24. Set the **Factor** value for the **Ambient Occlusion** subpanel to `0.05` but let it remain disabled. You can enable the **Ambient Occlusion** subpanel or not, depending on your preferences, but remember that it adds light to the rendered image. I would say that it's usually better not to have the **Ambient Occlusion** subpanel activated by default but enabled only if really needed. In this case, the very low value can compensate a bit for the darkened background of the World, which is shown in the following screenshot for your reference:



Preparing the optional Ambient Occlusion setting

Optionally, other things that you can do include scaling the floor Plane bigger. In the **Outliner** window, set the mode to **Visible Layers** and click on the arrow icon to the side of the **Plane** item to make it nonselectable. Substitute **Lamp**, the default **Point** item, with a different type (**Sun** or **Spot**) or with a mesh-light plane.

25. Go back to the **Material** window. If you want to save this setting as the user default, press *Ctrl* + *U* (**Save Startup File**), or save the file with a meaningful name. Among the files provided with this module, you will find this file by the name of `9931OS_02_interface.blend`. The 3D view now looks like what is shown in the following screenshot:



The final overall look of the customization

How it works...

We set a very low World global illumination, keeping its color within the gray scale in order not to affect the color of the material. The floor plane is meant to have light bouncing on the shadowed parts of the object, and this can eventually be helped by the low **Ambient Occlusion** subpanel as well.

We prepared the **Rendered** view port as a **Camera** view to get better feedback for the final rendered image, which will show at 25 percent of the established size in the **UV/Image Editor** window on the bottom-left side of the screen.

By setting the **Clamp** values to `1.00`, we reduced the fireflies produced by the glossy shader, and by increasing the render and preview samples to `50`, we reduced the noise, at the same time keeping the rendering times reasonable, even with a not-very-powerful workstation.

The **Viewport BVH Type** is set to **Static BVH**, and the **Use Spatial Splits**, **Cache BVH**, and **Persistent Images** options are useful to reduce the calculation time for the bounding volume hierarchy of the mesh, which Cycles has to calculate every time it starts rendering. Anyway, these options are useful only if the mesh doesn't get any internal modification between renderings.

There's more...

From now on, every time we start Blender, the layout and the settings we just saved as default will be seen first.

But maybe we don't want to have this Cycles material interface every time we start, and we prefer to have it only as an option to be used if needed. Actually, in the previous steps, we modified the **Default** screen, but it's also possible to create new screens while keeping the original screen available. Here is the way to do this:

1. Start Blender with the factory settings (click on the **File** menu on the top main header and navigate to **Load Factory Setting**) and look at the top of the screen, in the main header on the side of the **Blender Render** button. There are two more buttons labeled **Default** and **Scene**.
2. By clicking on the **Default** button, we can set a different interface layout (there are already nine, each of which is studied for a different task, and their names are perfectly explicative). Clicking on **Scene** shows just the current scene.
3. By clicking on the + icon on the side of the **Default** button, we add a new screen layout named `Default.001`. Rename it `Materials`.
4. Then click on the + icon on the side of the **Scene** button, and by choosing the **Full Copy** item, add a new scene to the **Scene.001** file. Rename the file as something like `Cycles_Materials`. This new scene is a full copy of the default scene, coexisting but independent.

At this point, we can start with all the instructions already seen in the *How to do it* section of this recipe: switching to **Cycles Render**, splitting the 3D window, assigning the **Subdivision Surface** modifier to the default Cube, and so on.

When done, just click on the screens button, switch back to **Default**, and then save the user preferences (*Ctrl* + *U*). Now our material creation interface is saved as a screen option in a different scene. Every time we need to access it, it's enough to select the layout **Materials** from the screens button.

Naming materials and textures

It is well known that one of the most important things to do when working in every workflow with every 3D package is to give proper and explicative names to all the assets, that is, to the materials and the textures in our case.

Getting ready

Start Blender, go to the **File** menu in the top-left corner, and choose **Load Factory Setting** (this is just to be sure to start with the default Blender/Cycles settings).

Now, if you are in Blender Internal mode, switch to **Cycles Render**.

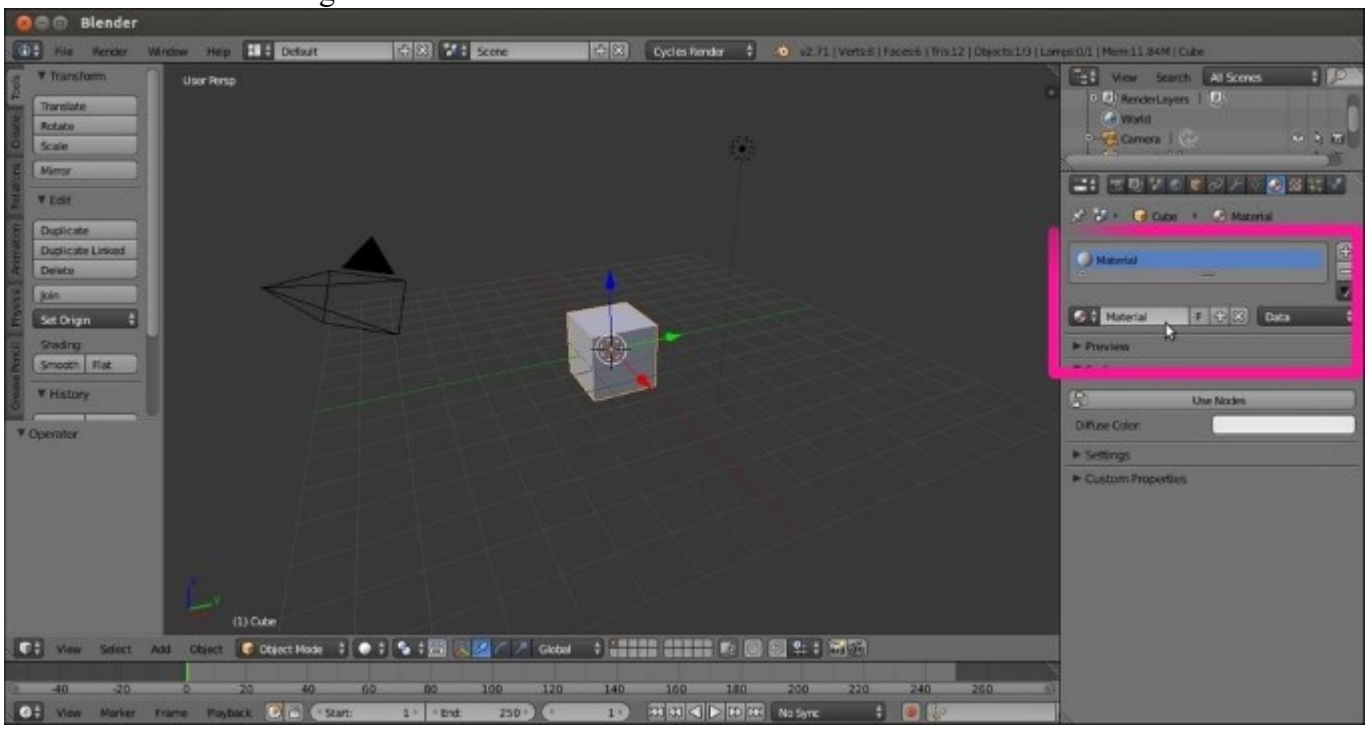
How to do it...

Now let's see the way material and texture naming works in Cycles.

Materials:

Adding and renaming materials in the **Material** window is done by performing the following steps:

1. Select the default Cube. Go to the **Material** window in the **Properties** panel. The default Cube already has a material assigned. This material has already been named `Material` by Blender, as shown in the following screenshot:

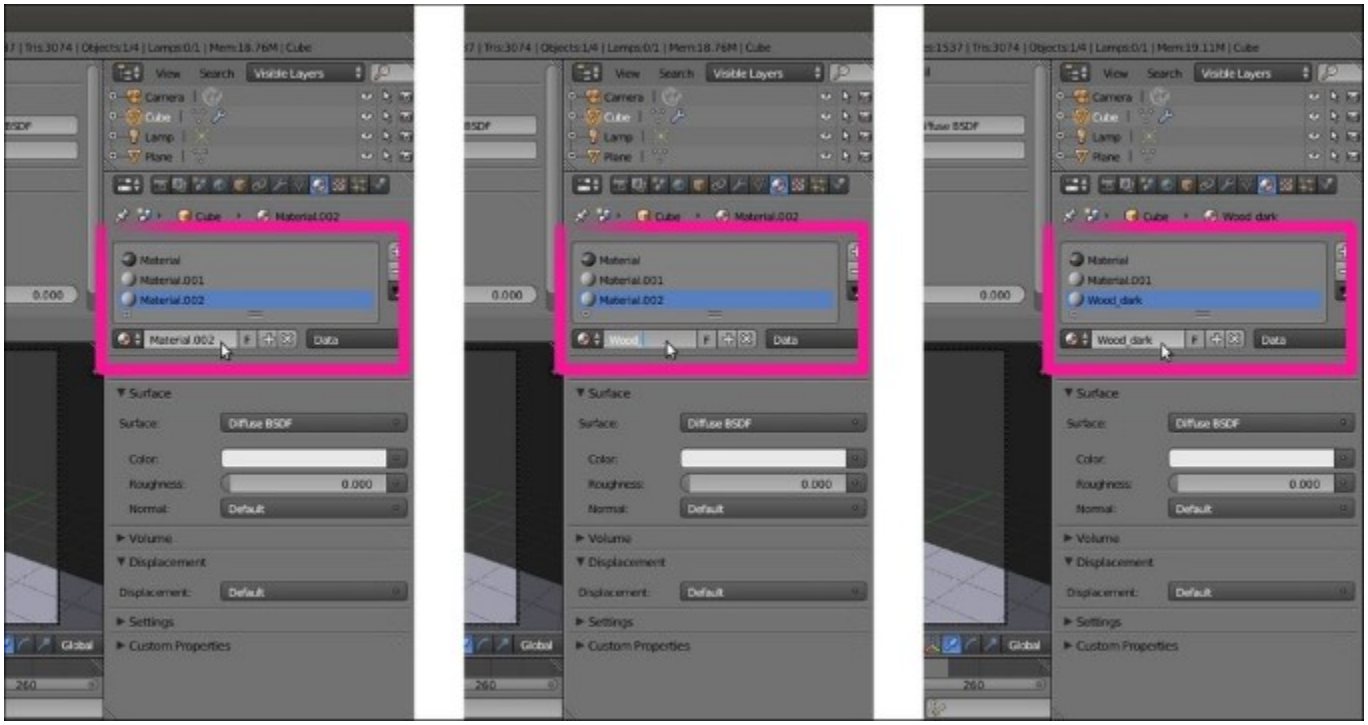


The material name datablock under the Material window

2. When you create a new material, for example, by clicking on the + symbol on the side of the material data block (add a new material) under the **Properties** panel, Blender automatically assigns a new name to this material, which is usually something like `Material.001`, `Material.002`, `Material.003`, and so on.

Having an automatic nomenclature can be handy in most cases, but it can become really confusing as a scene grows in complexity or if you have to reuse some of the materials in other situations. In such cases, we'd better rename all our materials with significant names.

To rename a material, it's enough to click with the left mouse button on the material name data block, type a new name, and then press *Enter* to confirm. This can be done in both the **Properties** panel and in the **Material** datablock button on the toolbar of the **Node Editor** window, as shown in the following screenshot:



Adding and renaming materials in the Material window

Textures:

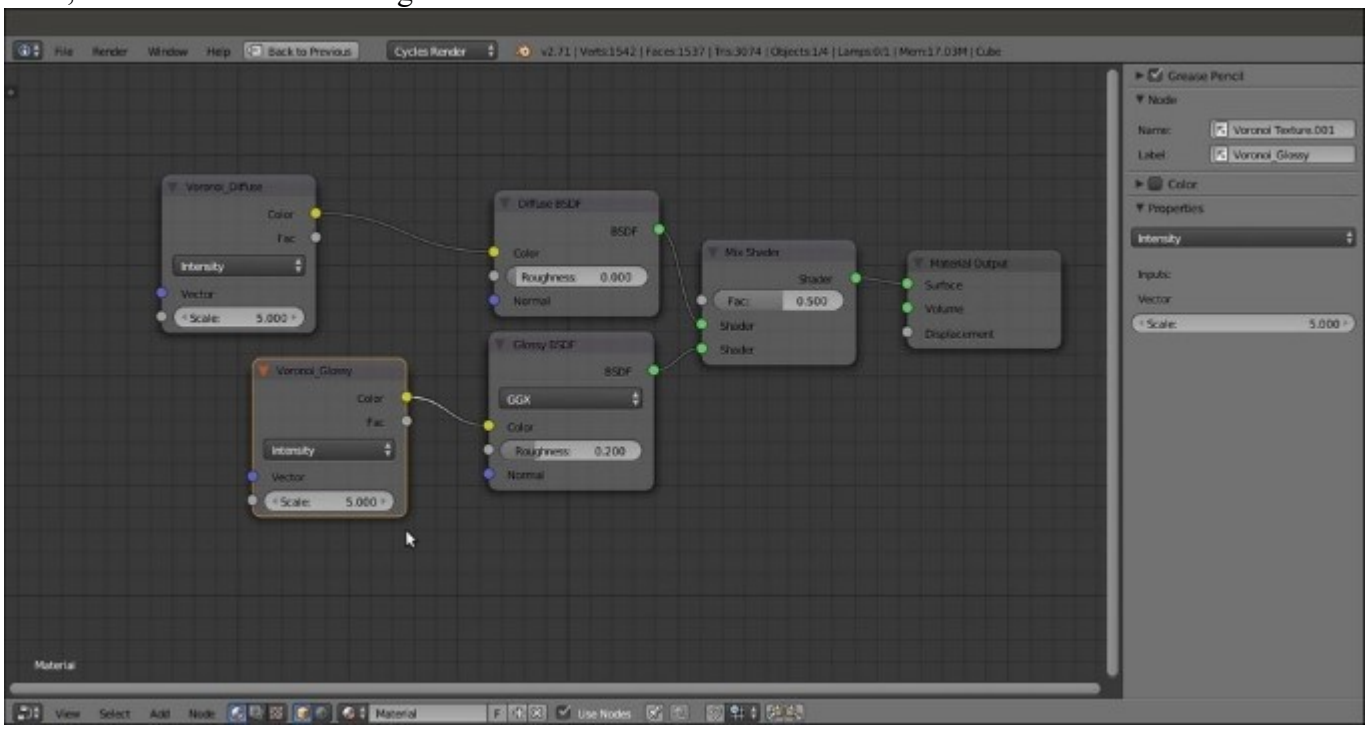
Things are a little different for textures. In Cycles, textures are no more data blocks but nodes, so every time we add a **Texture** node to a material network, it automatically gets named according to the kind of texture we added. This means that if we add a **Voronoi Texture**, the texture node is automatically named **Voronoi Texture**. What if we want to rename it to avoid confusion among three or four **Voronoi Texture** nodes? To rename the texture, perform the following steps:

1. Open the `9931OS_02_interface.blend` file.
2. Go to the **Node Editor** window and press *Shift* + *A* to add a **Voronoi Texture** node to the material (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**).

   On the right side of the **Node Editor** window, there is a **Properties** panel with several subpanels (press the *N* key if it's not already present). What interests us now is the **Node** subpanel with its two slots, **Name** and **Label**.

   As you can see, the name of the node (**Voronoi Texture**) is already present in the **Name** slot. By clicking on the name, it's possible to change it, but at the moment, this seems useful to identify the node in the **Properties** panel.

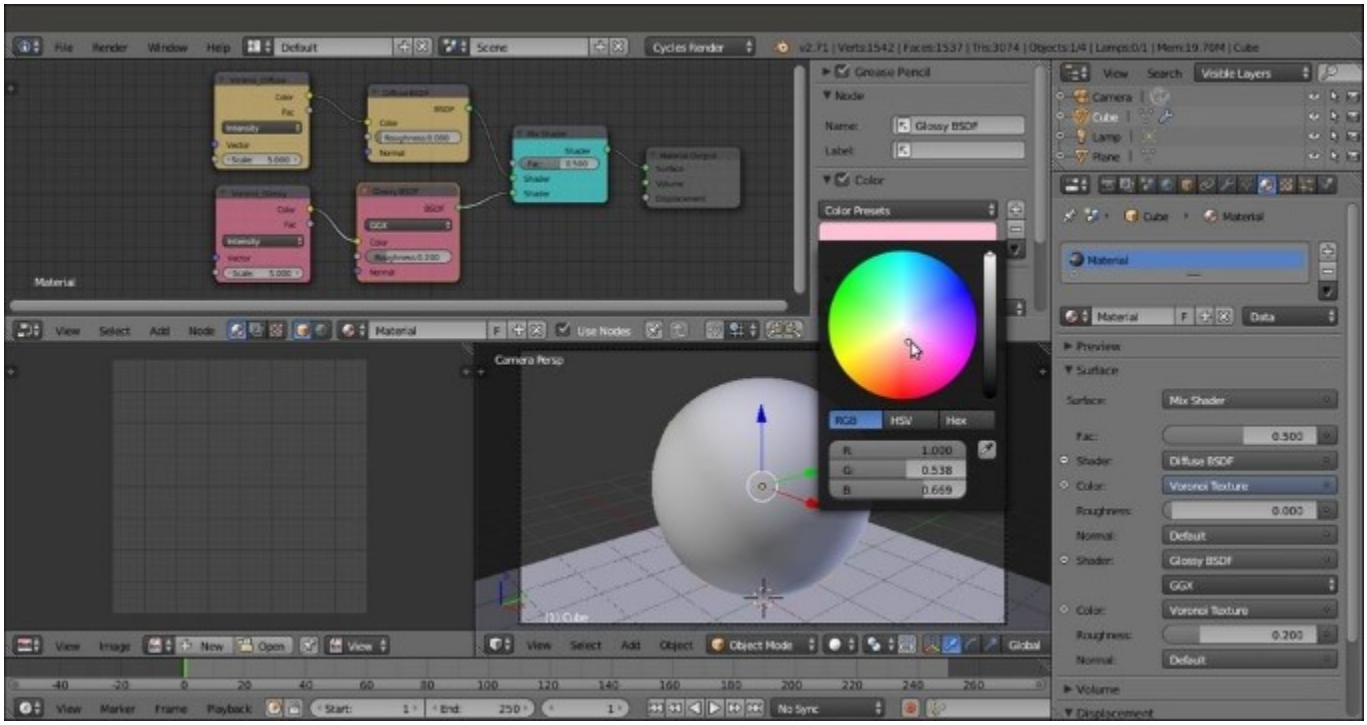
   The **Label** slot, which is empty by default, can be used to label a node in the **Node Editor** window.
3. Press *Shift* + *D* to duplicate the **Voronoi Texture** node. The duplicated node is automatically named `Voronoi Texture.001`.
4. Select the first **Voronoi Texture** node and write `Voronoi_Diffuse` in the **Label** slot of the **Properties** panel. Connect this node to the **Color** input socket of the **Diffuse BSDF** shader node.
5. Select the duplicated **Voronoi Texture** node and write `Voronoi_Glossy` in the **Label** slot of the **Properties** panel. Connect this node to the **Color** input socket of the **Glossy BSDF** shader node, as shown in the following screenshot:



The labeling of the nodes through the Active Node subpanel in the Node Editor Properties side-panel

There's more...

Even though this is not strictly related to renaming, let's quickly see one more option. Right below the **Node** subpanel, there is the **Color** subpanel. Once enabled, this subpanel permits us to assign a color to the selected node to further increase the readability in the **Node Editor** window, as shown in this screenshot:



Using colors to further label the nodes

Creating node groups

Single nodes (shaders, textures, input, or whatever) can be grouped together, and this is probably one of the best ways to organize our workflow.

Thanks to node groups, it's easy to store complex materials in ready-to-use libraries. It's possible to share or reuse them in other files, and they can also be used to build handy shader interfaces for easier tweaking of material properties.

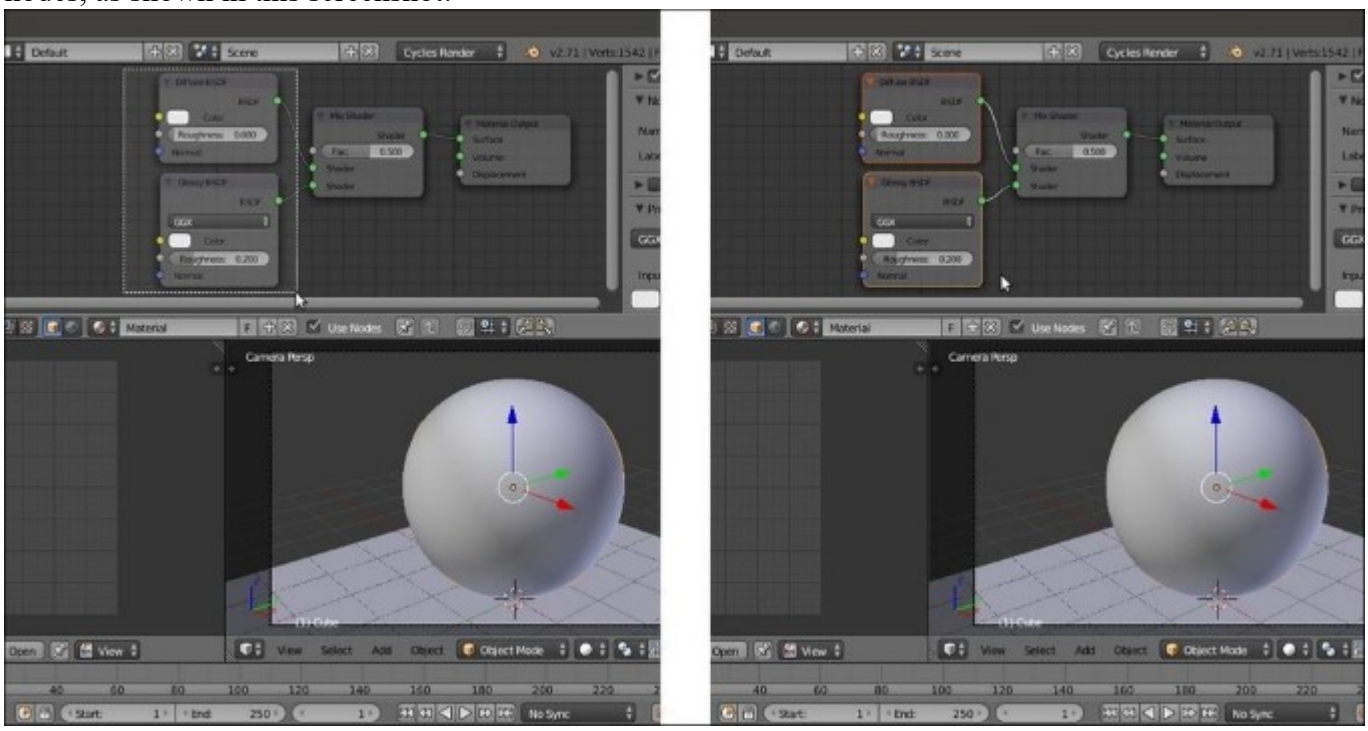
Getting ready

Start Blender and open the `9931OS_02_interface.blend` file.

How to do it...

Let's go to the **Node Editor** window directly:

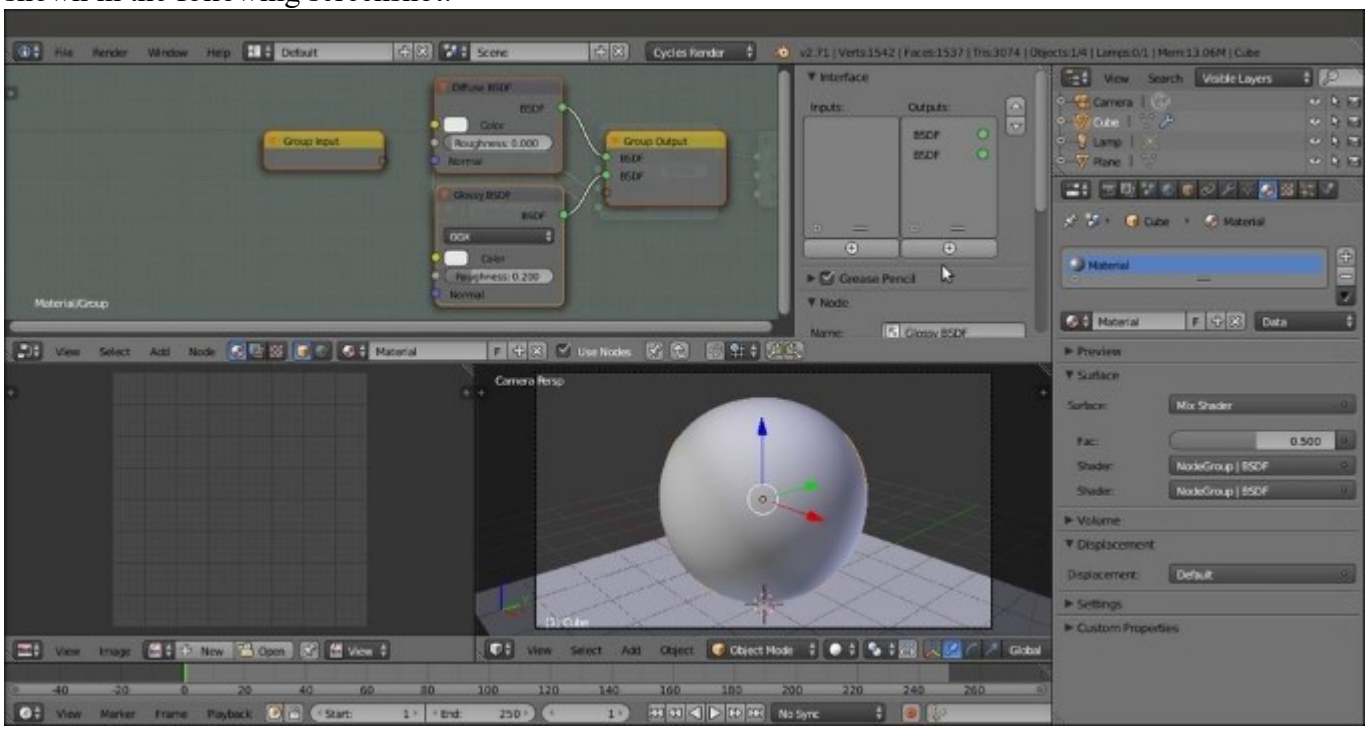
1. Now box-select (place the mouse cursor in the **Node Editor** window, press *B*, and click and drag a box to include the nodes you want to select) the **Diffuse BSDF** and the **Glossy BSDF** nodes, as shown in this screenshot:



Box-selecting two nodes

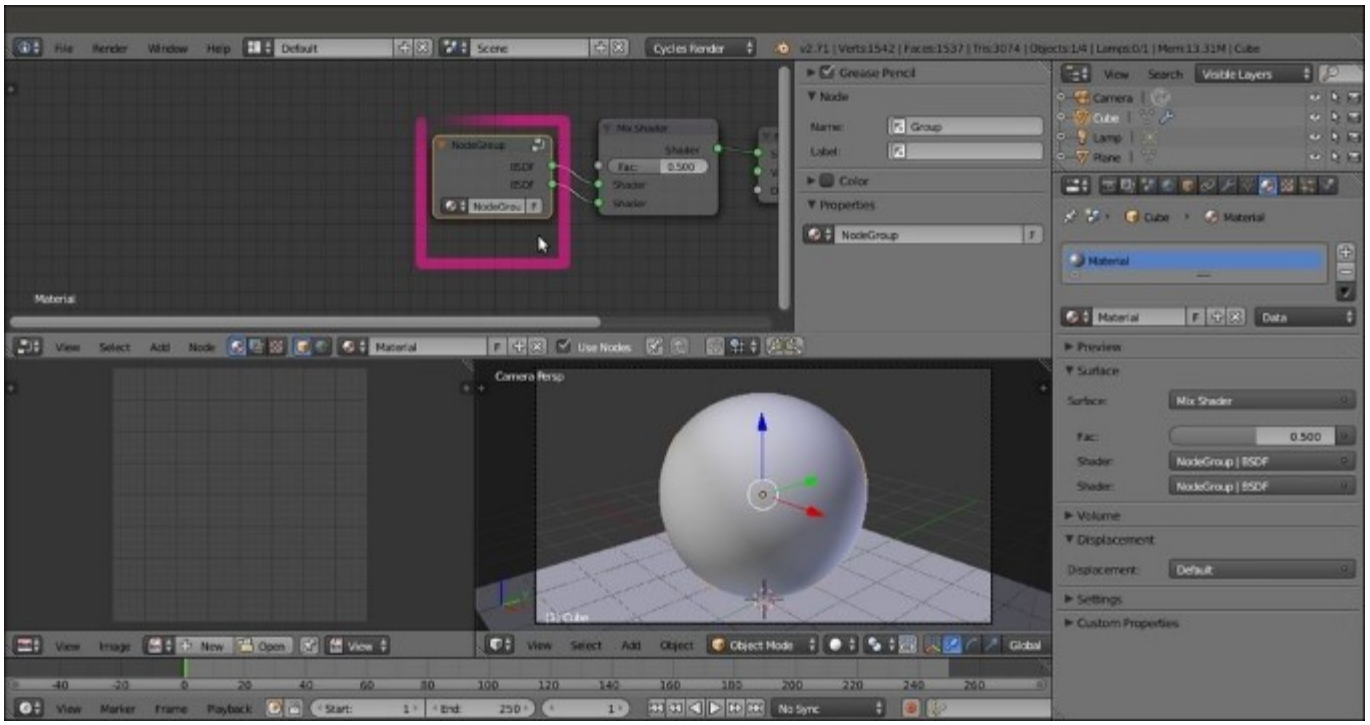
2. Press *Ctrl* + *G* on the keyboard. The background of **Node Editor** changes, showing that now we are in **Edit Mode** inside a group. In fact, there are two selected nodes with a **Group Input** node and a **Group Output** node. Also, the **Surface** subpanel under the **Material** window has

changed, and in the **Node Editor** in the **Properties** panel, a new **Interface** tab has appeared, as shown in the following screenshot:



The appearance of the just created and open for editing node group inside the Node Editor window

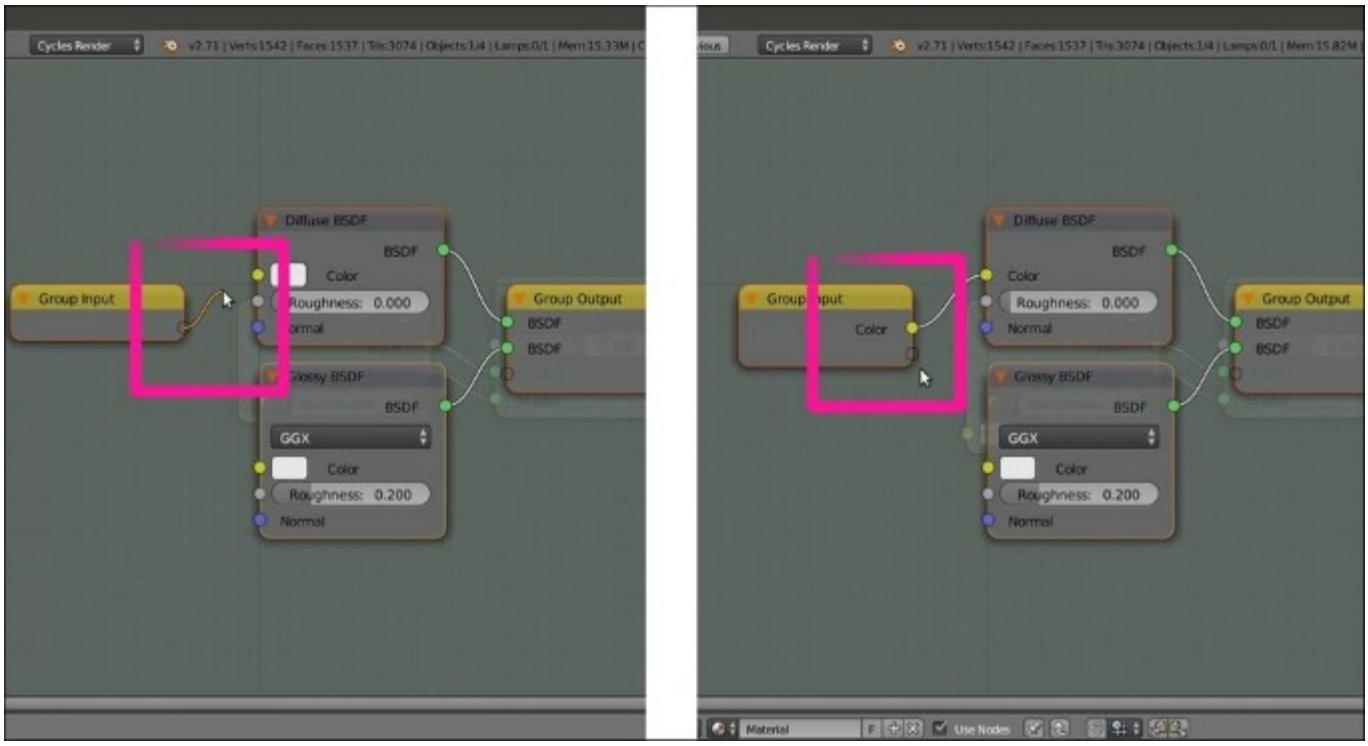
3. Because the two shaders were already connected to **Mix Shader** (which, in this case, we left out of the group on purpose), both the **Diffuse BSDF** and the **Glossy BSDF** outputs are now connected to two **BSDF** sockets automatically created on the **Group Output** node.
4. As for every **Edit Mode** in Blender, by pressing the *Tab* key, we go out of **Edit Mode**, closing the node group, as shown in this screenshot:



The closed node group

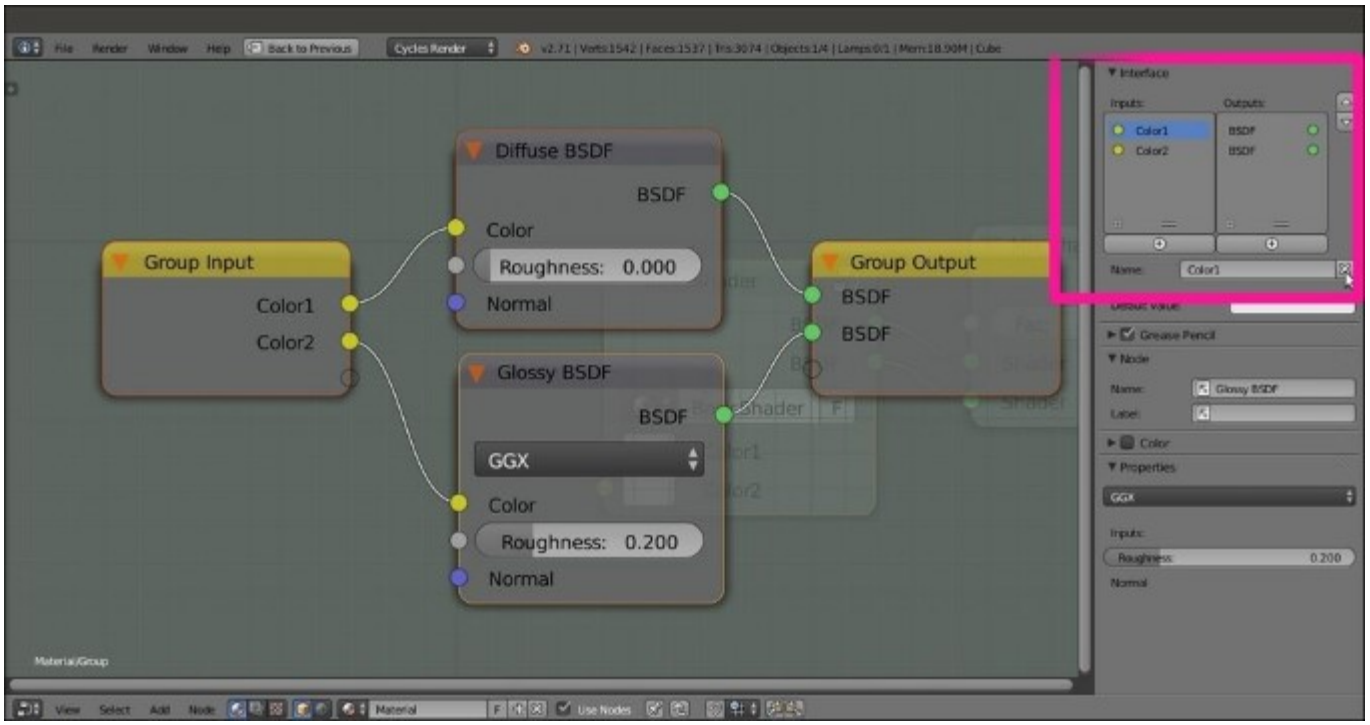
The node group is still showing the two **BSDF** outputs (actually connected to the input sockets of the **Mix Shader** node), the name data block, and the *fake user* button (**F**). This last one is the same as in Blender Internal. It prevents the user count from ever becoming zero, and therefore prevents the deletion of any non-assigned material. When you save the file and/or close Blender by assigning the *fake user* to a non-assigned material, you are sure that it will not be deleted. This is particularly handy when you are building a material library.

5. Now click on the name data block of the node group and change the default name, `NodeGroup`, to something else. I wrote `BasicShader`.
6. Press *Tab* and go to **Edit Mode** again. Click on the only empty socket in the **Group Input** node and drag a link to the **Color** input socket of the **Diffuse BSDF** node. The empty socket now connected to the **Diffuse BSDF** node has changed and is now indicated as **Color**. Moreover, a new empty socket has appeared on the **Group Input** node, as shown in the following screenshot:



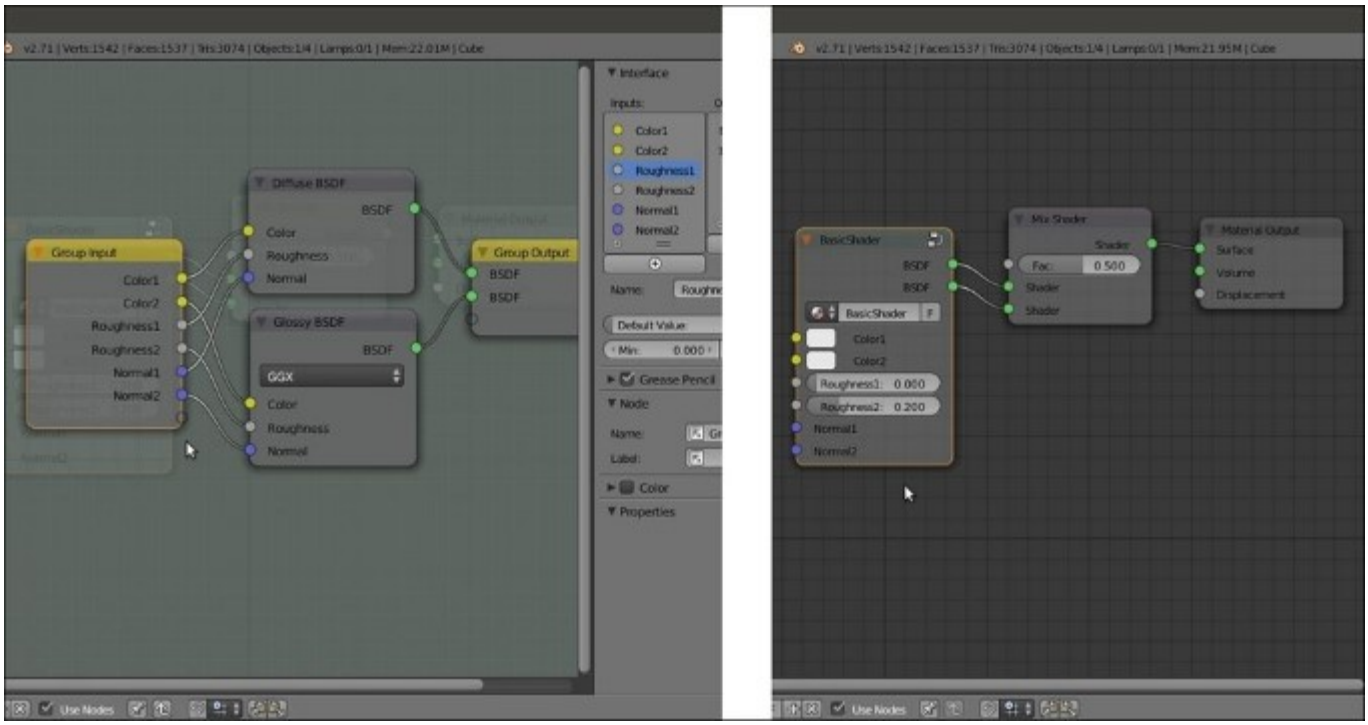
Editing the node group by connecting inner sockets to expose them

7. Repeat the previous step for the new empty input socket and connect it to the **Color** input socket of the **Glossy BSDF** node. Again, a new empty socket has appeared, ready to be connected to something else.
8. Now look at the **Properties** panel to the right. The **Interface** subpanel is reflecting what we are doing in the **Node Editor**. In fact, in the little **Input** window, there are two **Color** sockets, and we can double-click to rename them (`Color1` and `Color2` in our case). To remove a socket, just click on the name in the **Properties** panel, and then click on the **X** icon in the bottom **Name** slot, as shown in this screenshot:



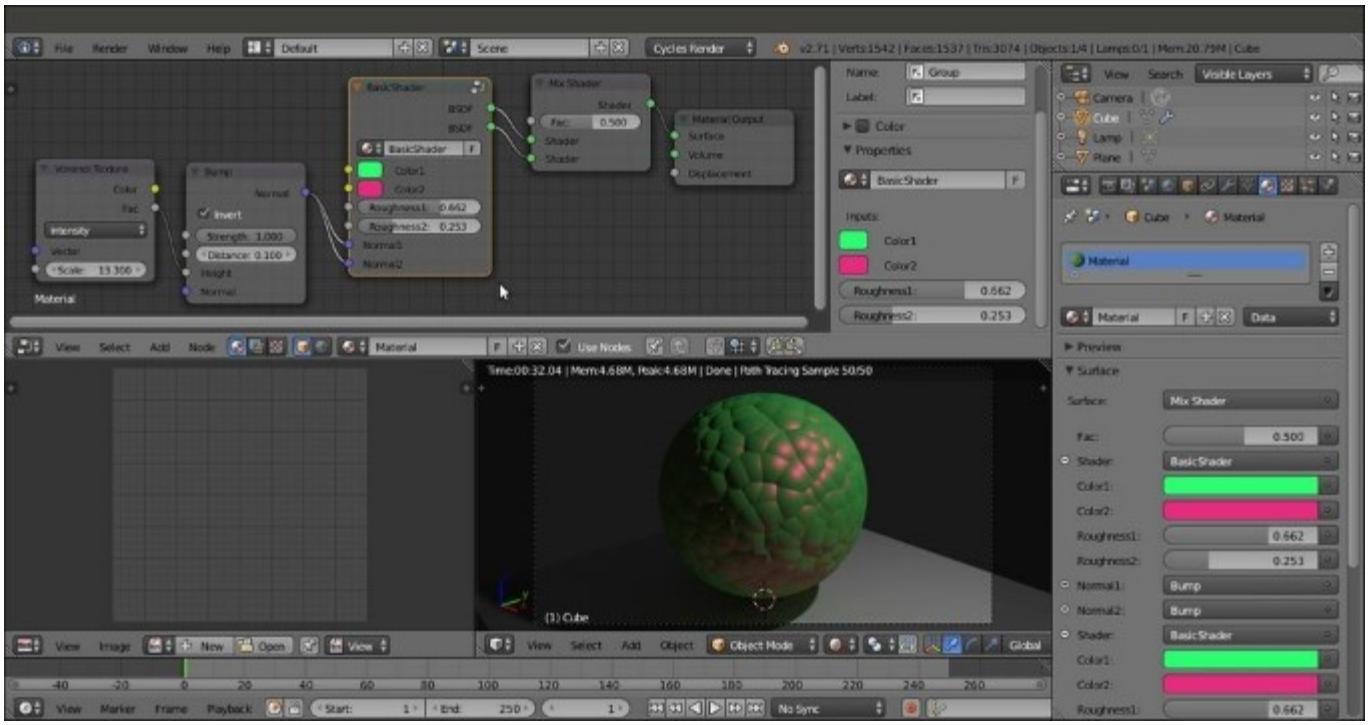
Renaming and ordering the new input sockets through the Interface subpanel in the Properties side-panel

9. Repeat the process to create input sockets for other properties of the **Diffuse BSDF** and **Glossy BSDF** nodes. Then press *Tab* to exit Edit Mode, which is also shown in the following screenshot for your reference:



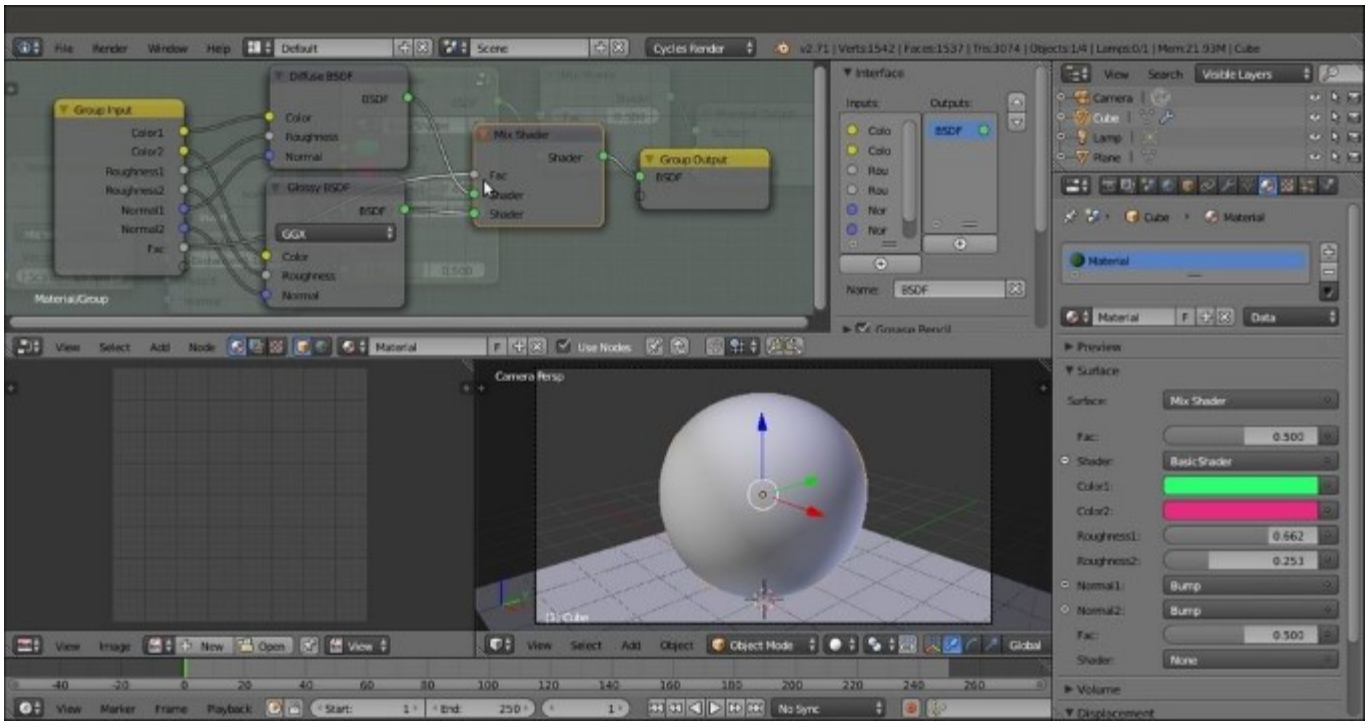
The appearance of the inner connections in the open node group and as exposed input sockets in the closed node group

Here, we get a simple interface for the **BasicShader** node group, and as you can see in the following screenshot, the exposed values can be tweaked. Also, the properties are driven by textures exactly as in other nodes:



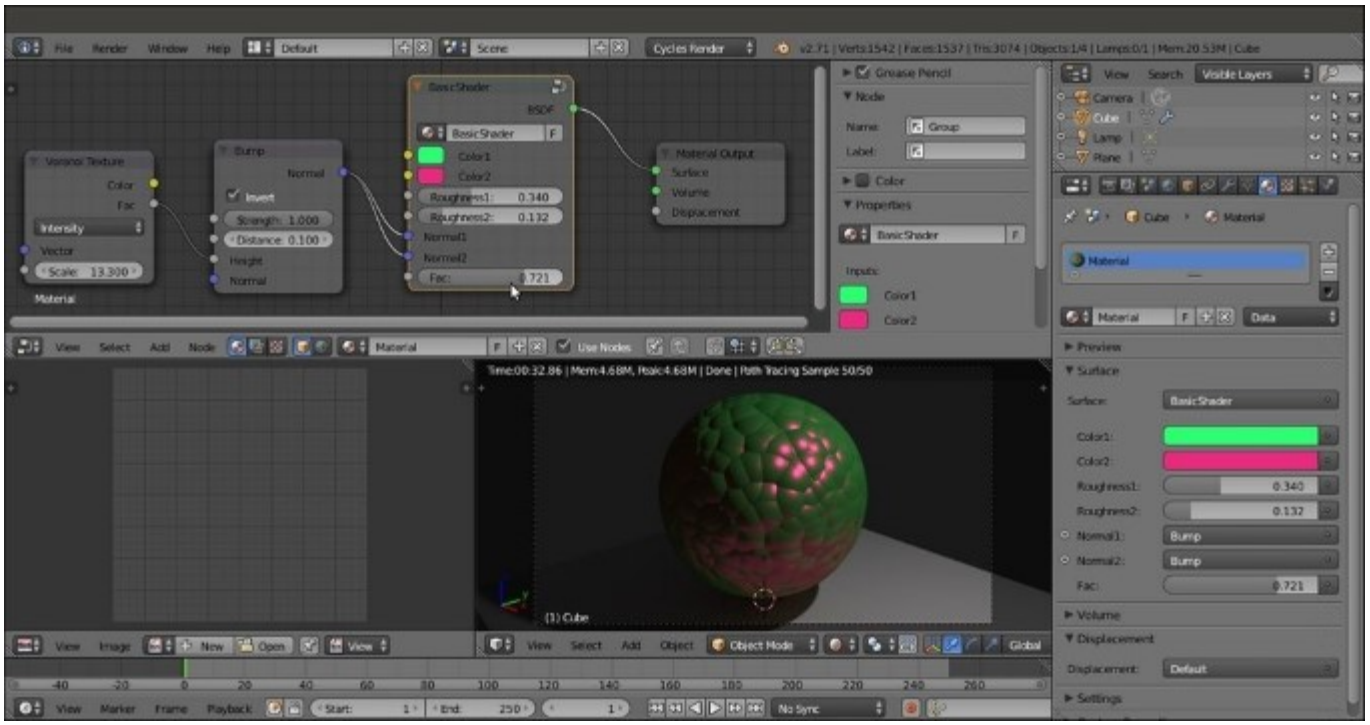
The BasicShader node group put to use

10. Press *Tab* again to go back to **Edit Mode**. Move the mouse cursor into the node and press *Shift* + *A* to add a **Mix Shader** node to the group (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**).
11. Connect the **Diffuse BSDF** and the **Glossy BSDF** shaders to the new **Mix Shader** node and its output to one of the **BSDF** sockets. Delete the other node by clicking on the **X** icon in the **Properties** panel. Connect the empty socket of the **Group Input** node to the **Fac** input socket of the **Mix Shader** node, as shown in this screenshot:



Adding a Mix Shader node inside the node group and one more exposed socket

12. Exit **Edit Mode** and select the outer **Mix Shader** node. Press *Alt + D* (this shortcut removes a node from a network, leaving the connection untouched) to disconnect it and then delete it, or simply press *Ctrl + X* to delete it, leaving the connection untouched. This is shown in the following screenshot:



The final interface of the BasicShader node group

How it works...

I think you get the picture. Basically, almost any input or output socket of the nodes wrapped inside a group can be connected to the outside of the node group to be tweaked.

The good thing about a node group is that you can make instances of that node (by pressing *Shift* + *D*). Note that when you modify the inner structure of a node group, the modifications get reflected in all the group instances, but the outer (exposed) values on the node group interface are local for each instance and can be individually tweaked.

Every newly created node group is available in both the **Add** menu (press *Shift* + *A*) and in the slots in the **Material** window of the **Properties** panel, under the item **Group**, to be added on the fly to the network.

To remove a node group, select it and press *Alt* + *G*. This will break the node envelope but keep the content intact and still connected.

Grouping nodes under frames for easier reading

The shaders we have seen so far are quite simple and easily readable in the **Node Editor** window, but for several materials we'll see in this Cookbook, the node connections will be a lot more complex and confusing. One more aid we can use to improve the readability of these nodes are the frames. We can use them to visually organize the shaders' network.

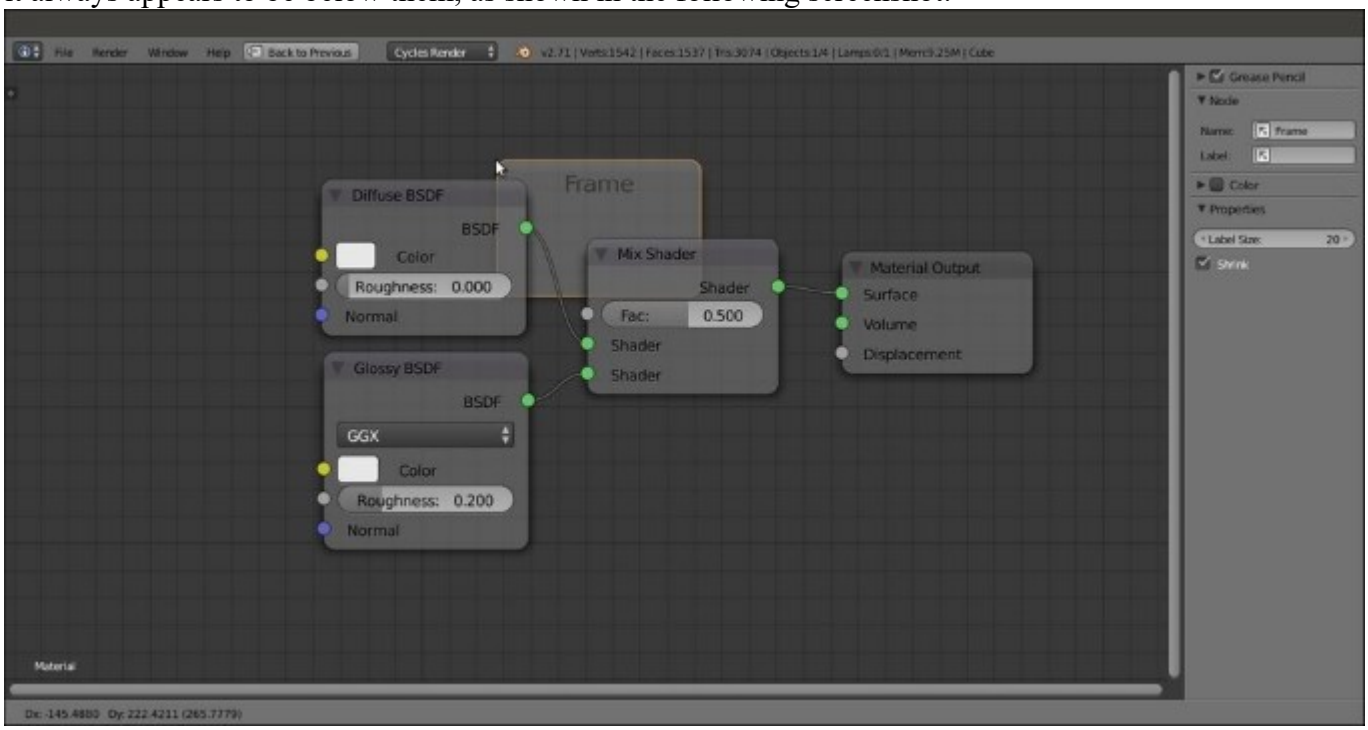
Getting ready

Start Blender and open the `9931OS_02_interface.blend` file.

How to do it...

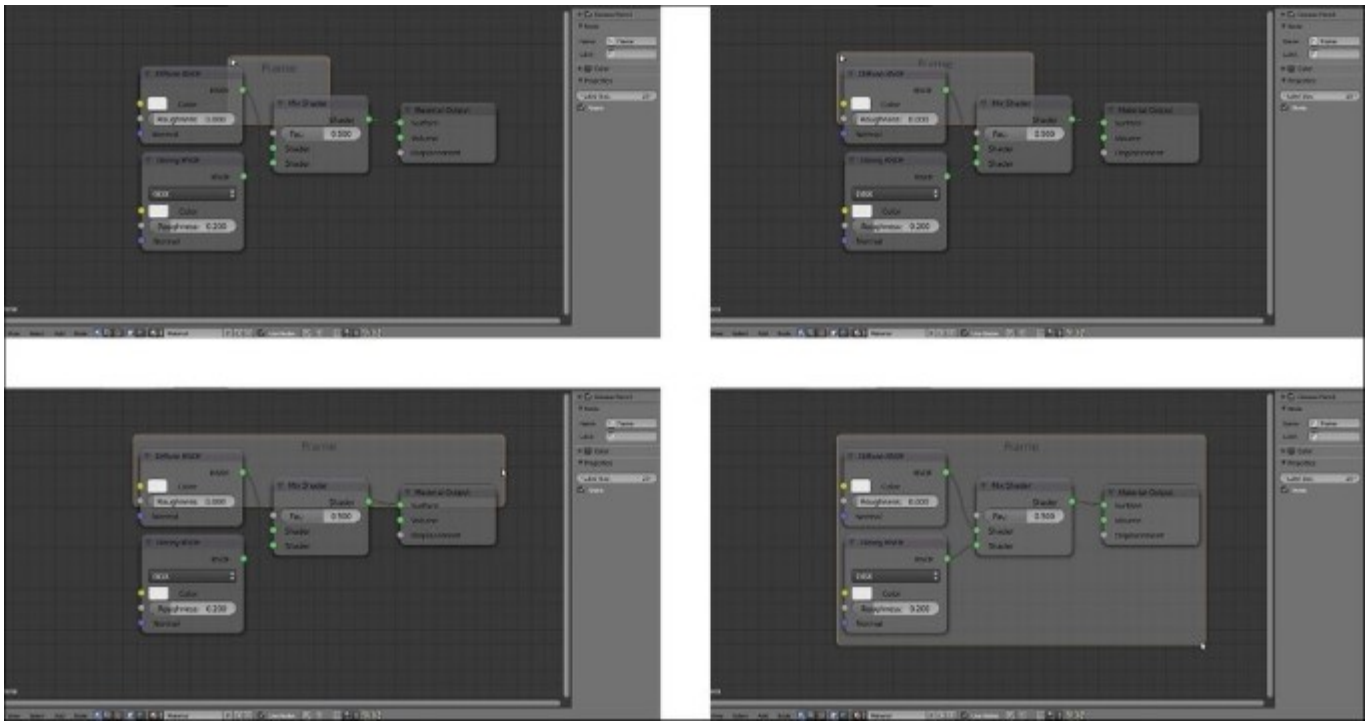
Let's see the use of a frame with a simple shader that we already know:

1. Go to the **Node Editor** window and press *Shift* + *A* to add a **Frame** (press Shift + A and navigate to Layout | Frame). Move the mouse to place its arrow over the nodes, and notice that it always appears to be below them, as shown in the following screenshot:



Adding a Frame to the network inside the Node Editor window

2. Move the mouse cursor to a corner of the frame. It turns into a cross, which means that you can click and drag to resize the corners and the sides of the frame to include the desired nodes as shown in the following screenshot:

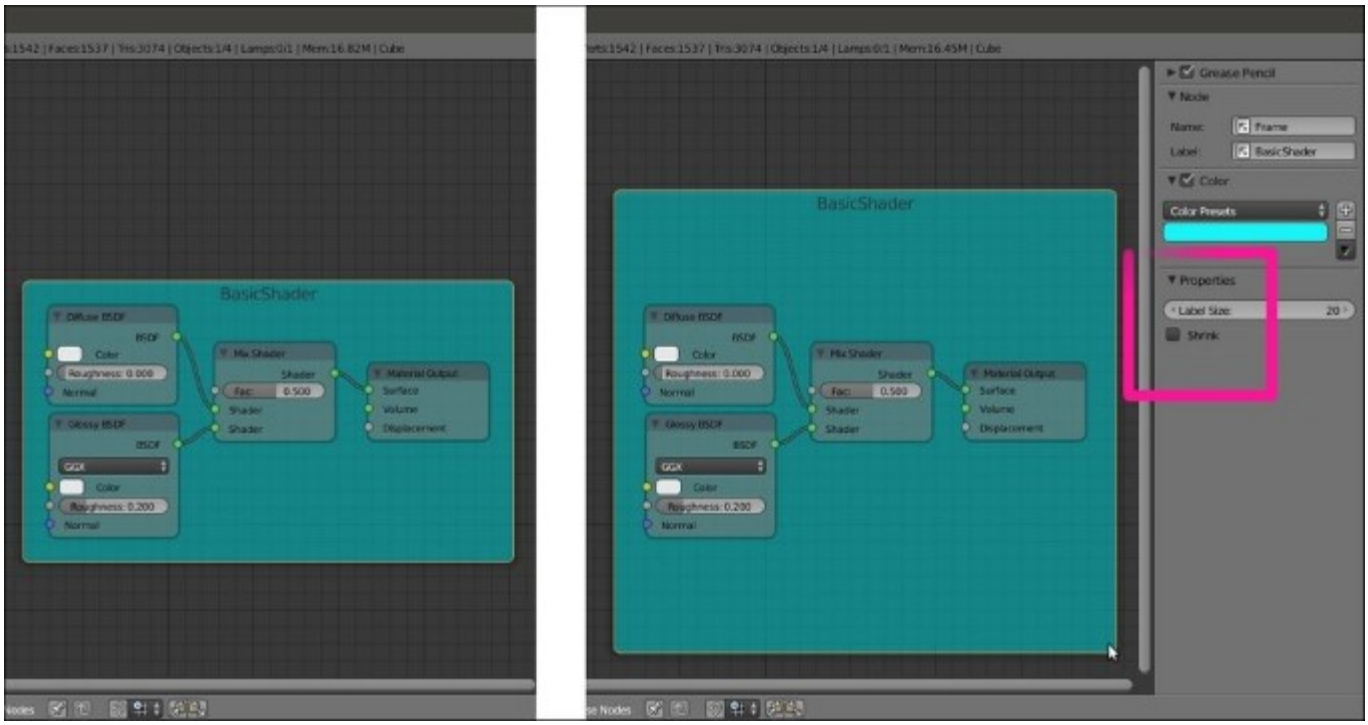


Resizing the Frame to comprise all the nodes

3. Box-select the nodes you want to arrange with the frame (in our case, all of them), then press *Shift*, and select the frame (or just press *A* to deselect the frame and box-select everything so that the frame is selected again as the active object). Press *Ctrl* + *P* to parent them.

   Now that the nodes are parented to the frame, we can select and move it, and all the contained nodes will follow it as a single object.

   It's still possible to select the single nodes inside the frames and arrange their individual position and connections. To add a new node to the network, we will do as usual (press *Shift* + *A* | …) and then parent it to the frame as well.
4. With the frame still selected, go to the **Properties** panel to the right of the **Node Editor** window (Press *N* if not already present). Just like the case of single nodes, in the **Node** subpanel, we can change the **Name** of the frame, assign a **Label** name visible in the **Node Editor** window (I wrote `BasicShader`), and also assign a color.
5. In the **Properties** subpanel right below the **Color** subpanel, we can change the size of the label name, which is set to `20` by default, and by unchecking the **Shrink** item, we can freely resize the frame bigger, which otherwise encloses the nodes with a fixed boundary (the default setting), which is shown in the following screenshot for your reference:



The Frame with label and color

Linking materials and node groups

Similar to Blender Internal, Cycles materials can be linked from libraries. Every blend file containing linkable assets can be a library.

Linking materials is really useful practice. Let's say you have 20 different blend files with objects using an `iron` shader, and at a certain point of your workflow, you need to modify this `iron` material in all the files. By having this material linked in all the 20 files from a single blend, it is possible to update all of them at once by modifying just one shader in the library file (as you know, a linked material reflects the properties of the library material and cannot be edited, differently from an appended material that is local to the file where it has been imported from the library file).

How to do it...

Start Blender, go to the **File** menu in the left part of the main header, and select **Link**, or press *Ctrl* + *Alt* + *O*. Then perform the following steps:

1. Browse to the directory where you store your library files. Select the blend file you want to link the material from; for example, try the provided `9931OS_02_library.blend` file.
2. Browse inside the blend structure, where the linkable assets are divided into subdirectories, shown as folders named `Scene`, `Mesh`, `Material`, `NodeTree`, `Object`, and so on. Note that the various folders appear only if the corresponding asset to be linked actually exists inside the blend file.
3. Click on the `Material` subdirectory. Once inside it, select the material you want to link (for example, `Brainy_blue`) and press *Enter* to confirm (or click on the **Link/Append from Library** button in the top-right corner), as shown in the following screenshot:

4. Now click on the **Material datablock** button on the toolbar of the **Node Editor** window and select the name of the linked material—the material labeled with a **LF** prefix; L is for linked and F is for *fake user*.
5. This is because, in the library file, we assigned the *fake user* to the material by clicking on the **F** icon on the side of the material name data block. If not assigned to any *fake user*, the prefix of the linked material would have been **L0**, that is, linked and zero fake users inside the blend file (for example, Plane is simply assigned to the object and has no *fake user*).

   The name of the material is grayed to show that it is a linked material. On the side of the name, a new icon has appeared (a little arrow), and the number of users has been updated to **2** (the *fake user* and the object we assigned the linked material to).

From now on, every modification we make to the material in the library will be reflected in the linked material the moment we load the file.

Not only whole materials but also single node groups can be linked. In this case, instead of the `Material` subdirectory to link from, choose the `NodeTree` subdirectory and then select one or more node groups you want to link.

The data block name of a linked node group is grayed as well. You can modify the exposed values and colors, and you can also enter **Edit Mode**, but that's all. You can't modify the connection or the nodes inside the linked node group. To do this, you have to click on the little arrow icon to the side of the grayed name to make it local and no longer linked from the library file. This would mean that from now on, you have a new independent node group, and that editing the node group in the library won't have any effect on it any more.

There's more...

A very useful add-on to help in node management is the **Node Wrangler** add-on. It allows for effects such as quick material visualization, node switching, UV layer assignment, frame assignments, node arrangements, and so on. To find out more about this add-on, go to http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Nodes/Node_Wrangler.

To enable it, just call the **Blender User Preferences** panel (press *Ctrl* + *Alt* + *U*) and click on the **Node** tab under the **Categories** item. Enable the **Node Wrangler (aka Node Efficiency Tools)** add-on by clicking on the checkbox to the right. Then click on the **Save User Settings** button to the bottom-left corner of the panel.

Chapter 3. Creating Natural Materials in Cycles

In this chapter, we will cover the following recipes:

- Creating a rock material using image maps
- Creating a rock material using procedural textures
- Creating a sand material using procedural textures
- Creating a simple ground material using procedural textures
- Creating a snow material using procedural textures
- Creating an ice material using procedural textures

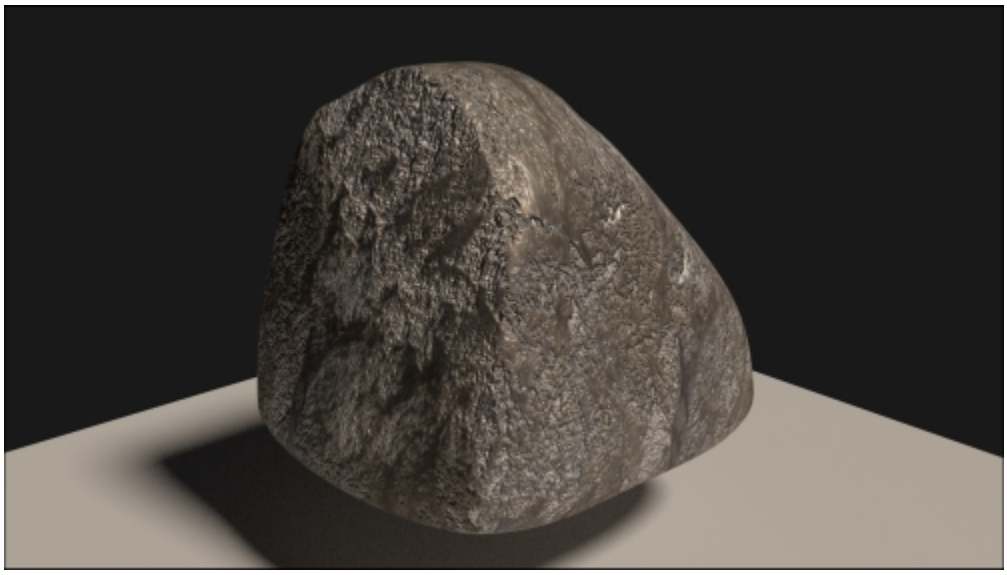
Introduction

Replicating nature can be difficult. Natural materials are usually the most difficult to recreate in a satisfying way using computers, mainly because the chaos of nature is not the best fit for the orderly logic of an electronic machine.

Too often, we even see cubes that look obviously computer-generated because of the neatness and regularity of their shapes or surfaces. Actually, in reproducing true to life a natural object (or material as well), we have to start from the absolute regularity of the computer simulation, and then blemish it step by step in a controlled way to reach a more natural look.

Creating a rock material using image maps

In this recipe, we will create a realistic rock material that looks like the samples shown in the following screenshot, using an image map. We'll see a material made with procedurals in the this recipe.



The image-based rock material as it appears in the final rendering

Image maps are particularly useful for several reasons: they already have the necessary color information of a natural surface ready to be used; they can be easily edited in any image editor to obtain different kinds of information, for example, high levels for the bump maps; and they are processed faster than procedurals by the software (procedural textures must be calculated every time). Moreover, they can nowadays be found easily on the Web for free, in several sizes and resolutions.

For our recipe, we'll use the `rockcolor_tileable_low.png` image map, which you can find in the `textures` folder provided with this cookbook. This is just a low-resolution texture image provided for the sake of this exercise. Obviously, you can use any other different image with a bigger resolution. Here is a screenshot of the tileable rock image texture for your reference:

The tileable rock image texture provided with this cookbook

In any case, for any image you are going to use, just remember to make it tileable in your preferred image editor. In GIMP, this task can be automatically done by a plugin that can be found by navigating to **Filter** | **Map** | **Make Tileable**.

Getting ready

Start Blender and load the `9931OS_02_interface.blend` file, which saw in the previous chapter. Remember that all the blend files and the textures needed for the exercises of this module can be downloaded from the support page on the Packt Publishing website.

Now, we'll create a new file named `9931OS_start.blend`, and we'll be using it as the starting point for the most of our recipes. To do so, perform the following steps:

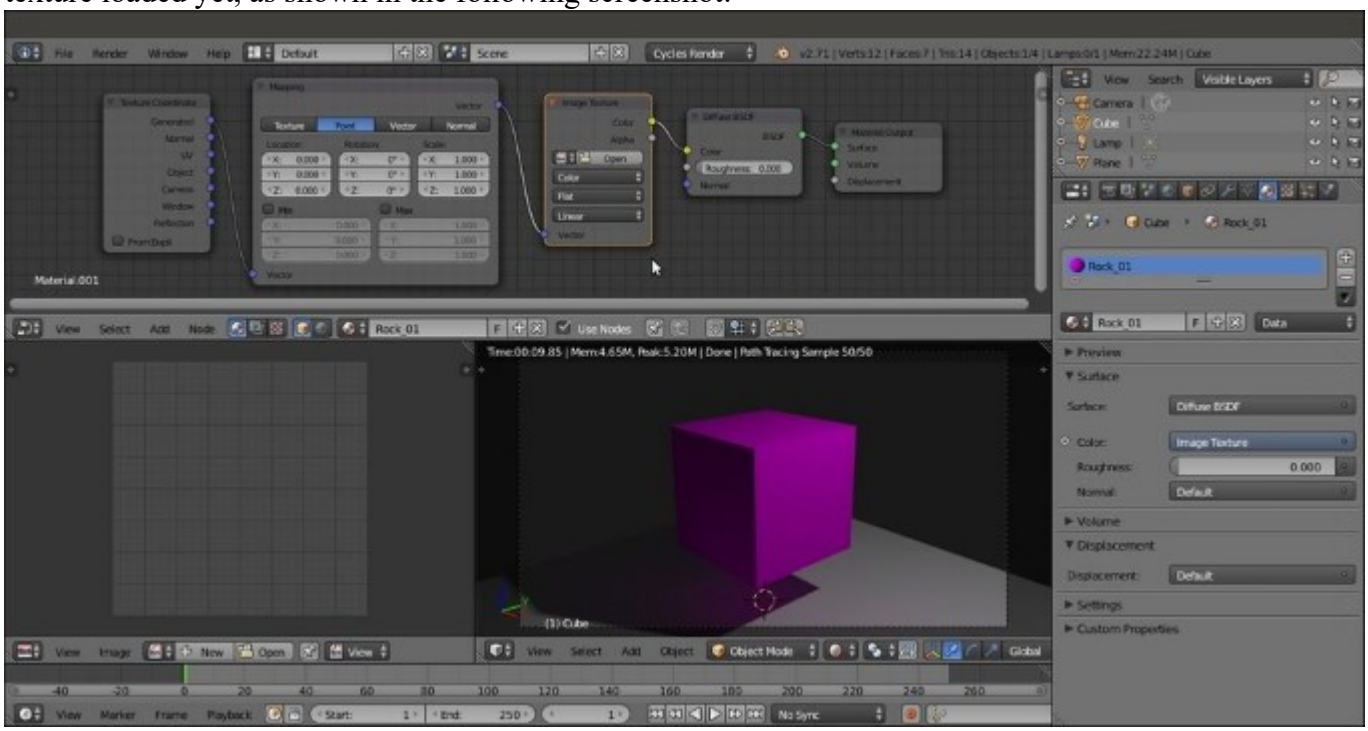
1. Select the Spheroid and (just for the purpose of this exercise) delete it by pressing *X*.
2. Ensure that the 3D cursor is at the center of the scene (*Shift* + *C*), and with the mouse pointer in the 3D window, press *Shift* + *A* to pop up the **Add** menu. Then add a new Cube primitive (press *Shift* + *A* and navigate to **Mesh** | **Cube**).
3. Press *Tab* to go out of **Edit Mode** if needed (this depends on whether you are using Blender with the factory settings or not), and move the cube 2 units up on the *z* axis.
4. Go to the **Outliner** and select the **Lamp** item, in the **Object data** window under the **Properties** panel, switch the **Type of Lamp** to **Sun**. Then set the **Strength** to `2.000` and the color values to `1.000` for **R**, `0.872` for **G**, and `0.737` for **B**.

5. Reselect the Cube, go to the **Material** window under the **Properties** panel, and save the file as `9931OS_start.blend`.

How to do it...

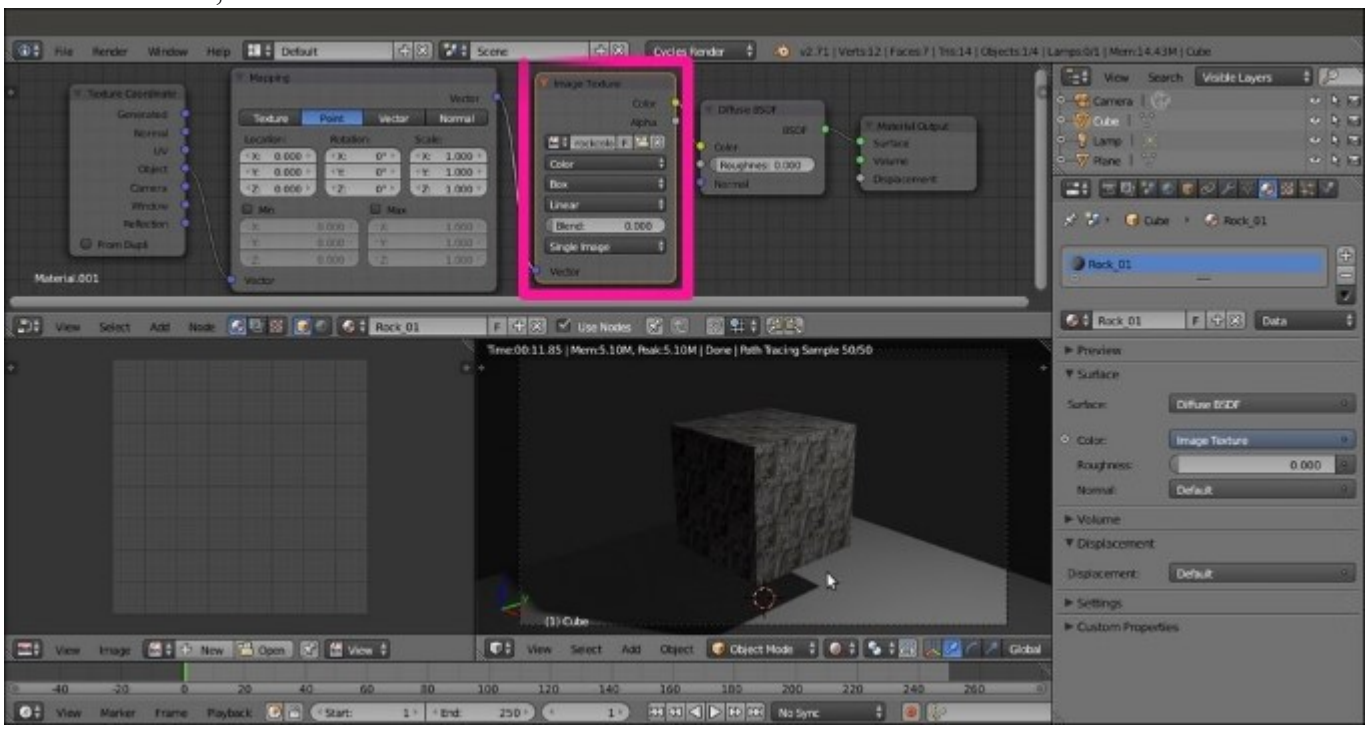
Now carry out the following steps to create the rock material:

1. Click on the **New** button in the **Material** window under the **Properties** panel, or in the toolbar of the **Node Editor** window.
2. Rename `Material.001` as `Rock_01` (the numbering is because I assume that you are going to experiment with several values, especially colors, producing more and different kind of rock materials) and save the file as `9931OS_03_Rock_imagemap.blend`.
3. Put the mouse on the **Node Editor** window and add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**). Then add a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**).
4. Connect the **Generated** output socket of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and its **Vector** output to the **Vector** input of the **Image Texture** node. Also connect the **Color** output of the **Image Texture** node to the **Color** input of the **Diffuse BSDF** shader node.
5. Set the **Viewport Shading** mode of the **Camera** view to **Rendered** by pressing *Shift* + *Z* with the mouse cursor in the 3D window. The rendered Cube turns pink because there is no image texture loaded yet, as shown in the following screenshot:



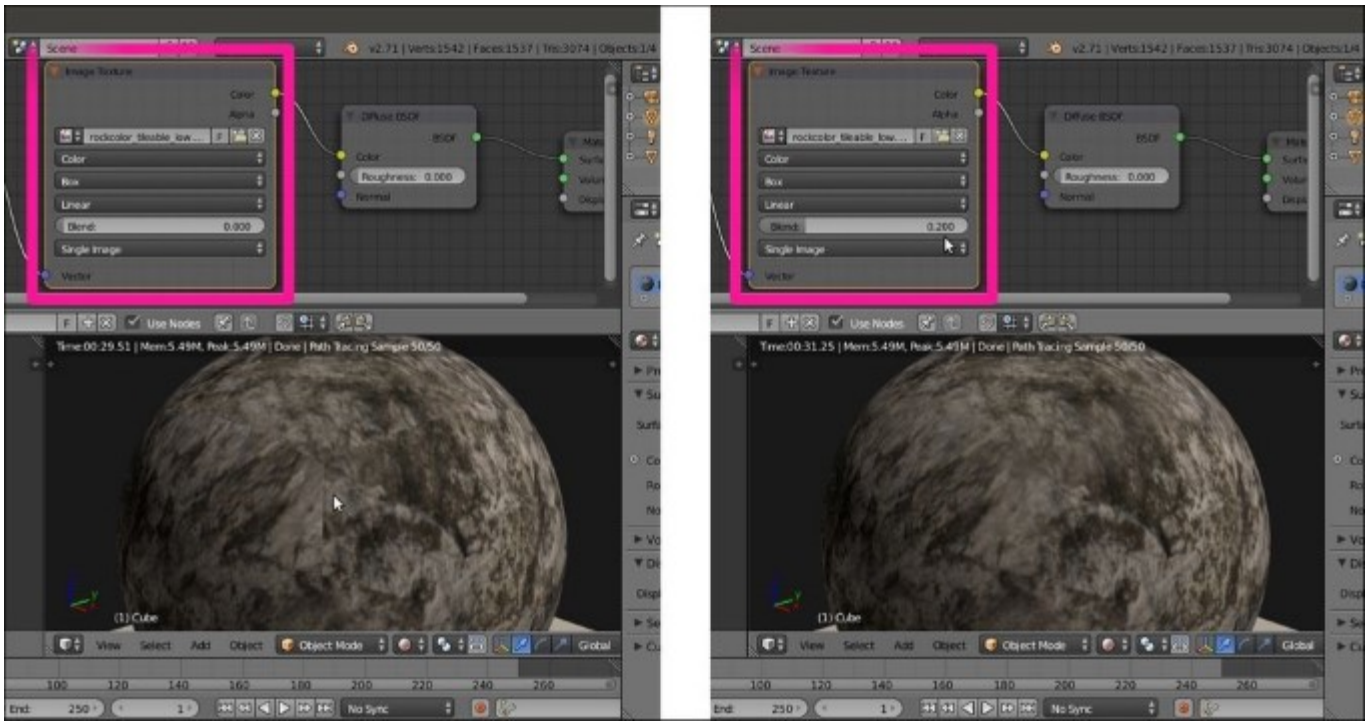
The overall view of Blender's customized Default screen with the rendered preview of the pink Cube

6. Click on the **Open** button in the **Image Texture** node, browse to the `textures` folder, and select the `rockcolor_tileable_low.png` image.
7. As we selected **Generated** as the mapping mode, the image is mapped flat on the Cube from the *z* axis and appears stretched on the sides. Disconnect the **Generated** output of the **Texture Coordinate** node and connect the **Object** node instead. Then click on the **Flat** button on the **Image Texture** node to select the **Box** item. The texture looks now correctly mapped on each face of the Cube, as shown in this screenshot:



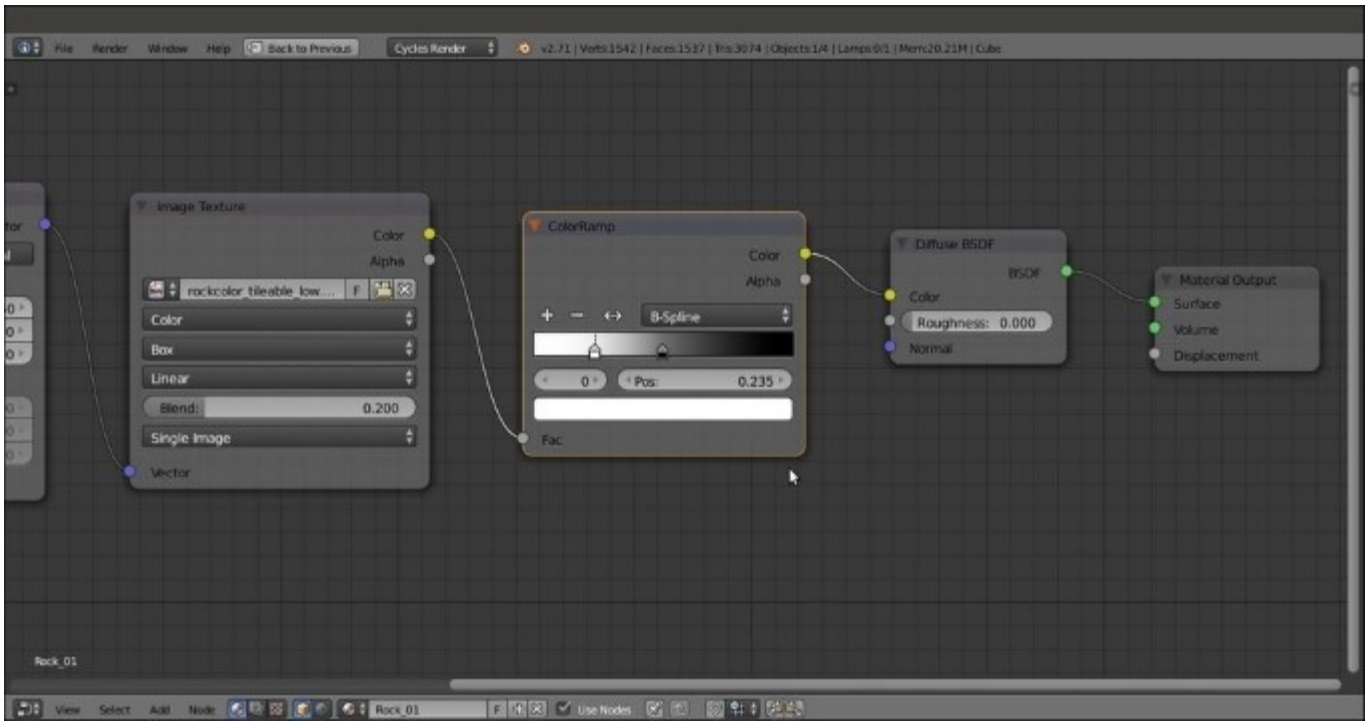
The rock image texture loaded in the Image Texture node

8. Go to the **Object modifiers** window and assign a **Subdivision Surface** modifier to the Cube. Set the **Subdivisions** levels to `4` both for **View** and **Render**. Then check the **Optimal Display** item.
9. Press *Shift* + *Z* to go out of the viewport's **Rendered** mode. Then press *Tab* to go to **Edit Mode** and scale all the vertices to double (press *Tab*, then press *S*, enter the digit *2*, and finally, press *Enter*). Now, go out of **Edit Mode**.
10. Press *T* to make the **Tool Shelf** panel appear in the **Camera** view, and click on the **Smooth** button under the **Tools** tab. Press *T* again.
11. Go to the **Mapping** node in the **Node Editor** window and set the **Scale** values for **X**, **Y**, and **Z** to `0.250`.
12. Although the image map we used is tileable, there are visible seams at the corners of the subdivided Cube. In the **Image Texture** node, set **Blend factor** to `0.200` to soften the seams (this factor is used to blend the edges of the faces of the Cube that, remember, is still a six-faced solid at its lower level, though looks like a Spheroid as of now). The output of blurring effect of the Blend factor is shown in the following screenshot:



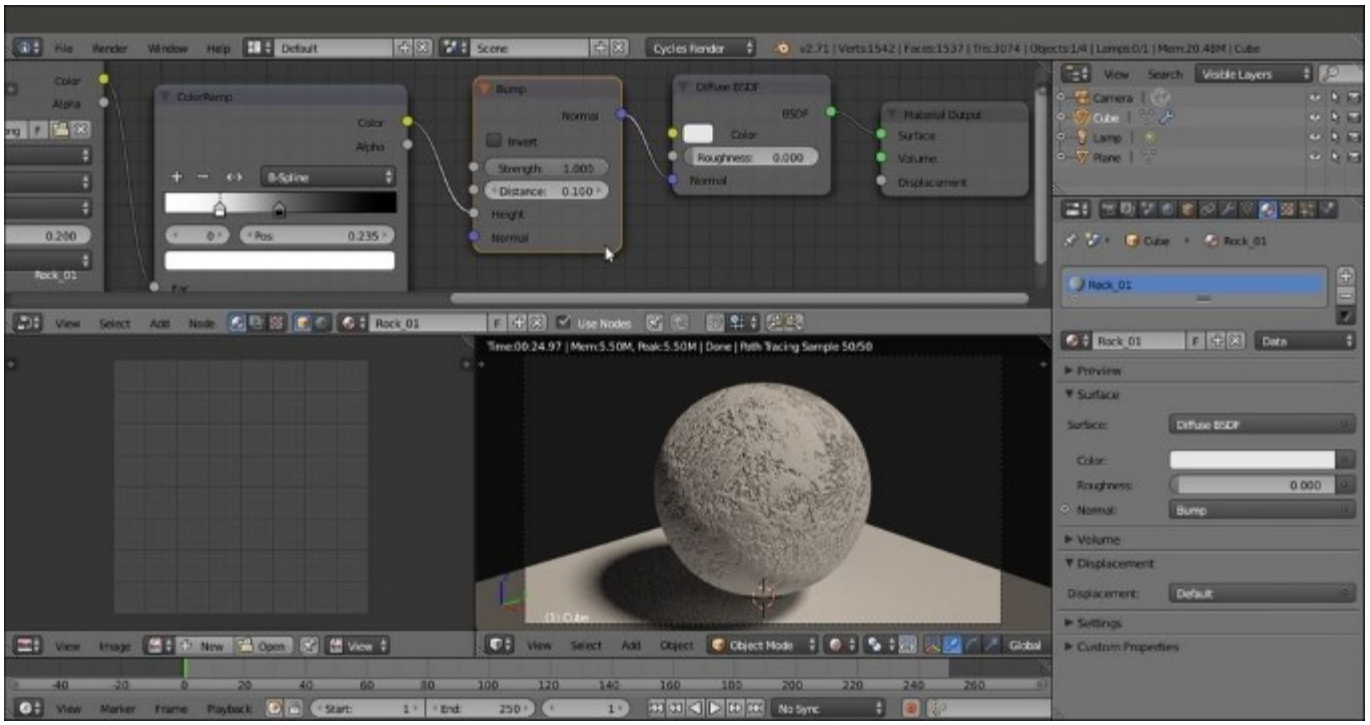
The edges seams visible on the surface of the subdivided Cube, and the blurring effect of the Blend factor

13. Now add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) between the **Image Texture** node and the **Diffuse BSDF** shader. Set the interpolation to **B-Spline**, move the marker of the black color to position (**Pos:**) `0.495` and the white marker to position `0.235`, as shown in this screenshot:



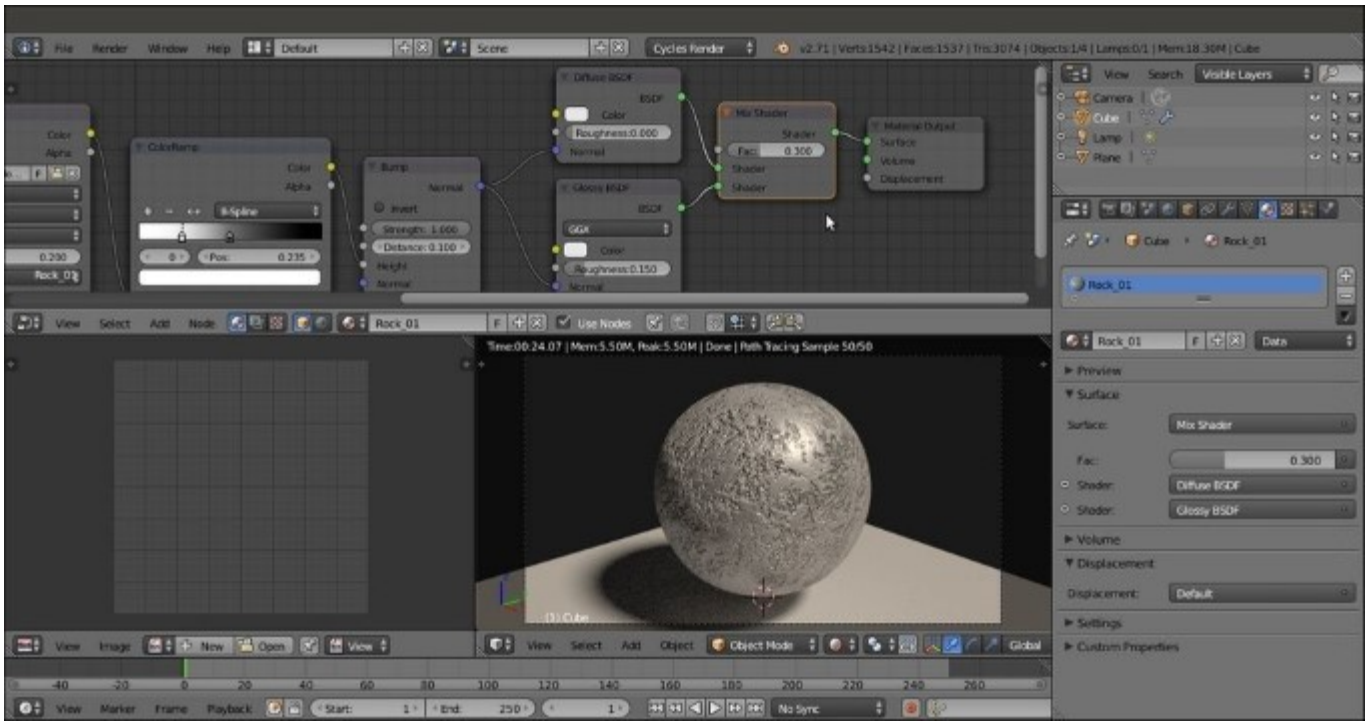
The ColorRamp node pasted between the Image Texture and the Diffuse BSDF nodes

14. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Color** output of the **ColorRamp** to the **Height** input of the **Bump** node, and the **Normal** output of the latter to the **Normal** input of the **Diffuse BSDF** shader. Detach the **Color** link from the **Color** input of the **Diffuse BSDF** node and leave the **Bump** node's **Strength** value to `1.000`, as shown in the following screenshot:



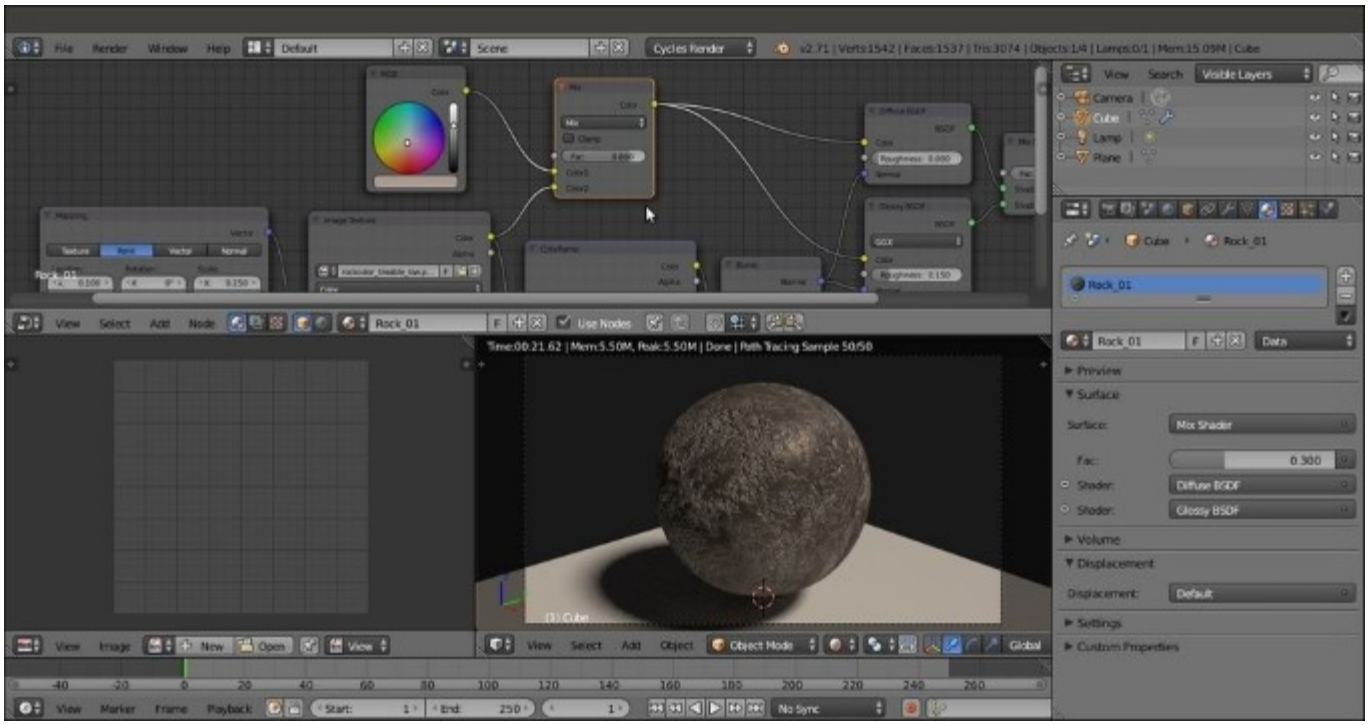
The Bump node pasted between the ColorRamp and Diffuse BSDF nodes

15. Add a **Mix Shader** node and a **Glossy BSDF** shader, and connect them to be mixed with the **Diffuse BSDF** shader. Set the **Glossy BSDF** node's **Roughness** value to `0.150` and the **Mix Shader** node's **Fac** value to `0.300`. Connect the **Normal** output of the **Bump** node to the **Normal** input socket of the **Glossy BSDF** shader, as shown in this screenshot:



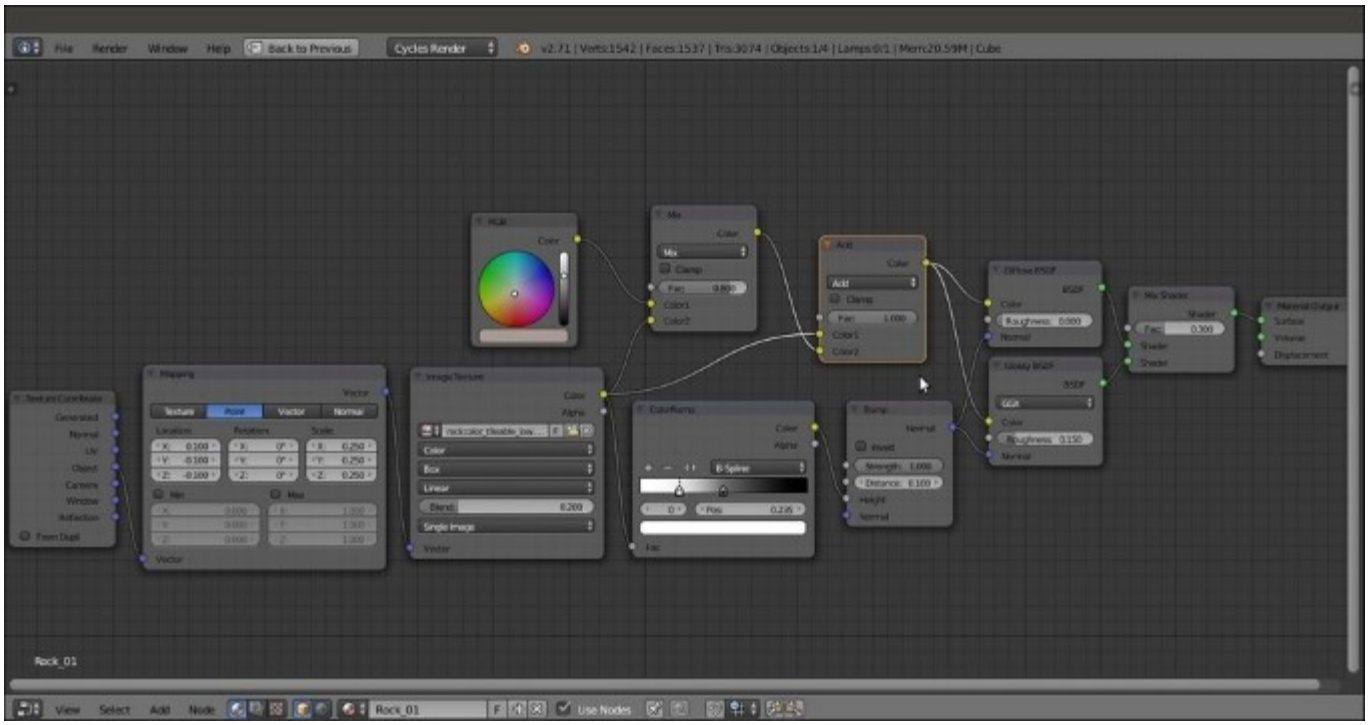
Adding the Mix Shader and the Glossy BSDF nodes

16. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Connect both the **Color** outputs of the **RGB** and **Image Texture** nodes to the **Color1** and **Color2** input sockets of the **MixRGB** node. Connect the **Color** output of the **MixRGB** node to the **Color** input sockets of the **Diffuse BSDF** and **Glossy BSDF** shaders.
17. Set the **Fac** value of the **MixRGB** node to `0.800`. Click on the color slider of the **RGB** node and set the values to `0.407` for **R**, `0.323` for **G**, and `0.293` for **B**, as shown in the following screenshot:



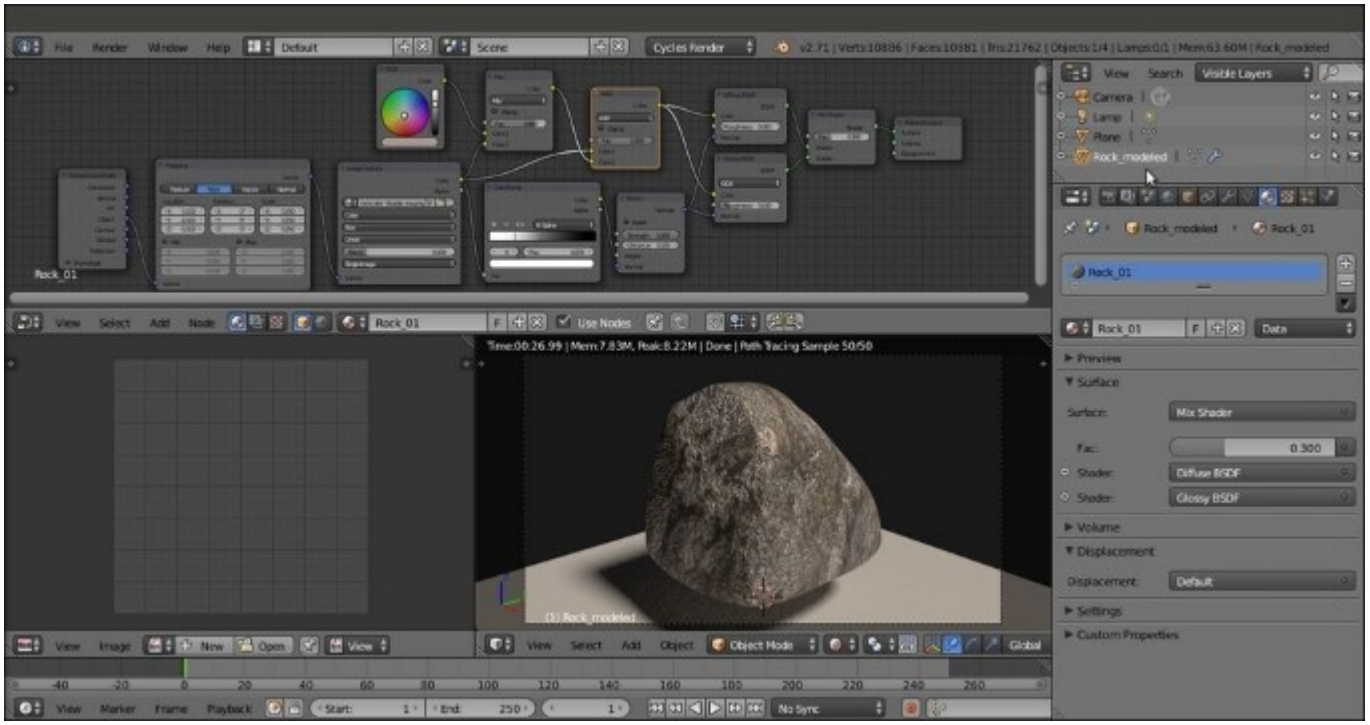
Setting the color for the material

18. Add a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Drag it to be pasted between the first **MixRGB** node and the **Diffuse BSDF** shader, and set **Blend Type** to **Add**.
19. Connect the **Color** output of the second **MixRGB** node (which we can call the **Add-MixRGB** node) to the **Color** input socket of the **Glossy BSDF** shader node, and set its **Fac** value to `1.000`.
20. Connect the **Color** output of the **Image Texture** node to the **Color1** input socket of the **Add-MixRGB** node so that the preceding connection coming from the first **MixRGB** node (which we can call **Mix-MixRGB**) switches automatically to the **Color2** input socket of the **Add-MixRGB** node, as shown in this screenshot:



Adding more variations to the rock color

21. If you wish, model a very quick rock mesh by sculpting or deforming the subdivided Cube in proportional **Edit Mode**, and assign to it the `Rock_01` material as shown in the following screenshot:



How it works...

We mapped a colored image of a rock with the **Box** option available in the **Image Texture** node (developed by the Project Mango team for open movie production of *Tears of Steel* to quickly map objects without the need to unwrap them), and set the **Blend** factor to `0.200` to have smooth transitions at the corners. Although we had a tileable image texture, this has been necessary because we set the **Scale** values for the three axes in the **Mapping** node to `0.250`.

First, by connecting the **MixRGB** node's **Color** output directly to the **Color** input of the **Diffuse BSDF** shader node, we had a quick visual feedback of the image mapping, and thanks to the **ColorRamp** node, we achieved the following goals:

- We converted the colored image to a gray-scale image to be used for the bump.
- By moving the color markers, we remapped the values of the **ColorRamp** node's position values to reverse and increase the contrast (we could have obtained the same result by processing the color map in GIMP, for example, by desaturating it and playing with the curve tool). In any case, it's possible to visualize the **ColorRamp** node itself on the object by temporarily connecting it to the **Color** input socket of the **Diffuse BSDF** shader or an **Emission** shader node connected to the **Material Output** node.

This contrasted result has been applied as a bump map to both the **Diffuse BSDF** and **Glossy BSDF** shaders.

Then we mixed a brownish color (the **RGB** node) with the **Color** output of the image of the rock, and the result was added to the **Image Texture** node's output.

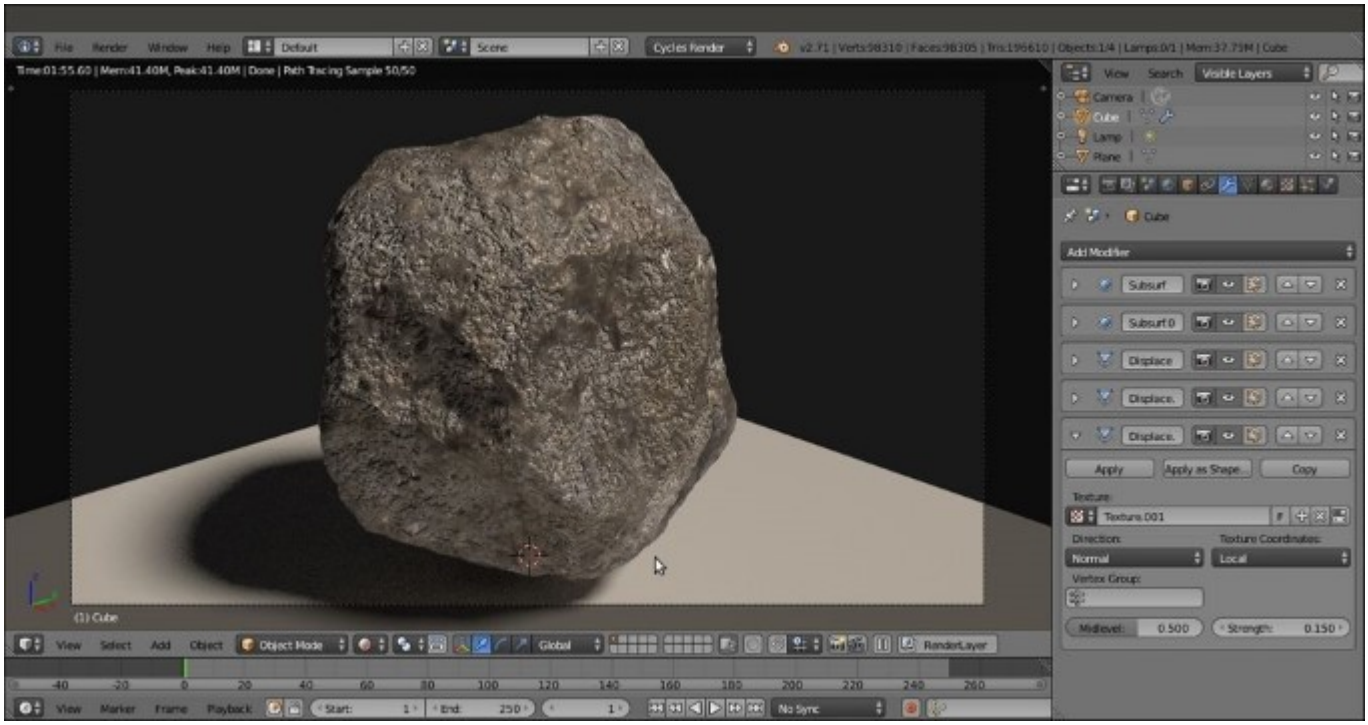
There's more...

We can improve the rocky effect by adding displacement to the geometry. Unlike bump or normal effects on the mesh surface, which are just optical illusions giving an impression of perturbing the mesh surface, displacement is an actual deformation of the mesh based on the gray-scale values of a texture.

At least in this case, there is no need for precise correspondence between the already textured surface and the displacement because it would be barely noticeable. Therefore, we can use object modifiers to obtain a fast but effective result, by performing the following steps:

1. Starting from the `Rock_imagemap.blend` file we just created, select the Cube and go to the **Object modifiers** window under the **Properties** panel. In the **Subdivision Surface** modifier already assigned, lower the **Subdivisions** levels for both **View** and **Render** to `3`.
2. Add a new **Subdivision Surface** modifier and set the levels to `4`.
3. Now add a **Displace** modifier. Click on the **Show textures in texture tab** button, the last button on the right of the **Texture** slot. This switches to the **Texture** window, where we can click on the **New** button and then change the default **Clouds** texture to a **Voronoi** texture node. Set the **Size** value to `2.00` and leave the rest of the values unchanged. Go back to the **Object modifiers** window and set the modifier's **Strength** value to `0.800`.

4. Add a new **Displace** modifier. Switch to the **Texture** window and assign a new **Voronoi Texture** node. Change the **Size** value to `1.20`. Back in the modifier, set the **Strength** value to `0.300`.
5. Add one more **Displace** modifier. This time, we are going to use the default **Clouds** texture as it is. Just go to the **Object modifiers** window and set the **Strength** value to `0.150`, as shown in this screenshot:

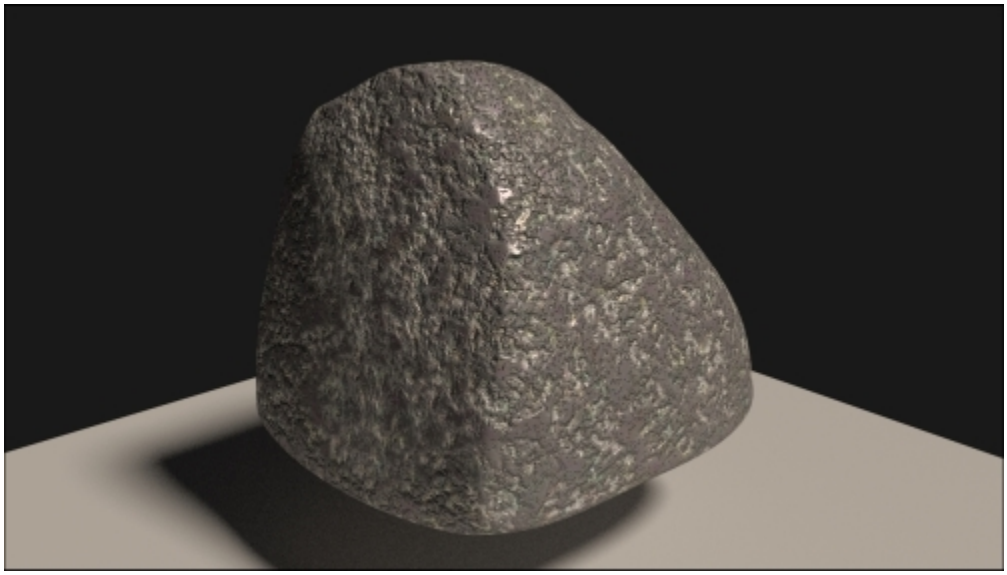


A different rock model, thanks to displacement

Of course, these are just basic values. You can change them and also play with different kinds of procedural textures to obtain several rock shapes.

Creating a rock material using procedural textures

In this recipe, we will try to reach a result similar to the rock material we made through image maps in the previous recipe, but using only procedural textures. The output will look like what is shown in the following screenshot:

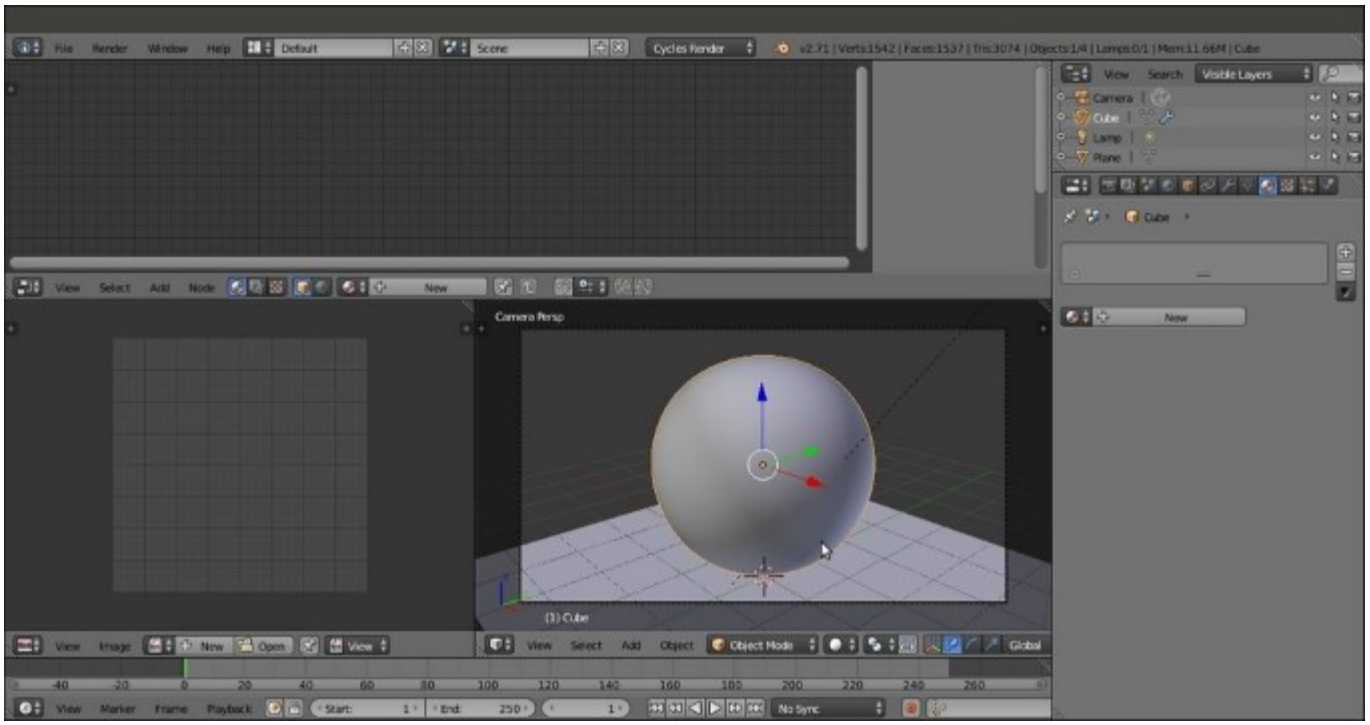


The procedural rock material as it appears in the final rendering

Getting ready

Start Blender and load the `9931OS_start.blend` file.

1. Select the Cube, go to the **Object modifiers** window, and assign a **Subdivision Surface** modifier. Set the **Subdivisions** levels for **View** and **Render** to `4`. Go back to the **Material** window.
2. Press *T* and in the **Tool Shelf**, click on the **Smooth** button under the **Shading** subpanel. Press *T* again to get rid of the **Tool Shelf**.
3. Press *Tab* to go to **Edit Mode**. If necessary, select all the vertices by pressing the *A* key and scale everything to double the current size (press *S*, enter the digit *2*, and press *Enter*). Go out of **Edit Mode**.
4. Save the file as `9931OS_start_smoothed.blend`. The customized Default screen will now look as shown in the following screenshot:

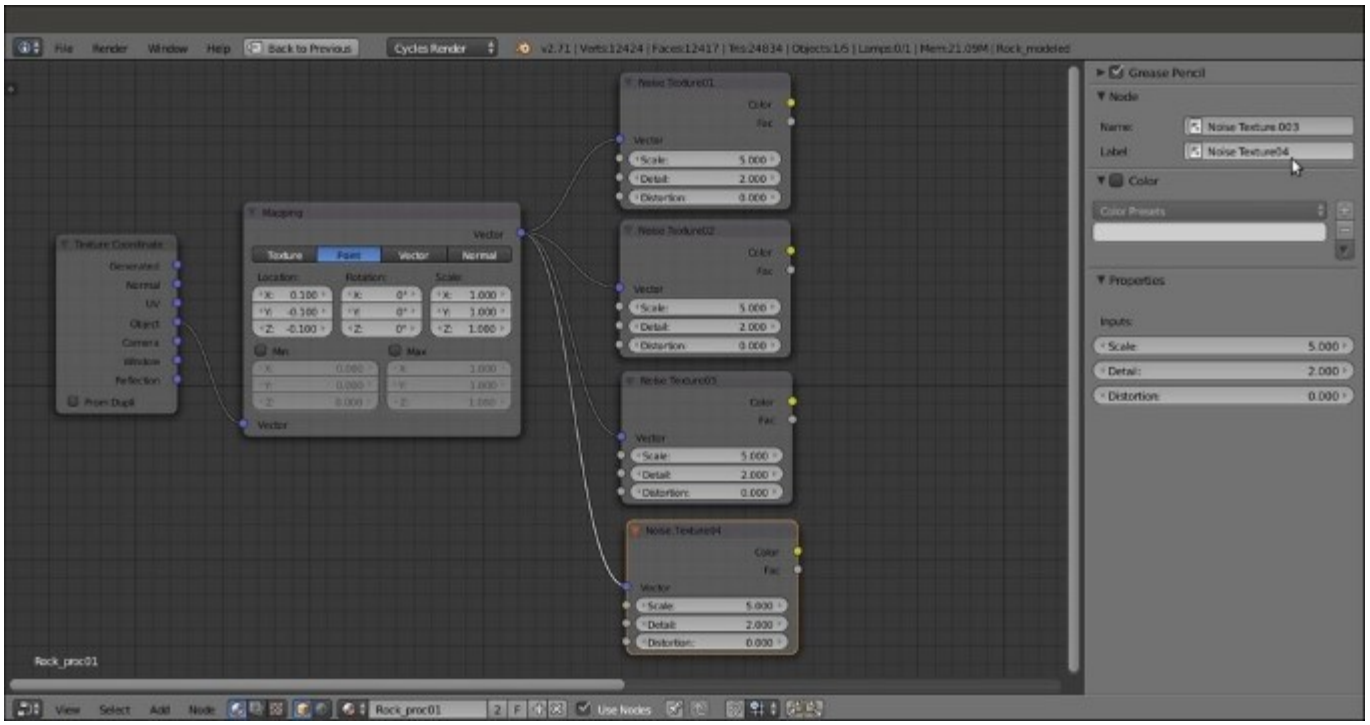


The customized Default screen with the subdivided Cube

How to do it...

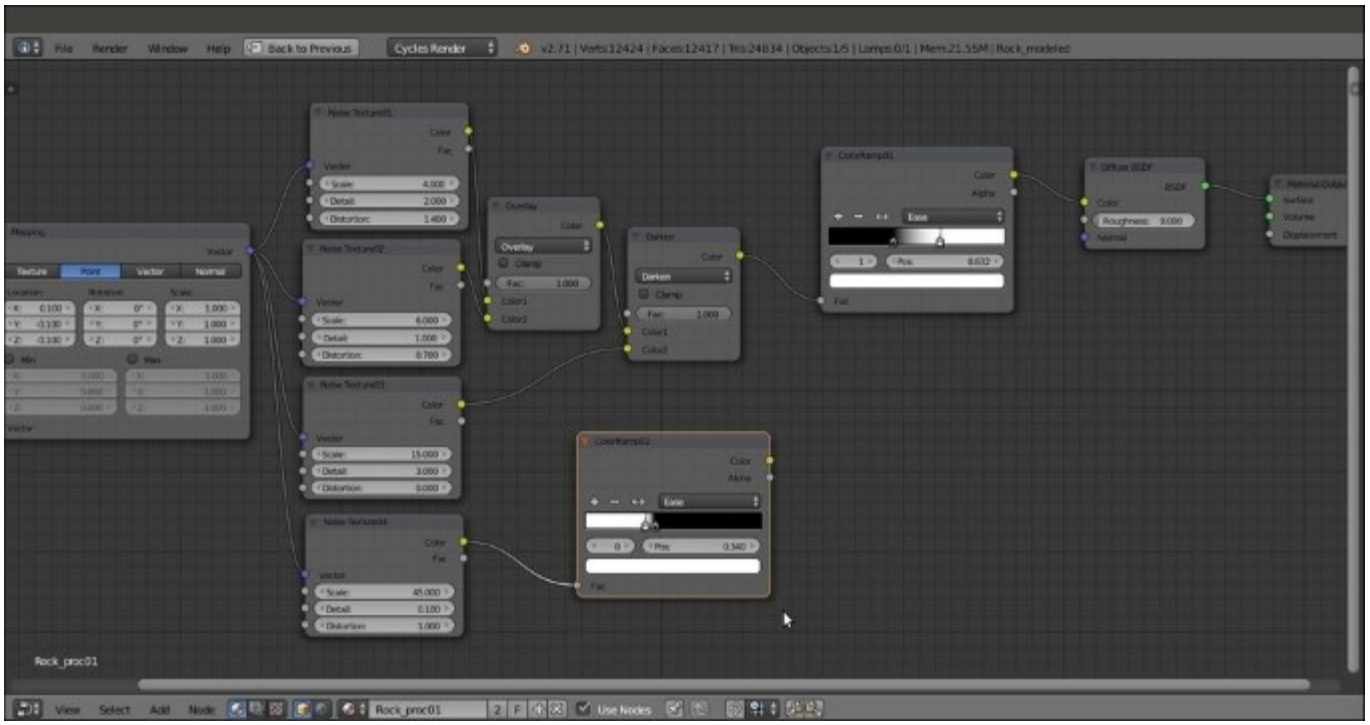
Now we are going to create the rock material by performing the following steps:

1. Select the Spheroid (the smoothed Cube) and click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Rock_proc_01`.
2. In the **Node Editor** window, add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**). Then press *Shift* + *D* to duplicate it three times. Adjust the four **Noise Texture** nodes in a column, and in the **Properties** panel to the right (press *N* key if it is not already present), label them `Noise Texture01`, `Noise Texture02`, `Noise Texture03`, and `Noise Texture04`.
3. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **Object** output of the **Texture Coordinate** node to the blue **Vector** input of the **Mapping** node. Then connect the **Mapping** node's **Vector** output to the **Vector** input sockets of the four texture nodes. Set the **Mapping** node's **Location** values to `0.100` for **X** and `-0.100` for **Y** and **Z**, as shown in the following screenshot:



The first steps to build the bump effect for the rock material

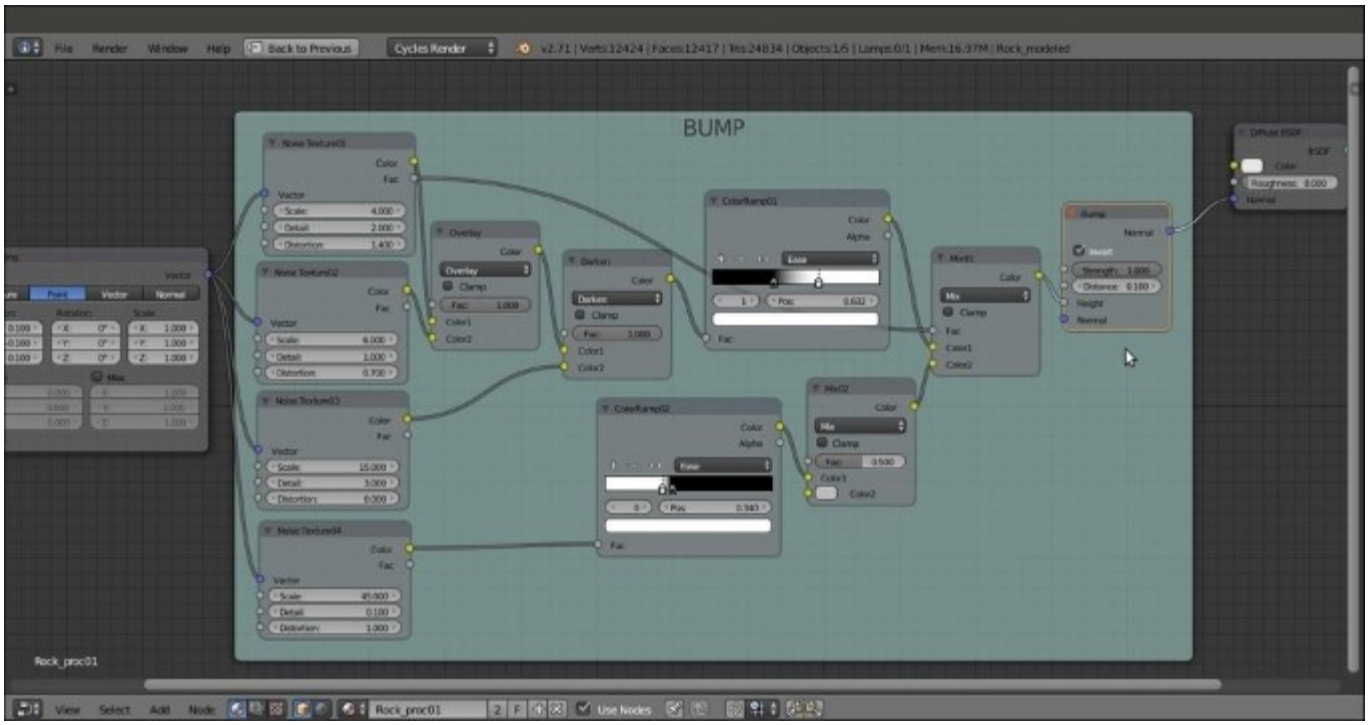
4. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Connect the first two **Noise Texture** nodes' **Color** output to the **Color1** and **Color2** input of the **MixRGB** node. Set **Blend Type** to **Overlay** and the **Fac** value to `1.000`.
5. Connect the **Overlay-MixRGB** node's output to the **Color** input socket of the **Diffuse BSDF** shader node.
6. Put the mouse cursor in the **Camera** view and press *Shift* + *Z* to set it to **Rendered** mode.
7. Go to the **Noise Texture01** node and set **Scale** to `4.000` and the **Distortion** value to `1.400`. Then go to the **Noise Texture02** node and set **Scale** to `6.000`, **Detail** to `1.000`, and the **Distortion** value to `0.700`.
8. Press *Shift* + *D* to duplicate the **MixRGB** node, and paste it between the **Overlay-MixRGB** and the **Diffuse BSDF** nodes. Connect the **Color** output of the **Noise Texture03** node to the **Color2** input socket and set the **Blend Type** to **Darken**. Go to the **Noise Texture03** node, and set **Scale** to `15.000` and **Detail** to `3.000`.
9. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and paste it between the **Darken-MixRGB** and the **Diffuse BSDF** nodes. Label it `ColorRamp01` and set **Interpolation** to **Ease**, the black cursor's position to `0.364`, and the white cursor's position to `0.632`.
10. Press *Shift* + *D* to duplicate the **ColorRamp** node, and move it close to the **Noise Texture04** node. Label the duplicated node `ColorRamp02` and set the white cursor's position to `0.340` and the black cursor's position to `0.400`.
11. Set the **Noise Texture04** node's **Scale** value to `45.000`, **Detail** value to `0.100`, and **Distortion** to `1.000`, as shown in the following screenshot:



Starting to mix the different procedurals together

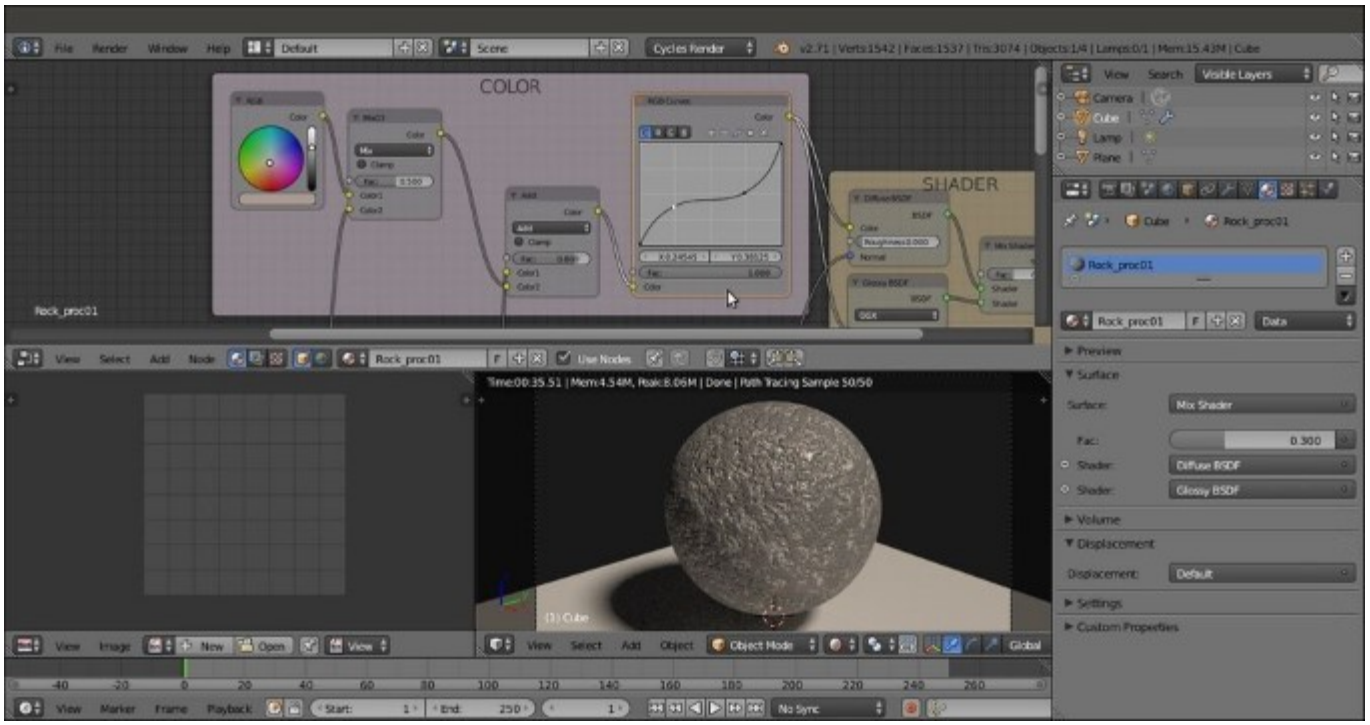
12. Connect the **Color** output of the **Noise Texture04** node to the **Fac** input socket of the **ColorRamp02** node. Then add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and label it `Mix01`.
13. Paste the **Mix01** node after the **ColorRamp01** node. Then connect the **Color** output of the **ColorRamp02** node to the **Color2** input socket of the **Mix01** node. Also connect the **Fac** output of the **Noise Texture01** node to the **Fac** input socket of the **Mix01** node.
14. Add a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and label it `Mix02`. Paste it between the **ColorRamp02** and the **Mix01** nodes.
15. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **Color** output of the **Mix01** node to the **Height** input socket of the **Bump** node. Connect the **Normal** output of the **Bump** node to the **Normal** input socket of the **Diffuse BSDF** shader node. Click on the **Invert** item checkbox.

    The steps that are detailed after this screenshot will result in the bump effect:



The BUMP frame containing the nodes to build the bump effect

16. Add a **Glossy BSDF** shader node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**) and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**). Paste the **Mix Shader** node between the **Diffuse BSDF** and the **Material Output** nodes, and connect the **Glossy BSDF** output to the second **Shader** input socket.
17. Connect the **Normal** output of the **Bump** node to the **Normal** input socket of the **Glossy BSDF** shader. Set the **Glossy BSDF** shader's **Roughness** to `0.150` and the **Fac** value of the **Mix Shader** node to `0.300`.
18. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and connect it to the **Color** input socket of the **Diffuse BSDF** shader node. Set the color values to `0.407` for **R**, `0.323` for **G**, and `0.293` for **B**.
19. Add a **MixRGB** node, label it `Mix03`, and paste it between the **RGB** and the **Diffuse BSDF** nodes. Connect the **Color** output of the **Darken-MixRGB** node to the **Color2** input socket of the **Mix03** node.
20. Add a new **MixRGB** node. Set **Blend Type** to **Add** and the **Fac** value to `0.800`. Connect the **Color** output of the **ColorRamp01** node to the **Color1** input socket, and the output of the **Mix03** node to the **Color2** input socket of the **MixRGB** node. Connect the **Color** output of the **Add-MixRGB** node to the **Color** input socket of the **Diffuse BSDF** node.
21. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**) and paste it between the **Add-MixRGB** and the **Diffuse BSDF** nodes. Connect its output to the **Color** input socket of the **Glossy BSDF** shader node.
22. Click on the **RGB Curves** node's little window to create a new point. Set its position values to `0.24545` for **X** and `0.38125` for **Y**. Then create another point and set the position values to `0.74545` for **X** and `0.51875` for **Y**, as shown in the following screenshot:

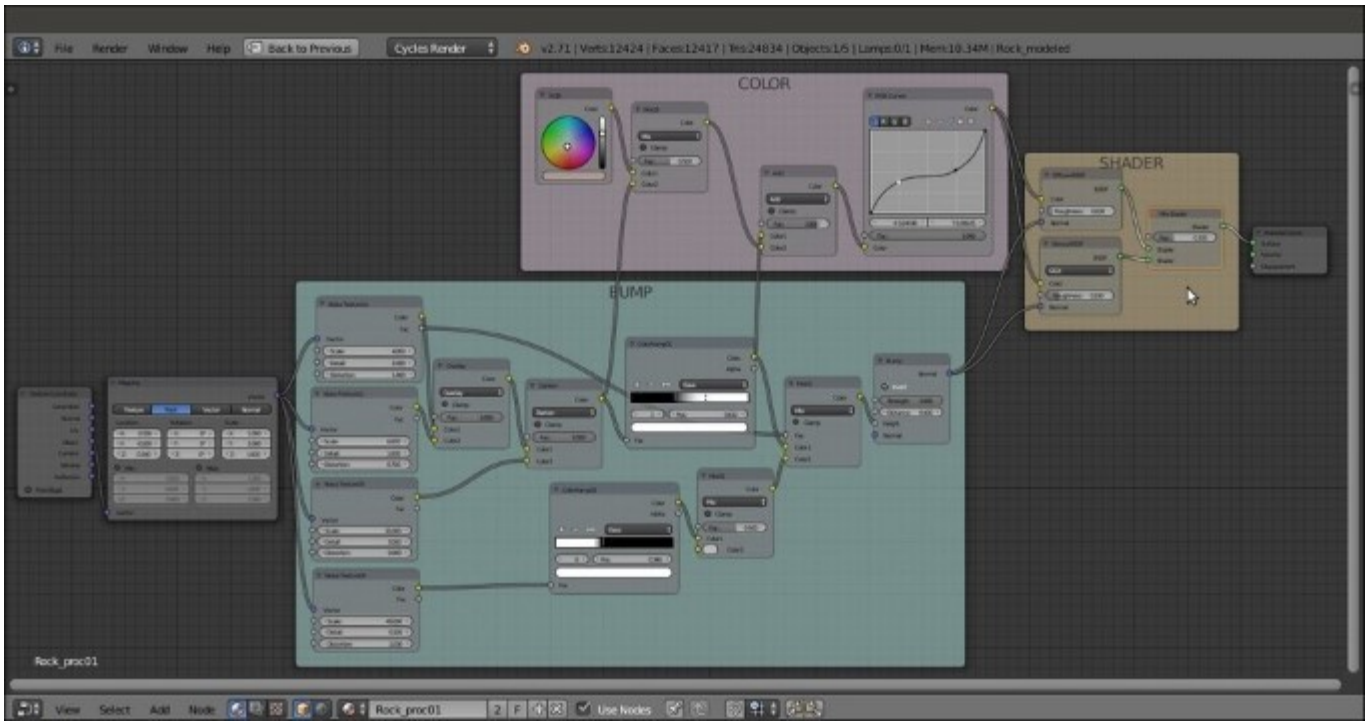


Screenshot of the COLOR frame connected to the Diffuse and Glossy shaders

How it works...

Even if this material looks a bit complex at first sight, you must note that we just mixed four procedural noise textures with different settings and iterations to build the bump effect and create the color pattern to some extent:

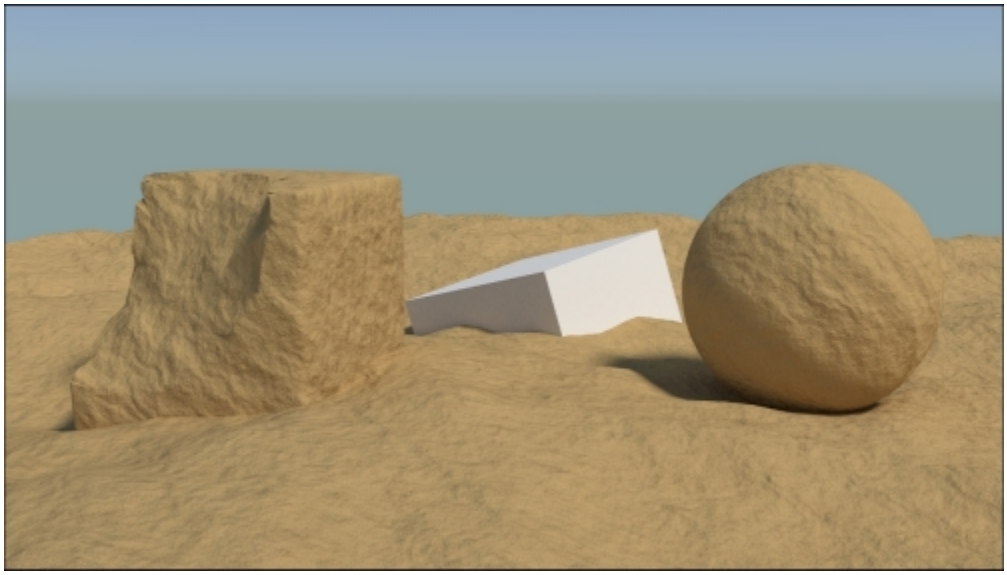
- In the first stage, from steps 2 to 15, we built the bump pattern by mixing the output of the noise textures (the **Overlay**, **Darken**, **Mix01**, and **Mix02** nodes converging to the input socket of the **Bump** node) through **MixRGB** nodes, and in some cases also edited their levels using the **ColorRamp** nodes to obtain a more random, natural look (**ColorRamp1** and **ColorRamp2**, all the nodes inside the **BUMP** frame).
- In the second stage, from steps 18 to 22, we built the color pattern by mixing a simple color output with the output of some of the **Noise Texture** nodes, and then edited the result using an **RGB Curves** node (nodes inside the **COLOR** frame).
- The results of both the bump and the color patterns were then piped into the appropriate sockets of the nodes inside the **SHADER** frame, that is, the **Diffuse BSDF** node was mixed with the **Glossy BSDF** node to add specularity (steps 16 and 17). The overall material network in the **Node Editor** window is shown in the following screenshot:



The overall vision of the material network

Creating a sand material using procedural textures

In this recipe, we will create a sand material that looks like what is shown in the following screenshot, which is good for both close and distant objects:

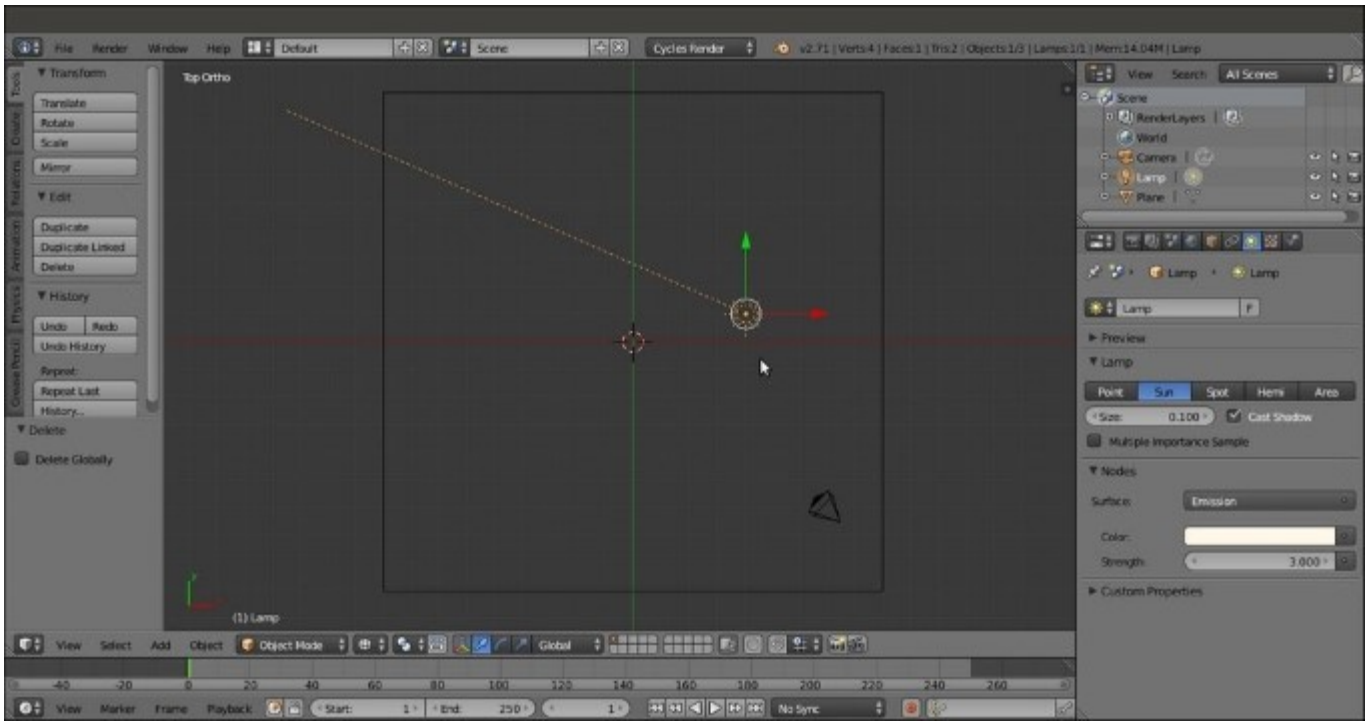


The sand material as it appears in the final rendering

Getting ready

Start Blender and switch to the **Cycles Render** engine. Then perform the following steps:

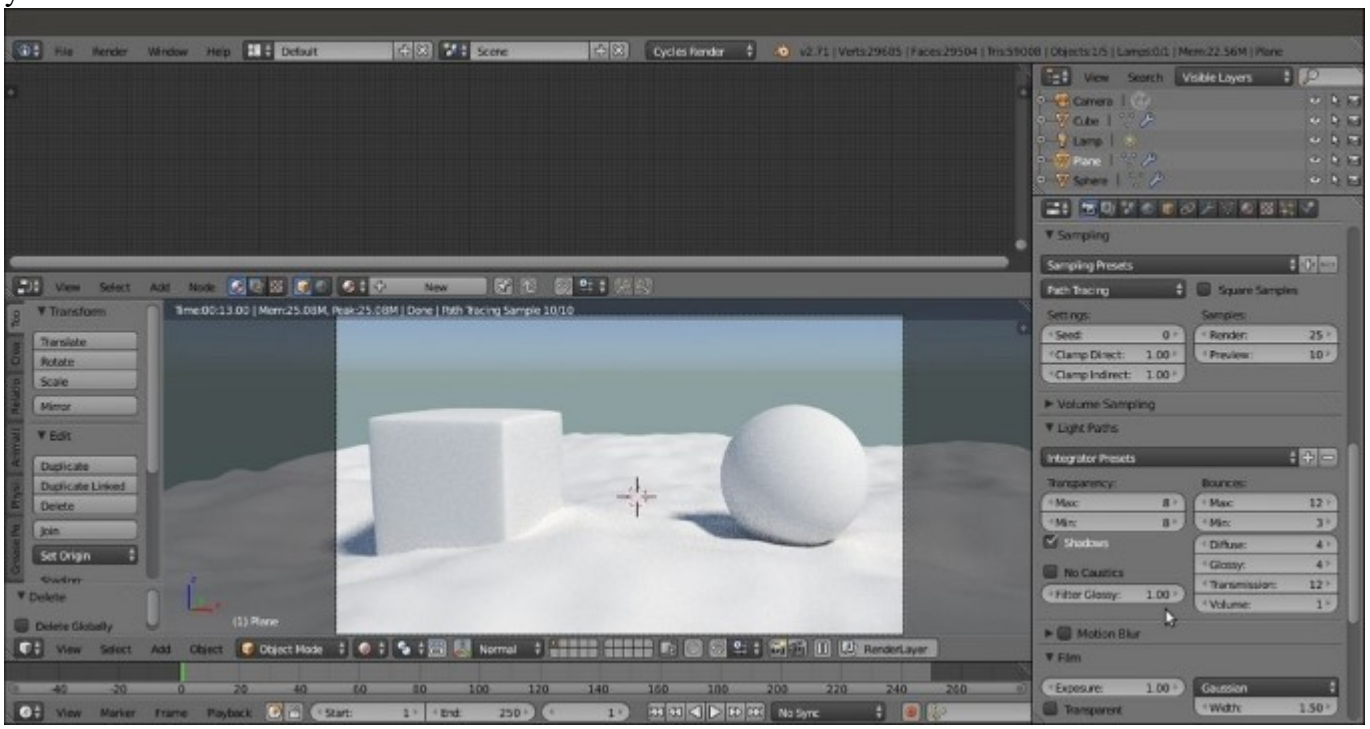
1. Delete the default Cube and add a Plane (press *Shift* + *A* and navigate to **Mesh** | **Plane**).
2. Press *Tab* to go to **Edit Mode** and scale it nine times bigger, with 18 units per side (press *S*, enter the digit *9*, and press *Enter*). Go out of **Edit Mode**.
3. Go to the **World** window and click on the **Use Nodes** button under the **Surface** subpanel in the **Properties** panel to the right of the screen. Then click on the little square with a dot on the right side of the **Color** slot. Select the **Sky Texture** item from the pop-up menu. Set the **Strength** value to `1.900`.
4. Select the **Lamp**, go to the **Object data** window, and click on the **Use Nodes** button. Then change the **Type of Lamp** to **Sun** and set **Strength** to `3.000`. Change the color values to `1.000` for **R**, `0.935` for **G**, and `0.810` for **B**. In orthogonal top view, rotate the **Sun** lamp by `45°`, as shown in the following screenshot:



The scene from the top with the Sun Lamp selected

5. Add a Cube and a UV Sphere to the scene and place them leaning on the Plane.
6. Select the Cube and click on the **Smooth** button in the **Shading** subpanel under the **Tools** tab to the left of the 3D window (press the *T* key to make it appear if it is not present).
7. Select the UV Sphere and perform the same actions as given in step 6.
8. Select the Cube, and in the **Object modifiers** window, add a **Bevel** modifier. Set **Width** to `0.1000`, **Segments** to `2`, and **Profile** to `0.15`. Assign a **Subdivision Surface** modifier, set both the **Subdivisions** levels to `4`, and check the **Optimal Display** item. Assign a **Smooth** modifier and set **Factor** to `1.000` and **Repeat** to `15`.
9. Select the UV Sphere and assign a **Subdivision Surface** modifier with **Subdivisions** levels set to `2` and the usual **Optimal Display** item checked.
10. Select the Plane, click on the **Smooth** button, and then go to **Edit Mode** and press *W*. In the **Specials** pop-up menu, select the **Subdivide** item. Press the *F6* key to call the **Options** pop-up menu (or go to the panel at the bottom of the **Tool Shelf** tabs) and set **Number of Cuts** to `10`.
11. Go out of **Edit Mode**, go to the **Object modifiers** window, and assign a **Subdivision Surface** modifier. Switch to the **Simple** subdivision algorithm and set both **Subdivisions** to `3`. Check the **Optimal Display** item.
12. Assign a **Displace** modifier and then click on the **Show texture in texture tab** button to the side of the **New** button. In the **Texture** window, click on the **New** button and switch the default **Clouds** texture with the **Voronoi** texture. Set the **Size** value to `5.00`.
13. Assign a new **Displace** modifier, click on the **Show texture in texture tab** button, and load a **Voronoi** texture again. Leave the **Size** value at `0.25`.
14. Assign a **Smooth** modifier and set **Factor** to `1.000` and the **Repeat** value to `5`.

15. Place the Camera to have a nice angle on the Plane and switch from the 3D view to a Camera view (by pressing the *0* key on the numeric keypad).
16. Split the 3D window into two horizontal rows and change the upper row to a **Node Editor** window. Put the mouse cursor in the 3D view and press *Shift* + *Z* to set the **Camera** view mode to **Rendered**.
17. Go to the **Render** window. Under the **Sampling** subpanel, set both the **Clamp Direct** and **Clamp Indirect** values to `1.000`. Go to the **Light Path** subpanel and set the **Filter Glossy** value to `1.000`.
18. Set the **Render** samples to `25`. The **Rendered** preview is shown in the following screenshot for your reference:



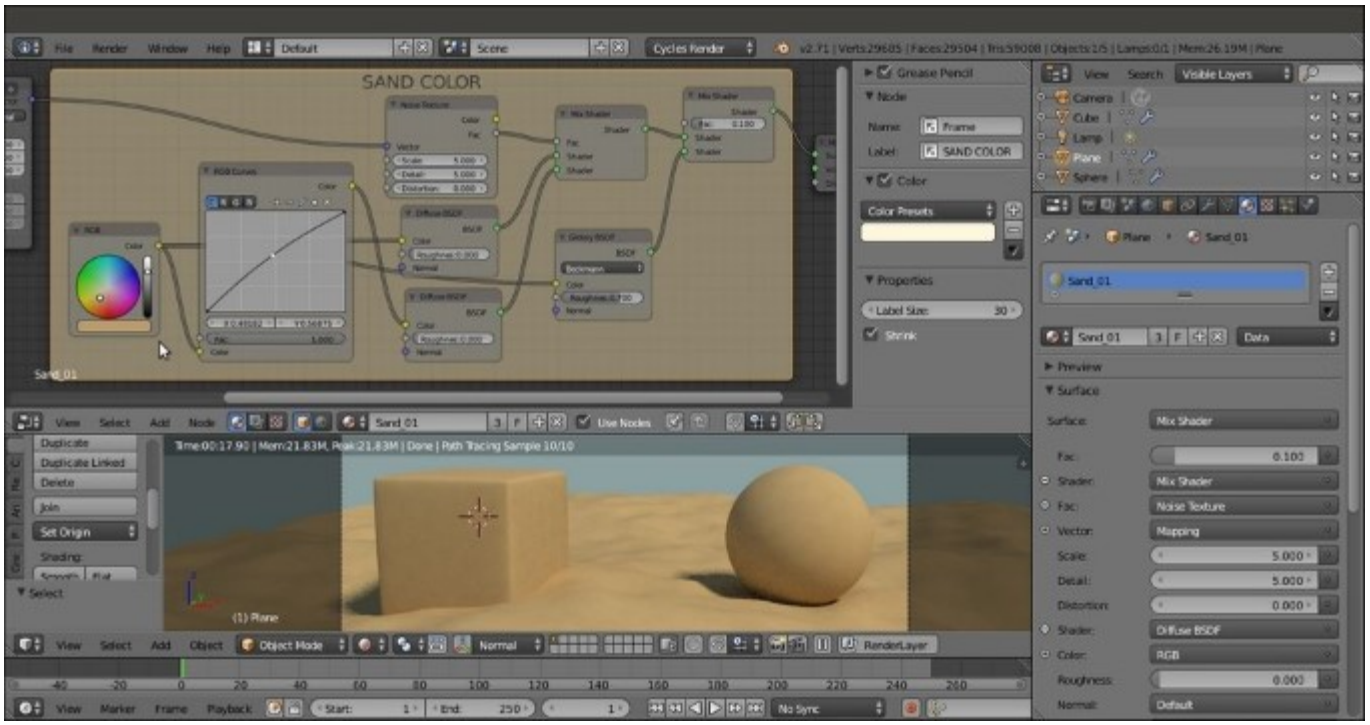
The prepared scene as it appears in the Rendered preview, with the rendering settings to the right

How to do it...

We have prepared the scene. Now let's start with the creation of the sand material using the following steps:

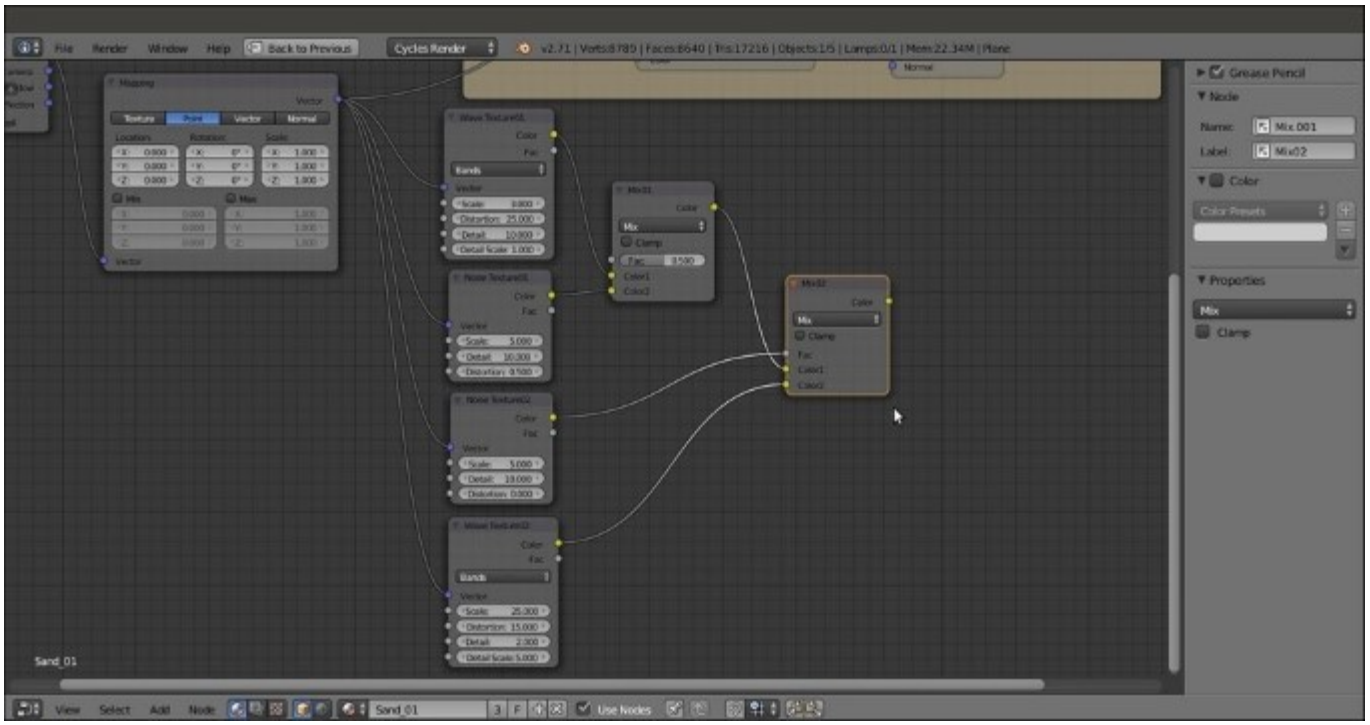
1. Select the Plane and click on the **New** button in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Sand_01`.
2. Keeping the *Shift* key pressed, select the UV Sphere, the Cube, and for last one, the Plane (that is the active object of the multi-selection) by right-clicking on them. Press *Ctrl* + *L*, and in the **Make Links** pop-up menu, select the **Material** item to assign the same material to the other two objects. The `Sand_01` material is now assigned to all three objects.

3. In the **Material** window under the **Properties** panel to the right, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In both the **Shader** slots, assign a **Diffuse BSDF** shader.
4. In the **Node Editor**, add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**). Connect the output of the **RGB** node to the **Color** input socket of the first **Diffuse BSDF** shader and to the **Color** input socket of the **RGB Curves** node. Then connect the output of the **RGB Curves** node to the **Color** input socket of the second **Diffuse BSDF** shader node.
5. Change the color values of the **RGB** node to `0.500` for **R**, `0.331` for **G**, and `0.143` for **B**. Click on the **RGB Curves** node window to create a new point, and set the coordinates to `0.48182` for **X** and `0.56875` for **Y**.
6. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**).
7. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and then connect the **Vector** output to the **Vector** input of the **Noise Texture** node.
8. Connect the **Fac** output of the **Noise Texture** node to the **Fac** input of the **Mix Shader** node. Increase the **Detail** value of the texture to `5.000`.
9. Press *Shift* + *D* to duplicate the **Mix Shader** node and paste it between the first **Mix Shader** node and the **Material Output** node. Add a **Glossy BSDF** shader node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**), set its **Roughness** to `0.700`, and connect its output to the second **Shader** input socket of the second **Mix Shader** node. Set the **Fac** value to `0.100`.
10. Connect the **Color** output of the **RGB** node to the **Color** input socket of the **Glossy BSDF** shader node.
11. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**), press *Shift* and multi-select the **RGB** node, the **RGB Curves** node, the **Noise Texture** node, the two **Diffuse BSDF** nodes, the **Glossy BSDF** shader, the two **Mix Shader** nodes, and the **Frame**, and press *Ctrl* + *P* to parent them. In the **Properties** panel (press *N* key in the **Node Editor** window), label the **Frame** as `SAND COLOR`, as shown in the following screenshot:



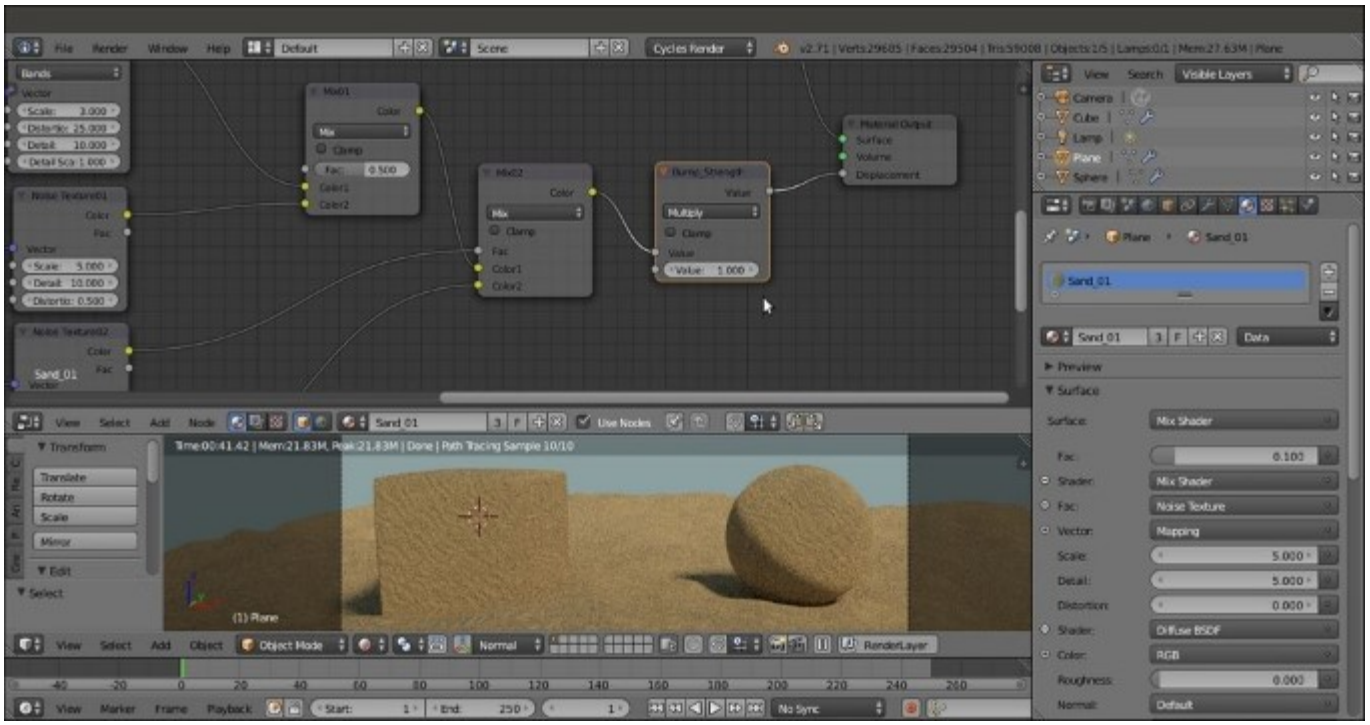
The total vision of the SAND COLOR frame and the rendered effect on the objects

12. Add a new **Noise Texture** node and a **Wave Texture** node (press *Shift* + *A* and navigate to **Texture** | **...**). Select both and press *Shift* + *D* to duplicate them. Label them `Wave Texture01`, `Wave Texture02`, `Noise Texture01`, and `Noise Texture02`.
13. Arrange the four texture nodes in a column like this: **Wave Texture01**, **Noise Texture01**, **Noise Texture02**, and **Wave Texture02**. Connect the **Mapping** output to the texture nodes' **Vector** input.
14. Set the **Scale** value of **Wave Texture01** to `3.000`, **Distortion** to `25.000`, and **Detail** to `10.000`. Set the **Detail** value of **Noise Texture01** to `10.000` and **Distortion** to `0.500`. Set the **Detail** value of **Noise Texture02** to `10.000` as well. Finally, set the **Wave Texture02** node's **Scale** value to `25.000`, **Distortion** to `15.000`, and **Detail Scale** value to `5.000`.
15. Add a **MixRGB** node (press *Shift* + *A* and **Color** | **MixRGB**) and label it `Mix01`. Connect the **Wave Texture01** node's **Color** output to the **Color1** input and the **Noise Texture01** node's **Color** output to the **Color2** input.
16. Select the **MixRGB** node and press *Shift* + *D* to duplicate it. Label it `Mix02` and connect the **Color** output of the **Mix01** node to the **Color1** input of the **Mix02** node. Then connect the **Wave Texture02** node's **Color** output to its **Color2** input.
17. Connect the **Color** output of the **Noise Texture02** node to the **Fac** input socket of the **Mix02** node. Now the **Node Editor** window looks like what is shown in the following screenshot:



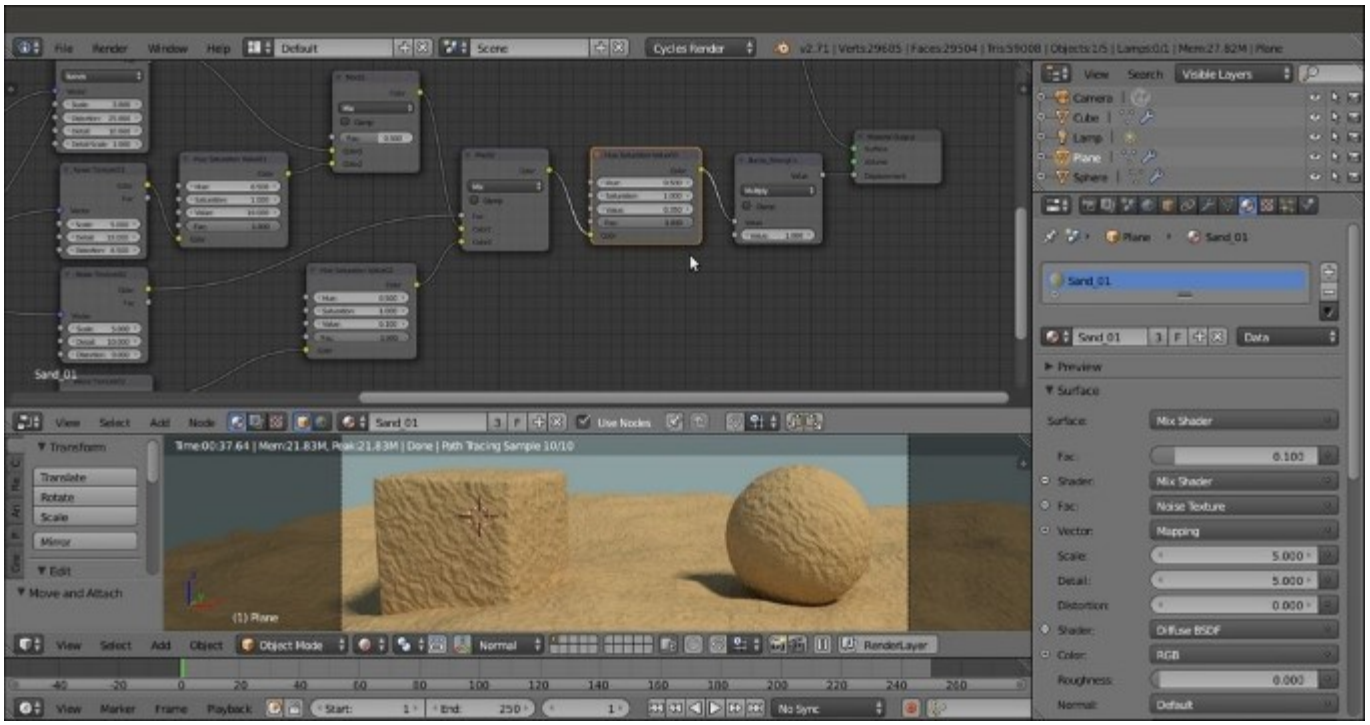
Building the bump pattern

18. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and label it `Bump_Strength`. Change **Operation** to **Multiply** and connect the output of the **Mix02** node to the first **Value** input socket. Set the second **Value** input socket's value to `1.000` and connect the node output to the **Displacement** input socket of the **Material Output** node, as shown in this screenshot:



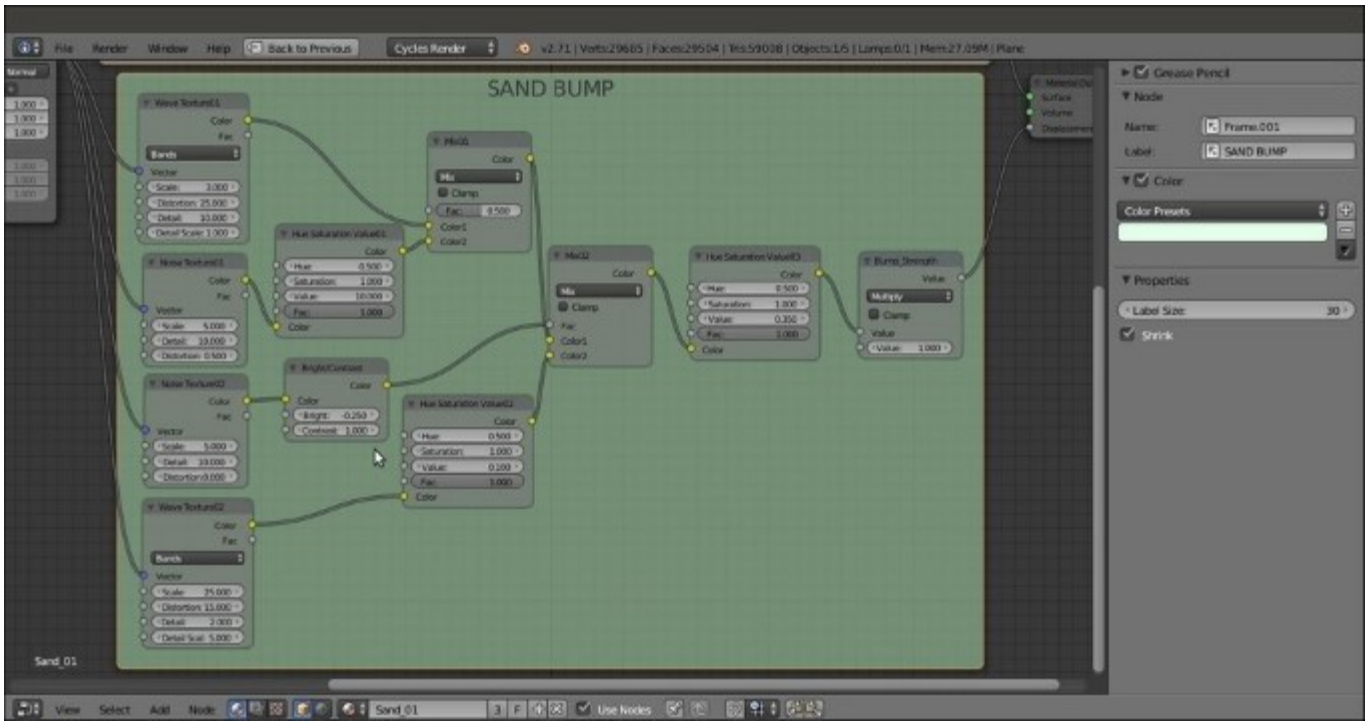
The effect of the total output of the bump nodes connected to the Displacement input of the Material Output node

19. Add a **Hue/Saturation** node (press *Shift* + *A* and navigate to **Color** | **Hue/Saturation**), label it `Hue Saturation Value01`, and drag it onto the link connecting the **Noise Texture01** node to the **Mix01** node to paste it in between. Set **Value** to `10.000`.
20. Press *Shift* + *D* to duplicate the **Hue/Saturation** node, label the duplicate `Hue Saturation Value02`, and drag it onto the link connecting the **Wave Texture02** node to the **Mix02** node. Set **Value** to `0.100`.
21. Press *Shift* + *D* to duplicate it again, name the duplicate `Hue Saturation Value03`, and drag it between the **Mix02** node and the **Bump Strength** node. Set **Value** to `0.350`. The following screenshot shows the effect of adding variation to the bump:



Adding variation to the bump pattern

22. Add a **Bright/Contrast** node (press *Shift* + *A* and navigate to **Color** | **Bright Contrast**). Drag it so that it's pasted between the **Noise Texture02** node and the **Mix02** factor input. Set the **Bright** value to `-0.250` and the **Contrast** value to `1.000`.
23. Add a **Frame**, press *Shift* (or box-select (press *B* key) these 11 nodes and then the **Frame**), and press *Ctrl* + *P* to parent them. Label the **Frame** `SAND BUMP`, as shown in the following screenshot:

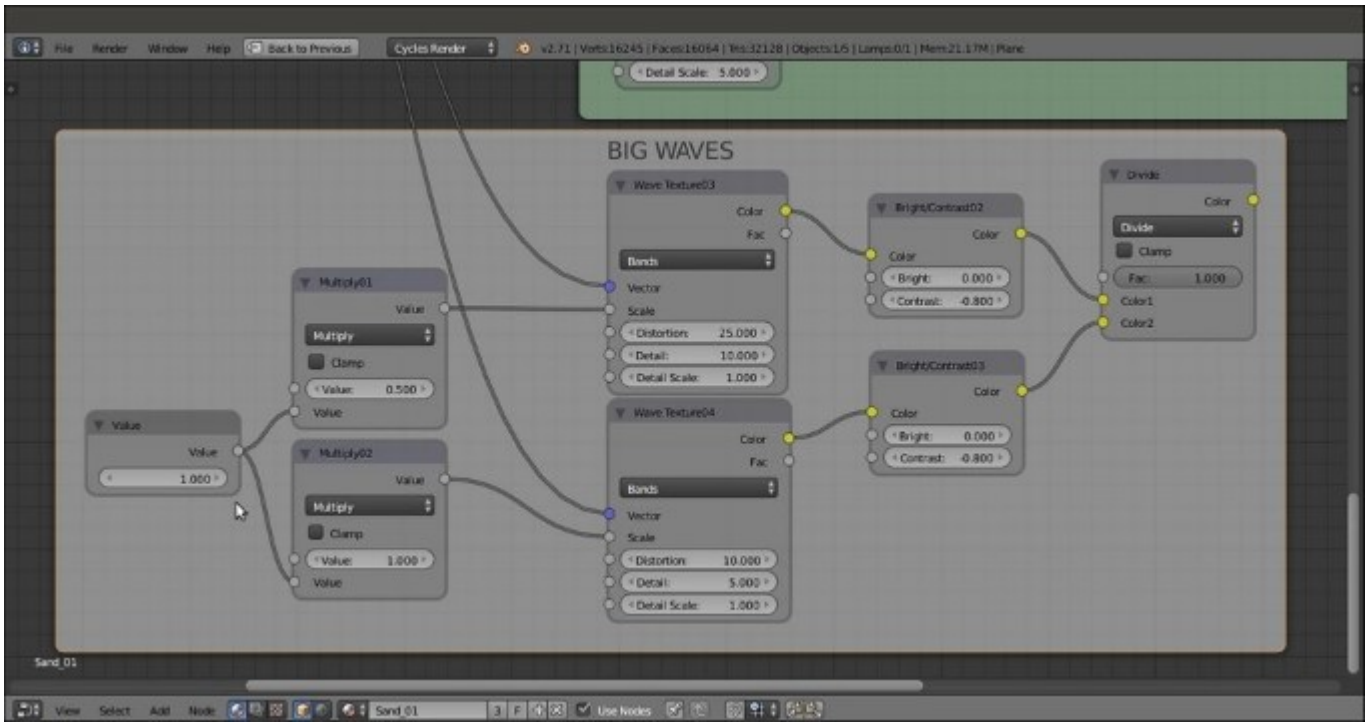


The SAND BUMP frame containing all the bump nodes

24. Select the **Wave Texture02** node and press *Shift* + *D* to duplicate it twice (if the node is still parented to the **SAND BUMP** frame after duplication, press *Alt* + *P* to unparent it). Label them `Wave Texture03` and `Wave Texture04`. Then duplicate the **Bright/Contrast** node twice and name the duplicates `Bright/Contrast02` and `Bright/Contrast03`.
25. Select one of the **MixRGB** nodes and press *Shift* + *D* to duplicate it. Set **Blend Type** (and the label) to **Divide** and the **Factor** to `1.000`.
26. Connect the **Mapping** vector output to the **Vector** input sockets of the two new wave textures.
27. Connect each **Color** output of the two new wave textures to the respective **Color** input of the **Bright/Contrast02** nodes. Then connect their color output to the **Color1** and **Color2** input of the **Divide** node.
28. In the **Wave Texture03** node, set **Scale** to `0.500`, **Distortion** to `25.000`, **Detail** to `10.000`, and **Detail Scale** to `1.000`. In the **Bright/Contrast02** node, set the **Bright** value to `0.000` and the **Contrast** value to `-0.800`.
29. In the **Wave Texture04** node, set **Scale** to `1.000`, **Distortion** to `10.000`, **Detail** to `5.000`, and **Detail Scale** to `1.000`. In the respective **Bright/Contrast03** node, set the **Bright** value to `0.000` and the **Contrast** value to `-0.800`.
30. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set **Operation** to **Multiply**, label it `Multiply01`, and leave the first **Value** as it is—the same as the **Scale** value of the **Wave Texture03** node (`0.500`). Set the second **Value** to `1.000`. Connect the **Value** output to the **Scale** input socket of the **Wave Texture03** node.
31. Press *Shift* + *D* to duplicate the **Math** node (label it `Multiply02`). Move it to the side of the **Wave Texture04** node and set the first **Value** to be the same as the **Scale** value of the texture

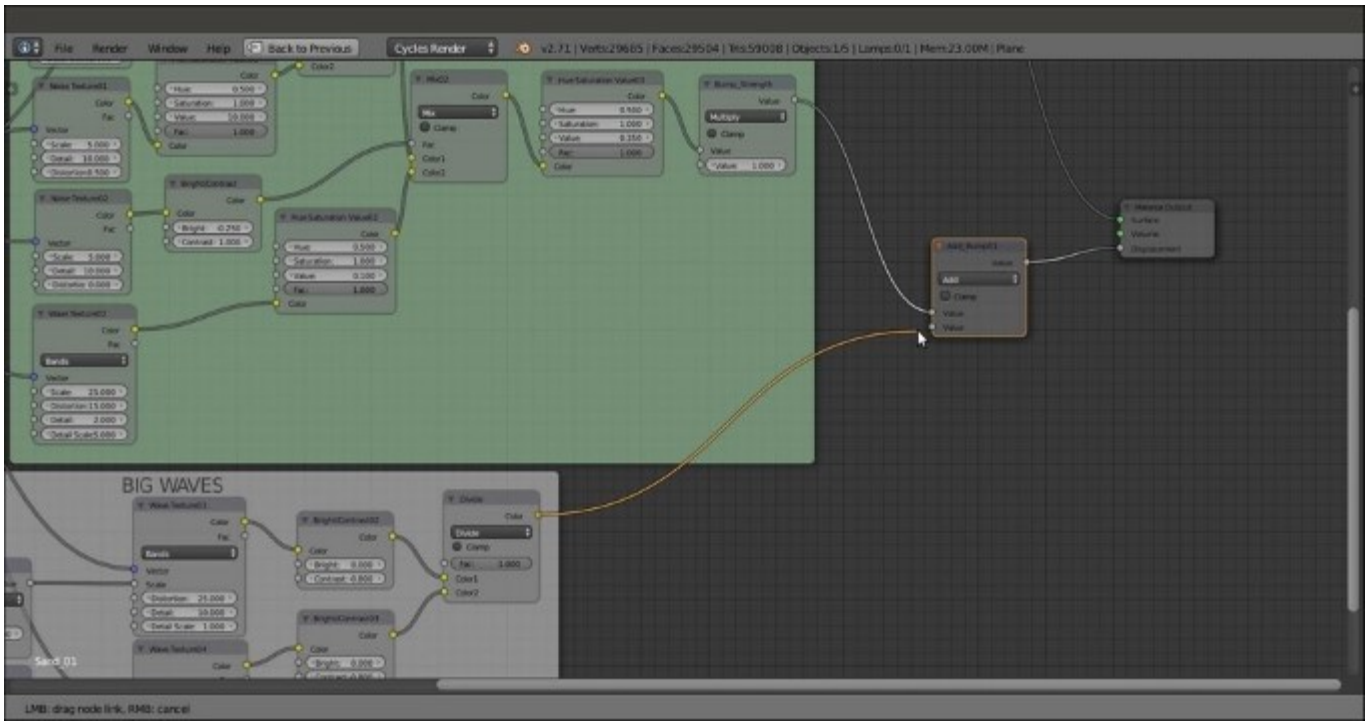
node (`1.000`). Set the second **Value** to `1.000` as well, and connect the **Value** output to the **Scale** input of the **Wave Texture04** node.

32. Add a **Value** node (press *Shift* + *A* and navigate to **Input** | **Value**). Connect the output to the second **Value** input sockets of both the **Multiply-Math** nodes. Label it `Waves_size` and set the input value to `1.000`.
33. Add a **Frame**, parent the last added nodes, and label it `BIG WAVES`, as shown in the following screenshot:



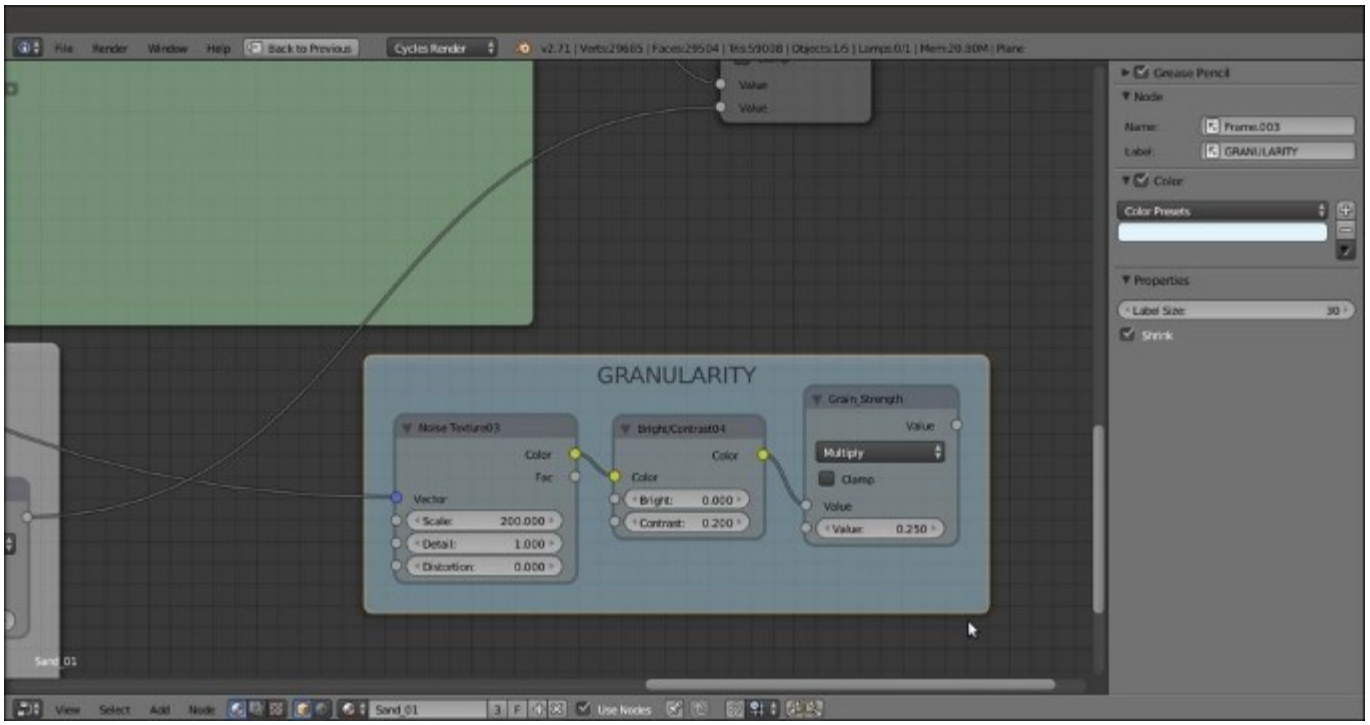
A new frame containing a new bump effect to be added to the previous one

34. Duplicate the **Bump_Strength** node, unparent it from the **SAND BUMP** node, and set the **Operation** to **Add**. Drag it onto the link between the **Bump_Strength** node and the **Material Output** node, and label it `Add_Bump01`.
35. Connect the **Divide** node output of **BIG WAVES** to the second **Value** input socket of the **Add_Bump01** node, as shown in this screenshot:



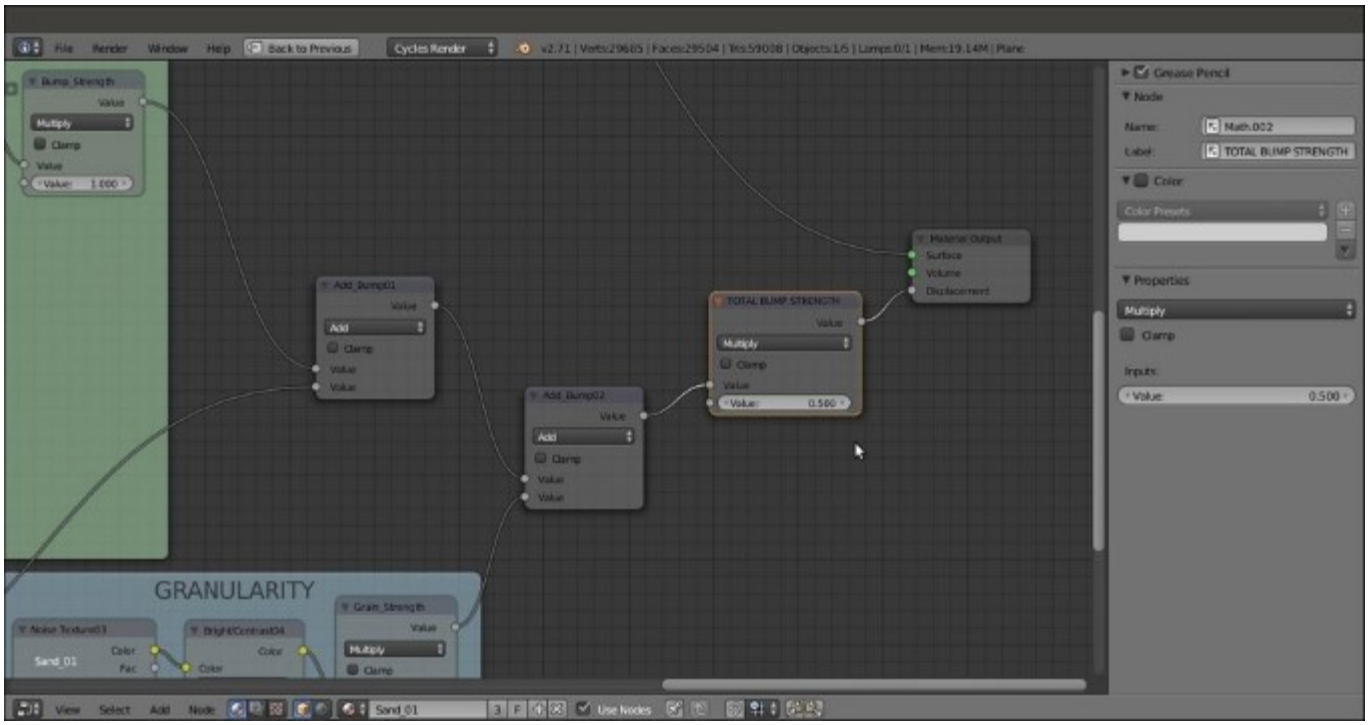
The output of the two bump frames added together

36. Duplicate a **Noise Texture** node (label it `Noise Texture03`), a **Bright/Contrast** node (label it `Bright/Contrast04`), and a **Math** node (label it `Grain_Strength`). Connect the **Mapping** output to the **Vector** input of the texture node. Then connect the **Noise Texture** node's **Color** output to the **Bright/Contrast04** node's **Color** input and its output to the first **Value** input socket of the **Grain_Strength** node. Set its operation to **Multiply** and the second **Value** to `0.250`.
37. Set the texture **Scale** to `200.000`, **Detail** to `1.000`, and **Distortion** to `0.000`. Set the **Bright/Contrast04** node's **Bright** value to `0.000` and **Contrast** to `0.200`.
38. Add a **Frame**, parent the three nodes to it, and label it **GRANULARITY**, as shown in the following screenshot:



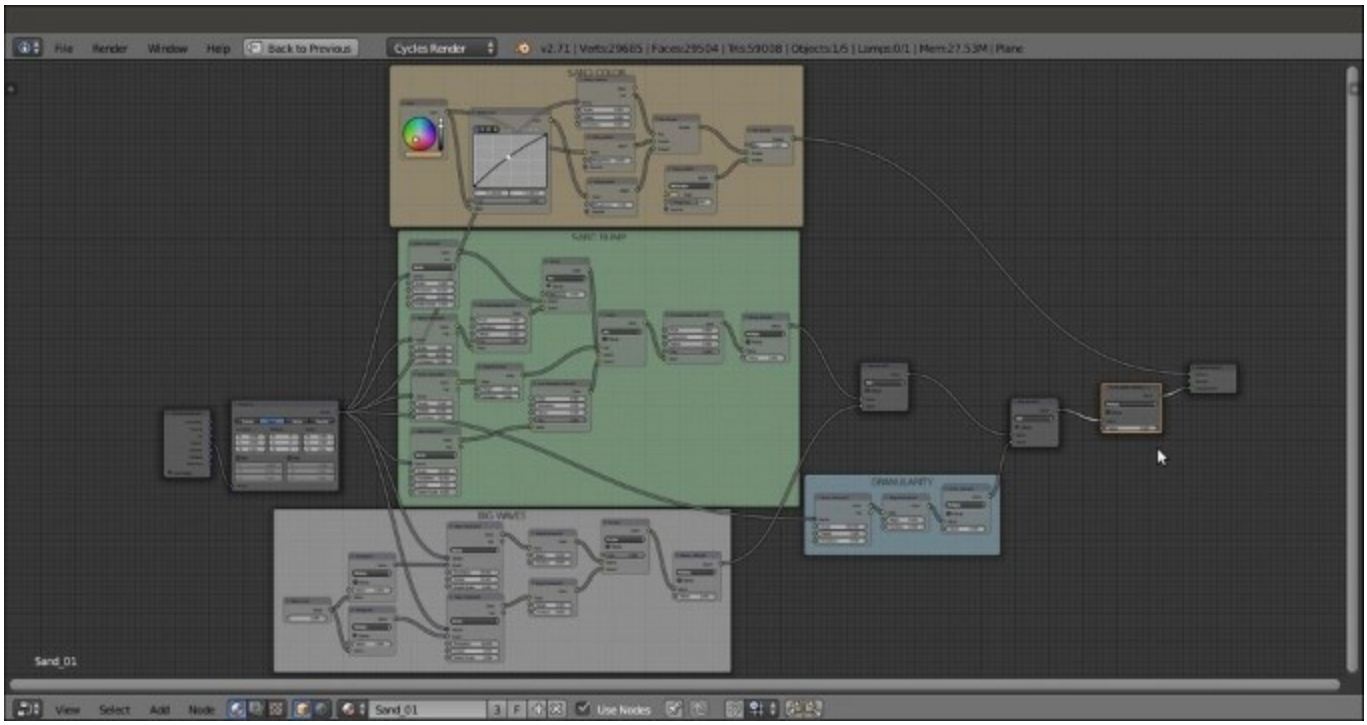
One more bump effect frame

39. Duplicate the **Add_Bump01** node, and label it `Add_Bump02`, and paste it between the **Add_Bump01** node and the **Material Output** node. Connect the **Grain_Strength** output of the **GRANULARITY** frame to the second **Value** input socket of the **Add_Bump02** node.
40. Duplicate the **Add_Bump02** node and paste it just before the **Material Output** node. label it **TOTAL BUMP STRENGTH**, set the node **Operation** to **Multiply**, and set the second **Value** to `0.500`, as shown in this screenshot:



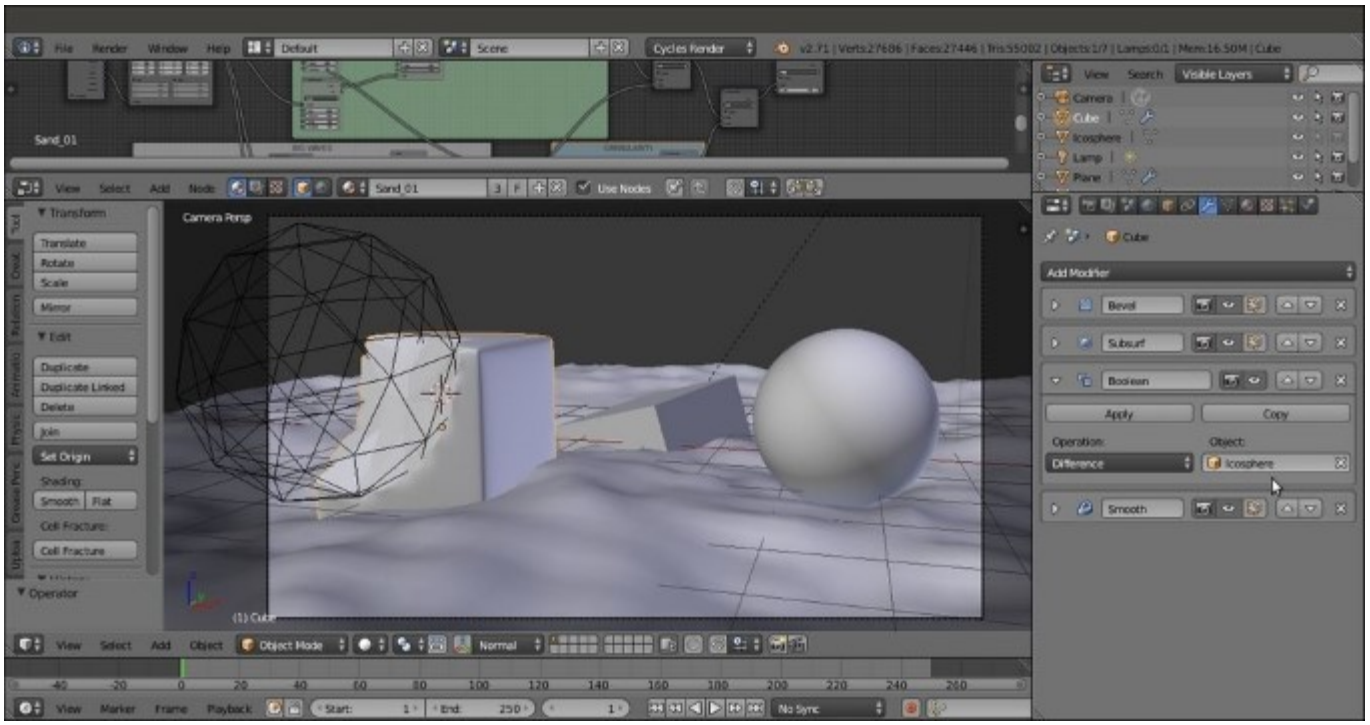
The GRANULARITY frame output added to the previous bump ones

So here we are now—the sand shader is complete, and this is how the nodes' network looks in the **Node Editor** window:



To obtain the image shown at the beginning of this recipe, we also added a few elements to the scene:

- A new Cube primitive, with a simple diffuse pure white material, added just for reference to light intensity.
- An Ico Sphere primitive, set as invisible and disabled for the rendering in **Outliner**. It works as target **Object** for a Boolean modifier assigned to the sand Cube and is placed in the stack between the **Subdivision Surface** and **Smooth** modifiers, as shown in the following screenshot:



How it works...

The concept behind the structure of this material is basically the same as that for the procedural rock, and it can be subdivided into stages as well:

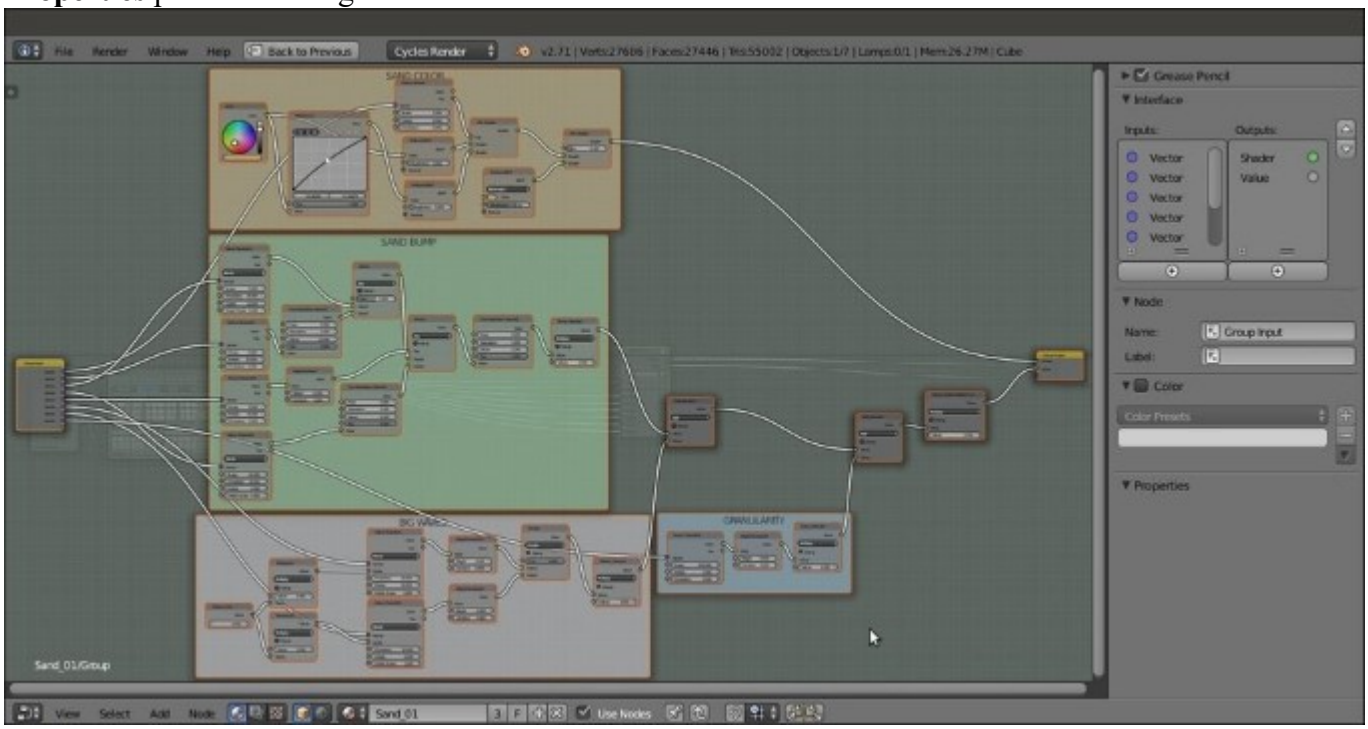
- From step 1 to step 10, we built the color part of the shader, blending two differently colored **Diffuse BSDF** nodes on the ground of a **Noise Texture** factor, and building a basic shader with the **Glossy BSDF** component.
- From step 12 to step 22, we built the main bump effect, this time piped directly as whole in the **Displacement** input of the **Material Output** node rather than to the **Normal** input sockets per shader.
- From step 24 to step 32, we built a supplementary bump effect, this time to simulate the big waves you usually see on a desert's sand dunes. This effect was left apart from the main bump to be easily reduced or eliminated if required. Then we added two **Math** nodes set to **Multiply** and driven by a **Value** node to automatically set the size of the big sand waves. Actually, this is more a repeating effect, and the bigger the value, the smaller and closer the waves.
- In steps 36 and 37, we built a last bump effect to add the sand grain if necessary, for example, for objects very close to the camera. In steps 39 and 40, we summed all the bump effects, to be driven by the last **Math** node value.

Every stage has been framed and properly labeled to make it more easily readable in the **Node Editor** window.

There's more...

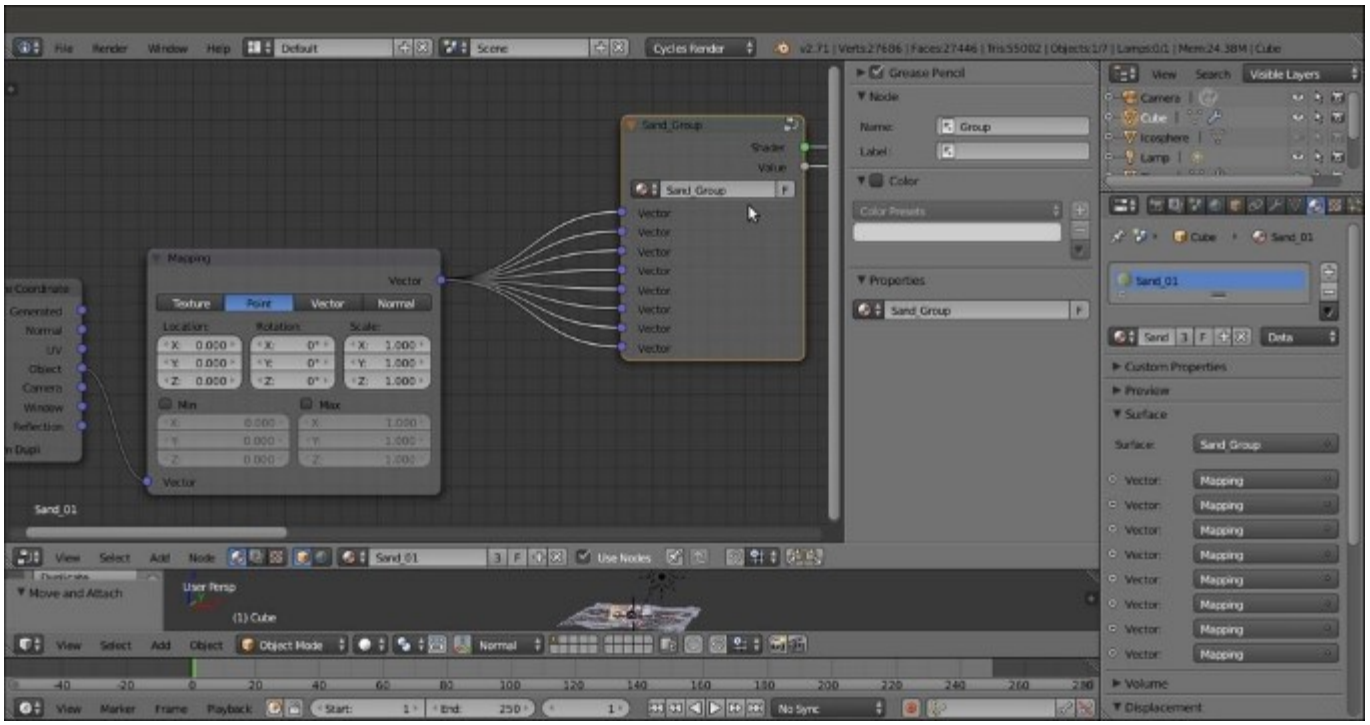
One more thing we can do to improve this material is combine everything into a handy group node, at the same time leaving the fundamental values to be tweaked exposed on the node group interface. To do this follow these steps:

1. Put the mouse cursor in the **Node Editor** window and press the *B* key. Two horizontal and a vertical lines appear at the location of the mouse cursor. Click and drag the mouse to encompass the framed nodes, leaving outside only the **Texture Coordinate** node, the **Mapping** node, and the **Material Output** nodes. After the mouse button is released, everything you encompassed is selected.
2. Press *Ctrl* + *G* and create the group. Then press *N* in the **Node Editor** window to call the **Properties** panel on the right side.



The previous sand material network inside a node group

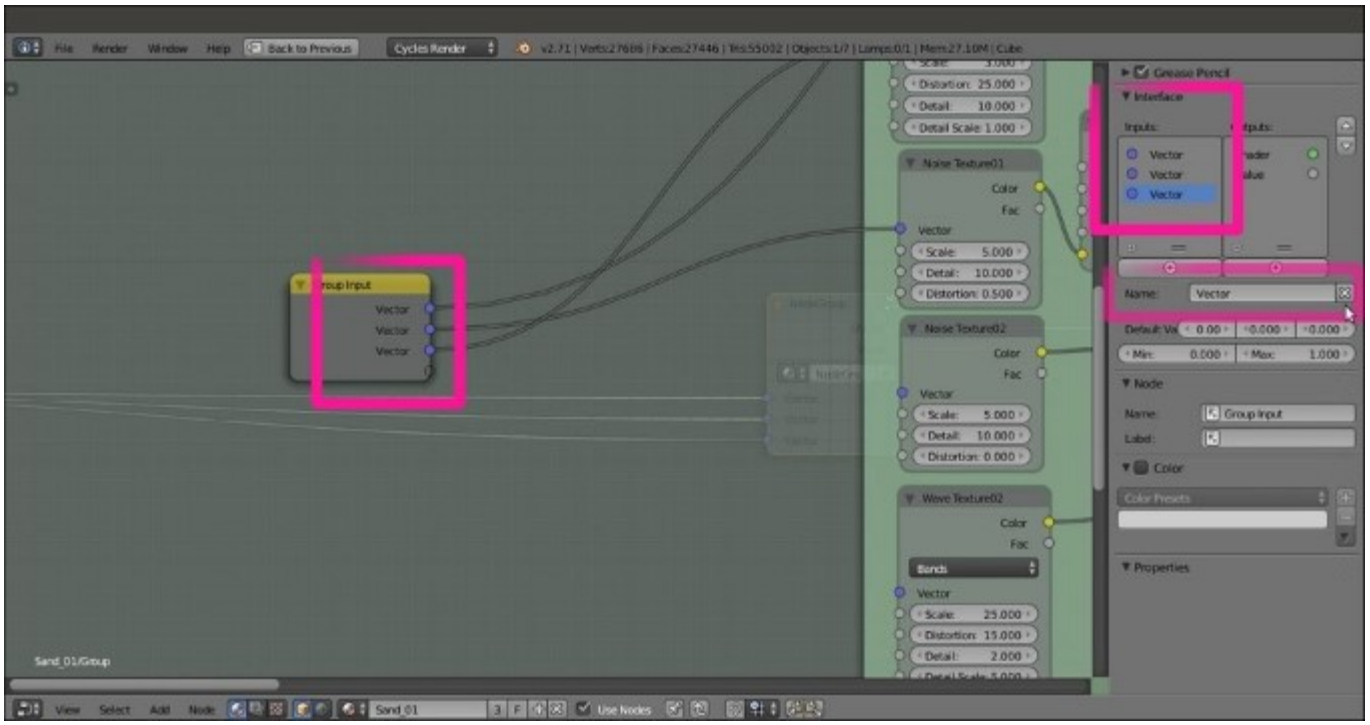
3. Press *Tab* to go out of **Edit Mode**. Then click on the little window on the interface to change the name from `NodeGroup` to `Sand_Group`.



The closed node group

As you can see, inside the group, the **Group Input** node collects all the **Vector** sockets from which the various texture nodes take their mapping coordinates, so we now have eight **Vector** sockets in the outer interface, all connected to the same **Object** output of the **Mapping** node. However, we need only one **Vector** input to map all the textures inside the group, so let's perform the following steps:

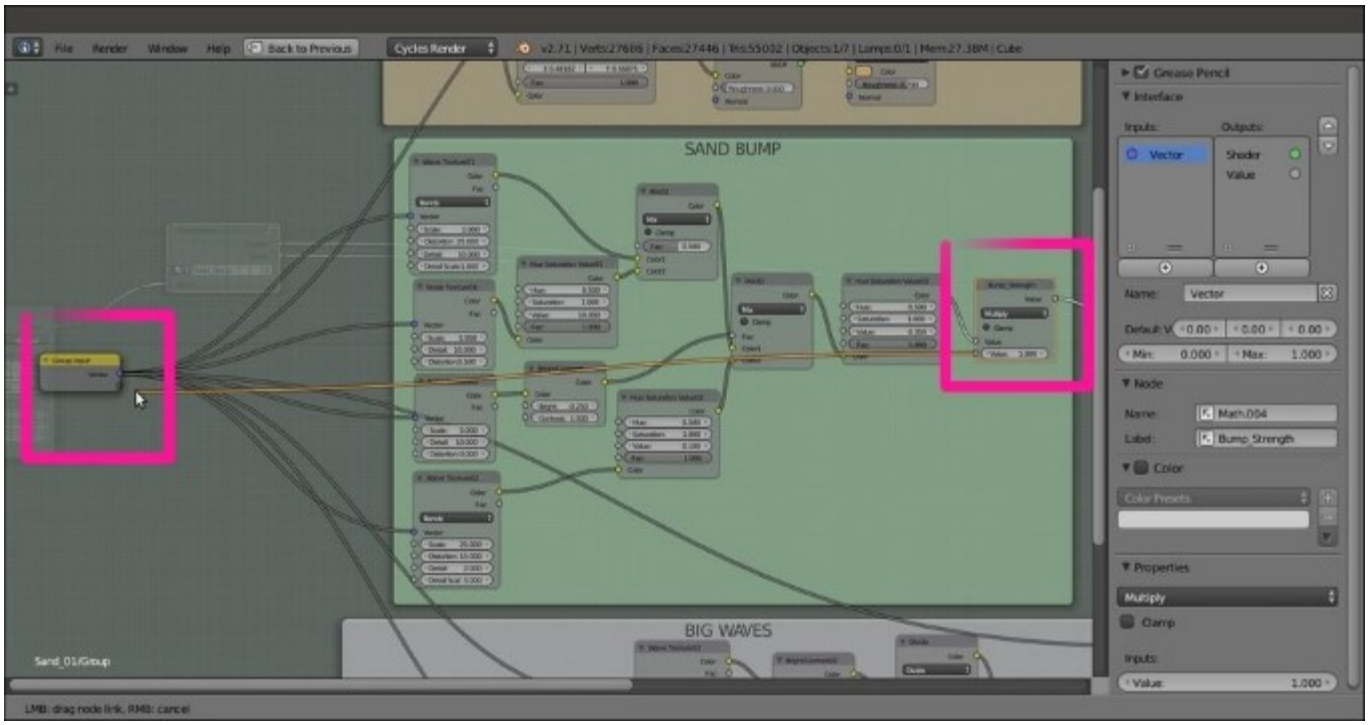
1. Press *Tab* to go to **Edit Mode** again and deselect everything by pressing the *A* key.
2. Select the first bottom **Vector** output by clicking on the list of names in the little **Inputs** window under the **Properties/Interface** panel. Delete the corresponding **Vector** socket from the **Group Input** node by clicking on the **X** icon to the side of the newly appeared **Name** slot. Then click on the **X** icon again, and go on like this to delete all the **Vector** sockets except the last socket at the top of the list as shown in the following screenshot:



3. Now press *Shift*, select the **Group Input** node and the first texture node, and press *F* to automatically connect them.
4. Repeat to connect all the eight texture nodes to the **Vector** socket of the **Group Input** node.

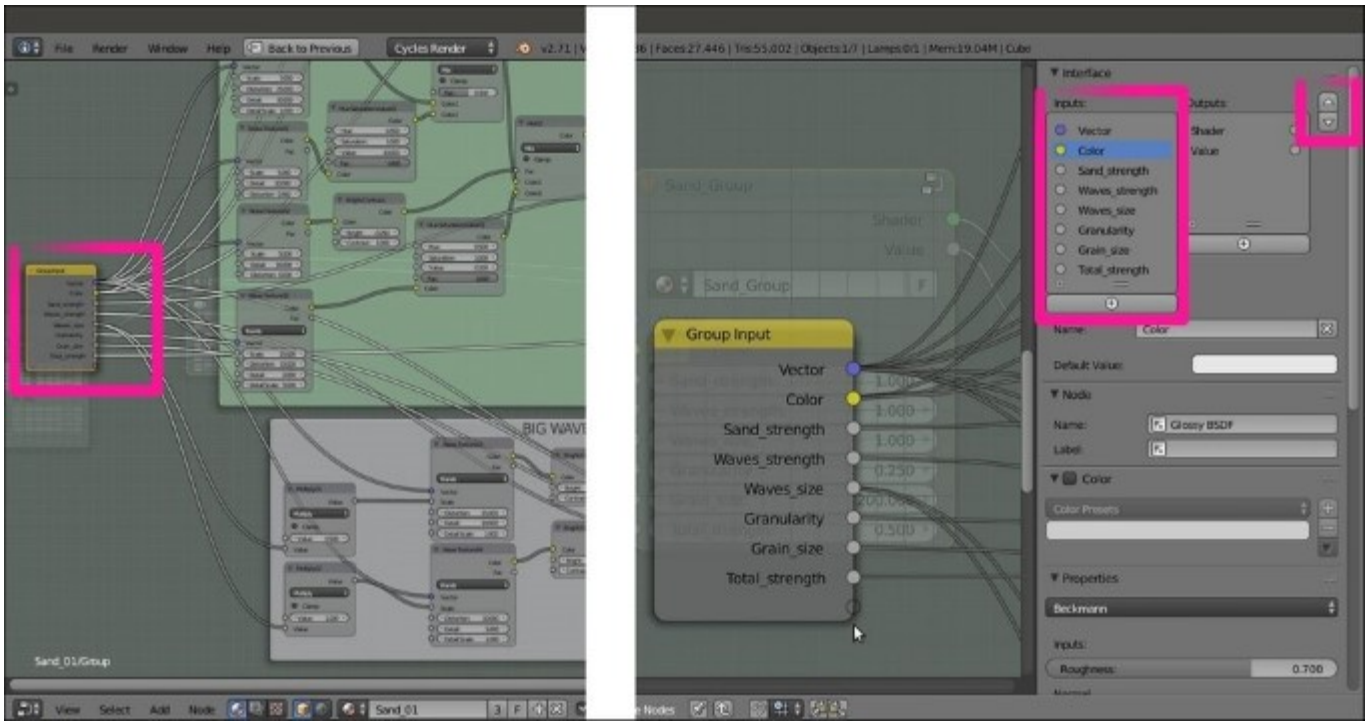
Now we need to expose some of the values to modify the material from the interface, so let's perform the following steps:

1. From the second **Value** socket of the **Bump_Strength** node inside the **SAND BUMP** frame, click and drag a link to the bottom free socket in the **Group Input** node. In the **Input** window under the **Properties** panel, double-click on the newly appeared input socket name, **Value**, and write `Sand_strength`.



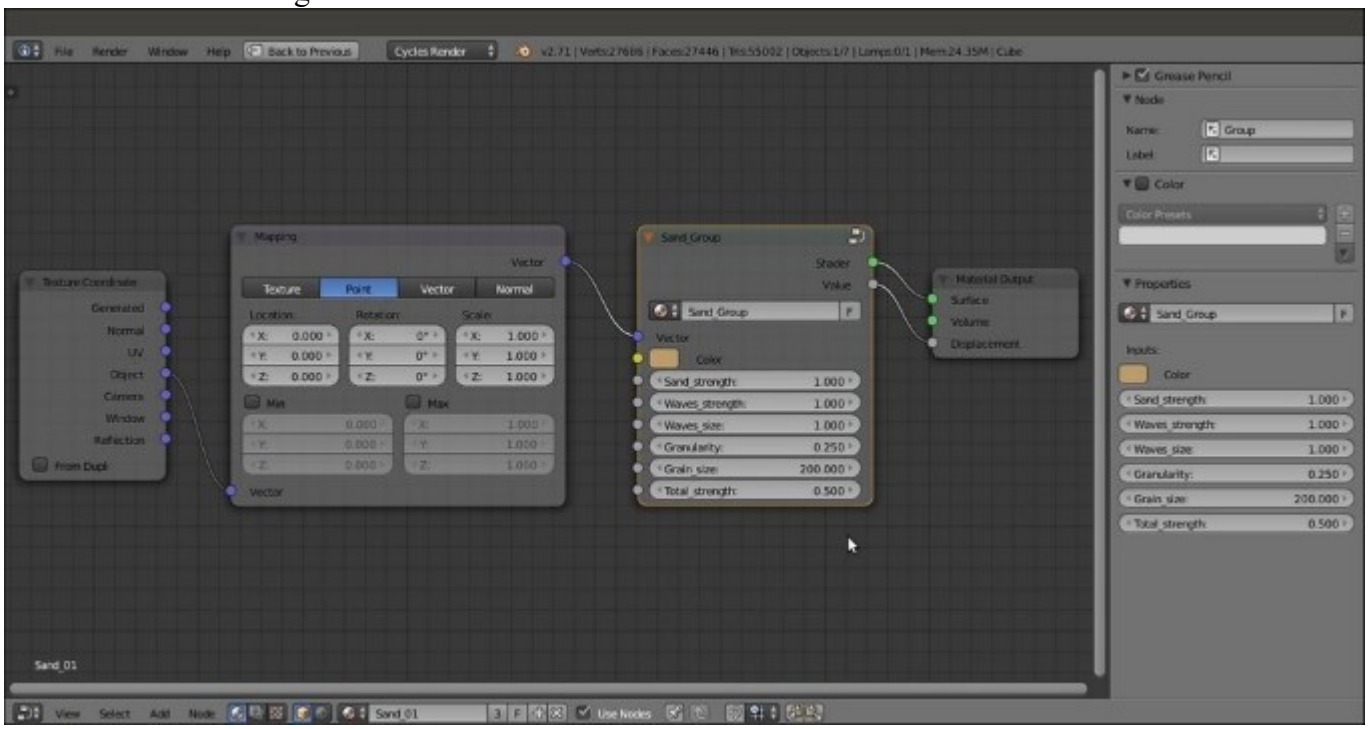
All the Vector input sockets of the nodes are connected to a single socket on the Group Input node, and the Bump Strength value is exposed by a new connection

2. Repeat step 1 for the second **Value** socket of the **Waves_Strength** and the **Grain_Strength** nodes, and rename the respective input as `Waves_strength` and `Granularity`.
3. Now click on the **Waves_size** node inside the **BIG WAVES** frame and delete it. Click and drag the second **Value** socket of the **Multiply01** node to the **Group Input** node, and rename the new socket `Waves_size`. Click and drag a link from the second **Value** socket of the **Multiply02** node, and connect it to the **Waves_size** socket as well.
4. We also need to expose the second **Value** socket of the **TOTAL BUMP STRENGTH** frame. Rename the new socket on the interface as `Total strength`. This is in fact the value for the overall bump of the material.
5. After this, we can do the following: expose the color input by deleting the **RGB** node in the **SAND COLOR** frame and connecting the **Color** input sockets of the **Diffuse BSDF**, **RGB Curves**, and **Glossy BSDF** shader node's to the **Color** socket on the **Group Input** node; expose the sand's grain size value, connecting the **Scale** input socket of the **Noise Texture03** node to a **Grain_size** socket; and finally, by clicking on the arrows in the **Properties** panel, order the position of the input sockets on the **Group Input** node as shown in this screenshot:



The sockets created on the Group Input node and reflected in the Interface subpanel

6. Press *Tab* to close the group. On the interface, we now have the controls to increase or decrease the overall bump effect, the sand color and grain, the wave strength, and scale/repetition, as we can see in the following screenshot:

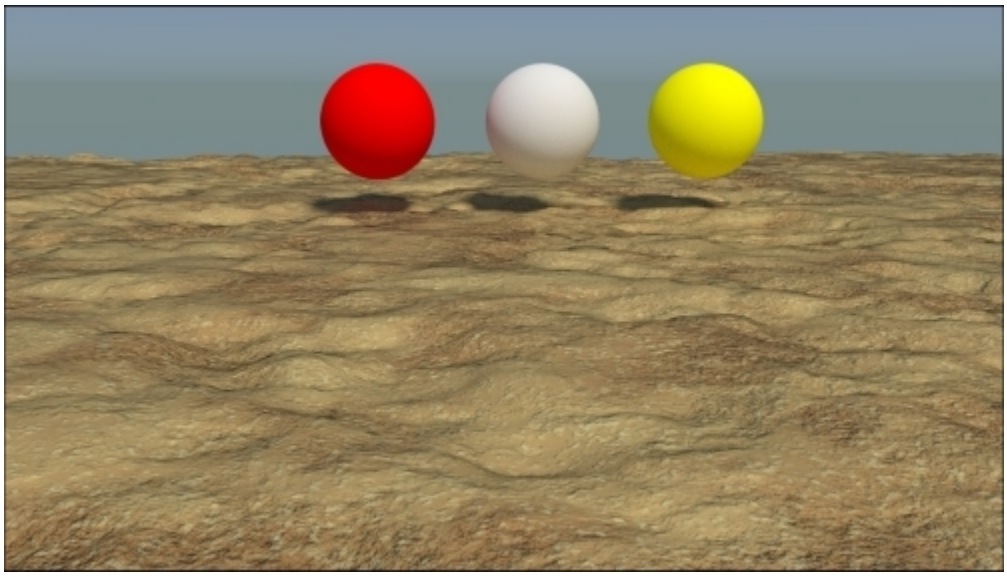


The final Sand_Group node with all the exposed input on its interface

The group is now available under the **Add** menu, and its shortcut involves pressing *Shift* + *A* and navigating to **Group** | **Sand_Group**. It can be reused for other materials in the same scene and also with different interface values, or linked/appended from a library in other blend files.

Creating a simple ground material using procedural textures

In this recipe, we will create a basic, raw ground material as shown in this screenshot:



The ground material as it appears in the final rendering

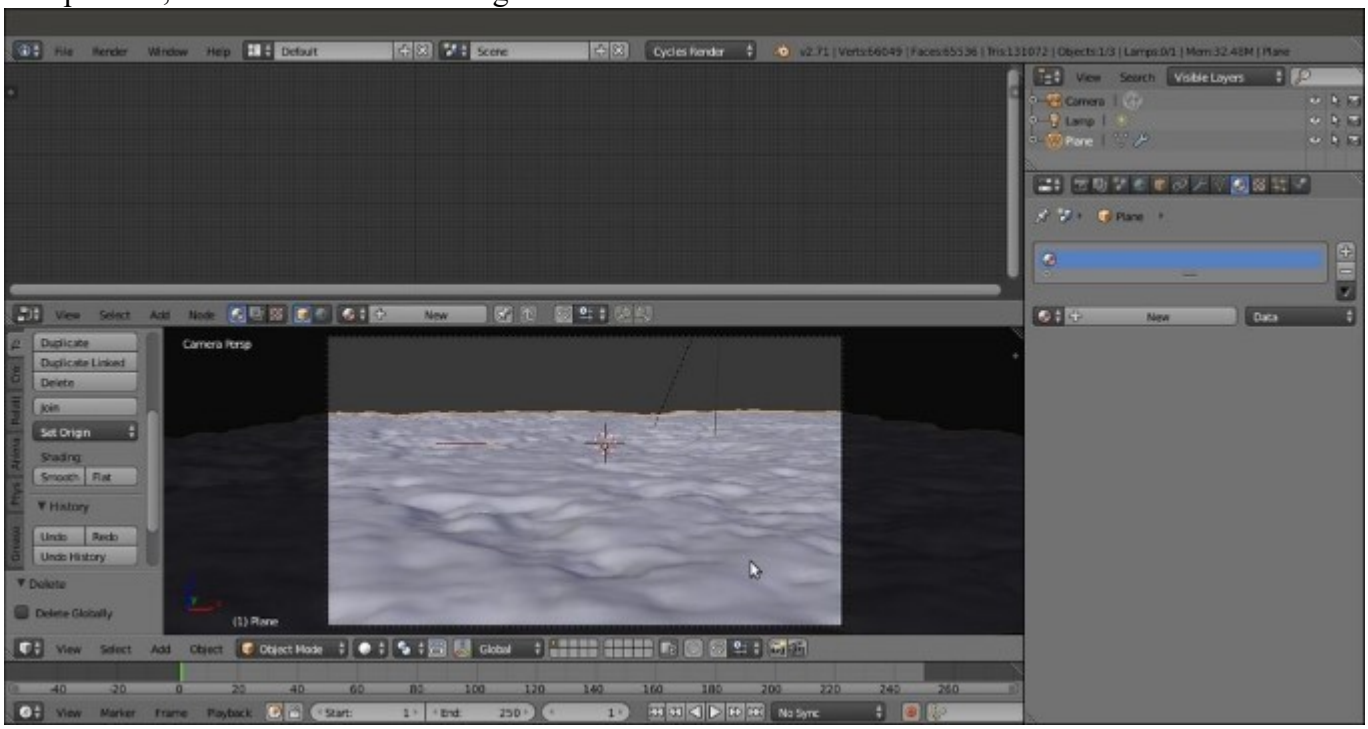
Getting ready

Start Blender and switch to **Cycles Render**. Then perform the following steps:

1. Delete the default Cube and add a Plane. Go to **Edit Mode** and scale it 15 times bigger (30 units per side; press *Tab*, then press *S*, enter the digit *15*, and press *Enter*). Go out of **Edit Mode**.
2. Go to the **Object modifiers** window and assign a **Subdivision Surface** modifier to the Plane. Switch from **Catmull-Clark** to **Simple**, and set the levels of **Subdivisions** for both **View** and **Render** to `4`. Check the **Optimal Display** item.
3. Assign a second **Subdivision Surface** modifier. Again, switch to **Simple**, set the levels of **Subdivisions** for both **View** and **Render** to `4`, and check the **Optimal Display** item.
4. Assign a **Displace** modifier. Click on the **Show texture in texture tab** button to the side of the **New** button. In the **Texture** window, click on the **New** button, select **Voronoi** texture, and increase the **Size** value to `1.80`. Go back to the **Object modifiers** window and set the displacement **Strength** to `0.100`.
5. Assign a second **Displace** modifier and select the default **Clouds** texture. Set the **Size** value to `0.75`, the **Depth** value to `5`, and the displacement **Strength** value to `0.150`.
6. Assign a third **Displace** modifier. Again, select the default **Clouds** texture and increase the **Size** value to `4.00` (the slider arrives at a maximum of `2.00`, but you can click on the value and enter higher values) and the **Depth** value to `4`. Then switch **Noise** from **Soft** to **Hard** and click

on the **Basis** button to change **Noise Basis** from **Blender Original** to **Voronoi F4**. Go to the **Colors** subpanel above the **Clouds** subpanel, and adjust the **Brightness** value to `0.900` and the **Contrast** value to `1.500`. Set the displacement strength to `0.500`.

7. In the **Shading** subpanel (which is accessible from the **Transform** menu), under the **Tool Shelf** tabs to the left of the 3D view, click on the **Smooth** button.
8. Go to the **World** window and click on the **Use Nodes** button in the **Surface** subpanel under the **Properties** panel. Then click on the little square with a dot on the right side of the **Color** slot. From the menu, select **Sky Texture**. Set the **Strength** value to `1.400`.
9. Go to the **Outliner** and select the **Lamp** item. Go to the **Object data** window and click on **Use Nodes**. Then change **Type of Lamp** to **Sun** and set the **Strength** value to `1.400`. Change the light color values to `1.000` for **R**, `0.935` for **G**, and `0.810` for **B**. In the orthogonal top view (press the *7* and *5* keys in the numeric keypad), rotate the Sun Lamp by `90°`.
10. Place the **Camera** to have a nice angle on the Plane (you can also use the **Lock Camera to View** item in the (press *N*) **Properties** side panel), and switch from the 3D view to the **Camera** view (by pressing *0* from the numeric keypad).
11. Split the 3D window into two horizontal rows. Change the upper row to a **Node Editor** window.
12. Go to the **Render** window, and under the **Sampling** subpanel, set both the **Clamp Direct** and **Clamp Indirect** values to `1.000`. Go to the **Light Path** subpanel and set the **Filter Glossy** value to `1.000`.
13. Reselect the **Plane** and go to the **Material** window under the **Properties** panel. Disable the transformation widget by clicking on the icon in the 3D window toolbar or by pressing *Ctrl* and the spacebar, as shown in the following screenshot:



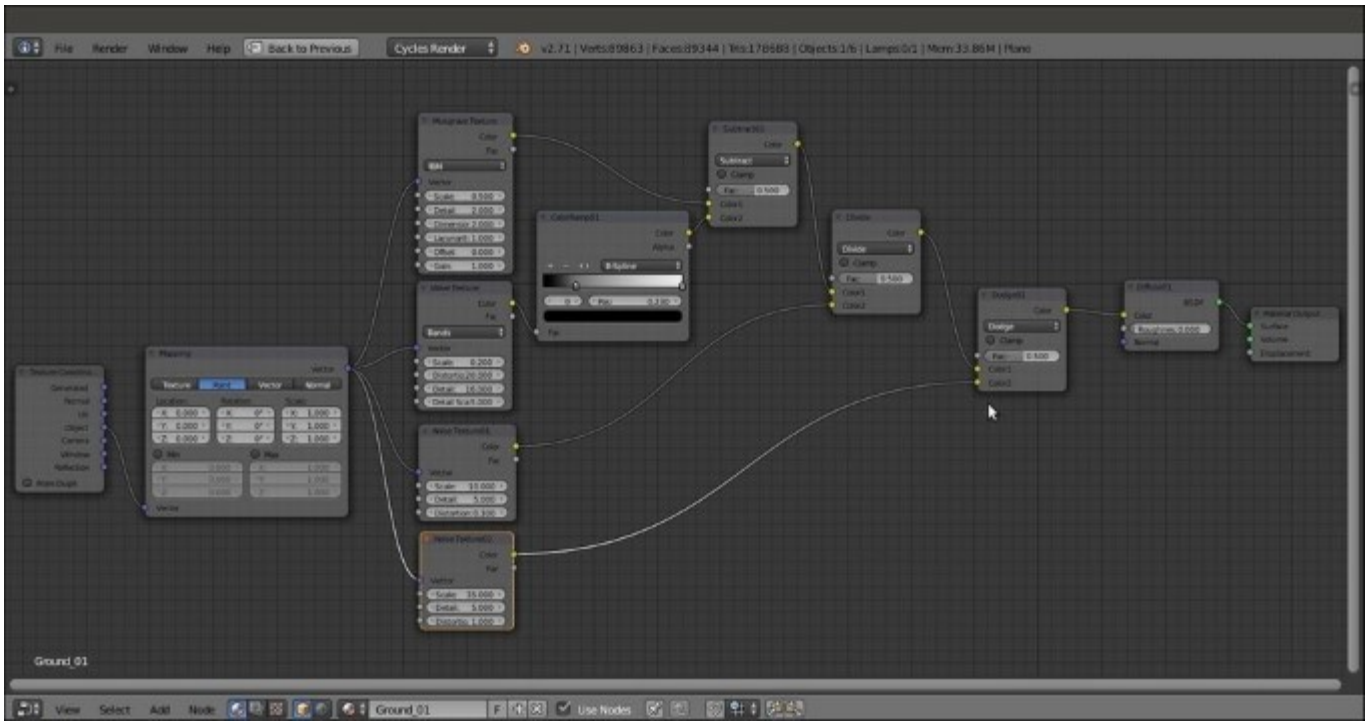
Screenshot in the Solid viewport shading mode of the ground scene

In the final scene, I added three UV Spheres with simple diffuse colors, just for lighting reference. Obviously, you can skip this step.

How to do it...

Let's now start with the ground material:

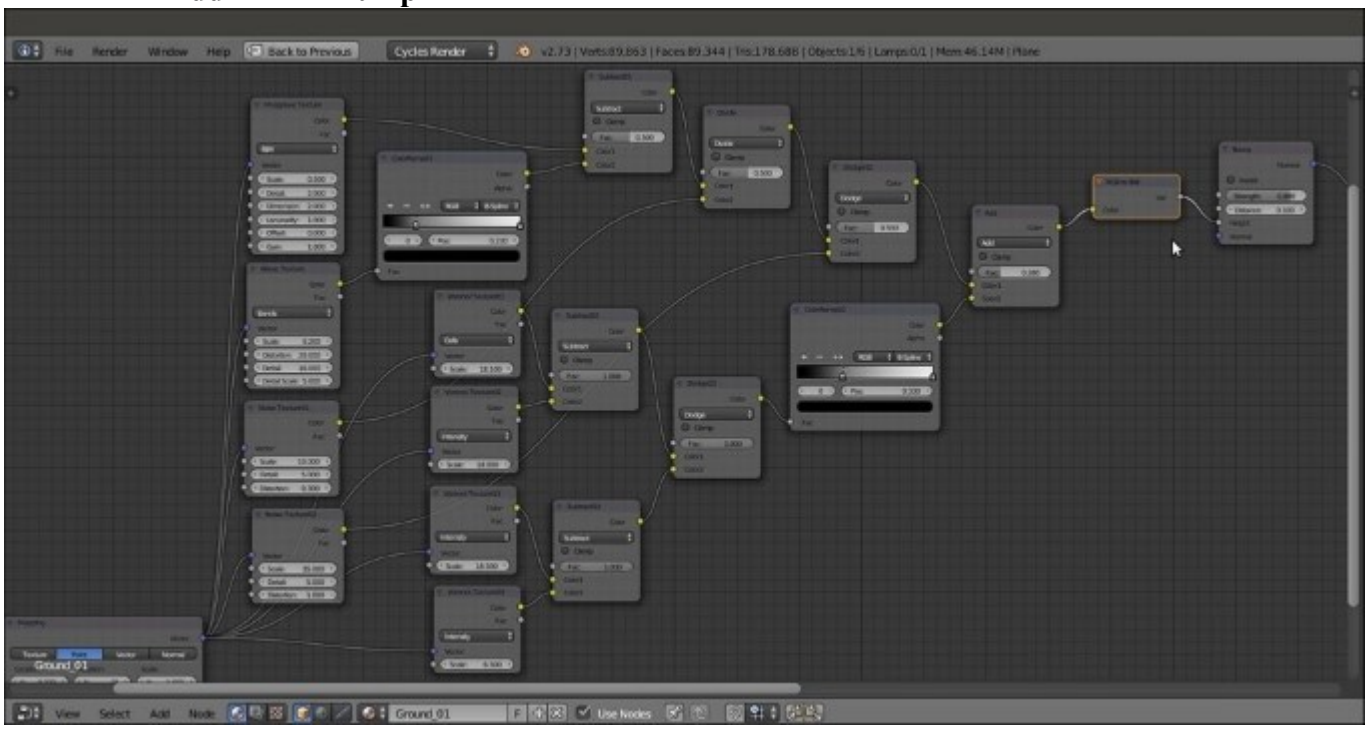
1. Put the mouse cursor in the **Camera** view and press *Shift* + *Z* to switch the **Viewport Shading** mode to **Rendered**.
2. Click on the **New** button in the **Material** window or in the **Node Editor** toolbar. Rename the material `Ground_01`.
3. In the **Node Editor** window, add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and a **Musgrave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Musgrave Texture**).
4. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node and the **Vector** output of this node to the **Vector** input of the **Musgrave Texture** node.
5. Connect the **Color** output of the **Musgrave Texture** node to the **Color** input of the **Diffuse BSDF** shader. In the **Properties** panel, label the **Diffuse BSDF** shader as `Diffuse01`. Set the **Scale** value of the **Musgrave Texture** node to `0.500`.
6. Add a **Wave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**) and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Paste the **MixRGB** node between the **Musgrave Texture** node and the **Diffuse01** shader node, and connect the **Wave Texture** node's color output to the **Color2** input socket of the **MixRGB** node.
7. Set the **MixRGB** node's **Blend Type** to **Subtract** and label it `Subtract01`. Connect the **Mapping** output to the **Wave Texture** node's **Vector** input.
8. Set the **Wave Texture** node's **Scale** value to `0.200`, **Distortion** to `20.000`, **Detail** to `16.000`, and **Detail Scale** to `5.000`.
9. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and drag it onto the link connecting the **Wave Texture** node to the **Subtract01** node to paste it between them. Label it `ColorRamp01`, change the **Interpolation** mode to **B-Spline**, and move the black stop to the `0.230` position (**Pos:**).
10. Add two **Noise Texture** nodes (press *Shift* + *A*, navigate to **Texture** | **Noise Texture**, and then press *Shift* + *D* to duplicate it) and name them `Noise Texture01` and `Noise Texture02`. Connect the **Mapping** node to them. Select the **Subtract01** node, press *Shift* + *D* to duplicate it twice, and change **Blend Type** to **Divide** and **Dodge**. Connect the **Color** output of the **Subtract01** node to the **Color1** input of the **Divide** node, and connect the **Color** output of the **Noise Texture01** node to the **Color2** input of the **Divide** node.
11. Then connect the **Color** output of the **Divide** node to the **Color1** input of the **Dodge** node, and the **Color** output of the **Noise Texture02** node to the **Color2** input of the **Dodge** node.
12. Label the **Dodge** node `Dodge01` and connect its output to the **Color** input of the **Diffuse01** shader. Set the **Noise Texture01** scale to `10.000`, **Detail** to `5.000`, and **Distortion** to `0.300`. For **Noise Texture02**, set **Scale** to `35.000`, **Detail** to `5.000`, and **Distortion** to `1.000`, as shown in the following screenshot:



The first steps to build the bump effect for the ground material

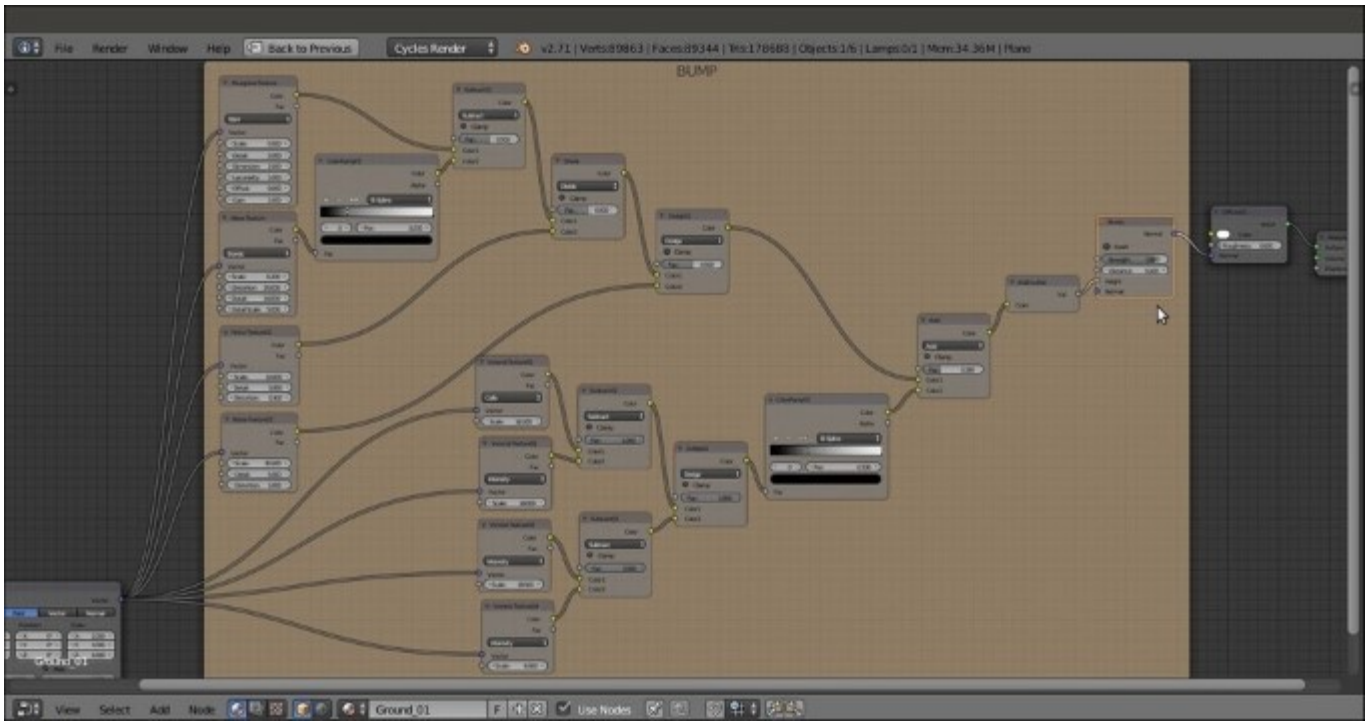
13. Add two **Voronoi Texture** nodes (press *Shift* + *A*, navigate to **Texture** | **Voronoi Texture**, and rename the nodes `Voronoi Texture01` and `Voronoi Texture02`) and a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Blend Type** to **Subtract** and label it `Subtract02`. Connect the color output of the **Voronoi Texture01** node to the **Color1** input socket, and the color output of the **Voronoi Texture02** node to the **Color2** input socket.
14. Set the **Subtract02** node's **Fac** value to `1.000`, and then go to the **Voronoi Texture01** node. Set **Coloring** to **Cells** and **Scale** to `18.100`. Go to the **Voronoi Texture02** node, leave **Coloring** as **Intensity**, and set the **Scale** value to `18.000`.
15. Select the two **Voronoi Texture** nodes and the **Subtract02** node, and press *Shift* + *D* to duplicate them. Label the texture nodes as `Voronoi Texture03` and `Voronoi Texture04`, and the **MixRGB** node as `Subtract03`.
16. Connect the **Mapping** node output to the **Vector** input sockets of the four **Voronoi Texture** nodes.
17. Change the **Coloring** of the **Voronoi Texture03** node back to **Intensity**, and set the **Scale** value to `18.500`. Set the **Scale** value of **Voronoi Texture04** to `6.500`.
18. Add a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and change the **Blend Type** to **Dodge**. Label it `Dodge02` and set the **Fac** value to `1.000`. Connect the output of the **Subtract02** node to the **Color1** input socket and the output of the **Subtract03** node to the **Color2** input socket.
19. Add a **MixRGB** node again (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Change the **Blend Type** to **Add** and paste it between the **Dodge01** node and the **Diffuse01** shader node. Then connect the output of the **Dodge02** node to the **Color2** input socket.

20. Disconnect the link between the **Add** output and the **Color** input socket of the **Diffuse01** shader node, and add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the output of the **Add** node to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of the **Bump** node to the **Normal** input socket of the **Diffuse01** shader. Set the **Add** node's **Fac** value to `0.280` and the **Bump** node's **Strength** value to `0.800`.
21. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and label it `ColorRamp02`. Paste it between the **Dodge02** and the **Add** nodes. Set **Interpolation** to **B-Spline** and move the black color slider and stop at position `0.330`.
22. Add a **RGB to BW** node (press *Shift* + *A* and navigate to **Converter** | **RGB to BW**) and paste it between the **Add** and the **Bump** nodes.



The total BUMP network for the ground material

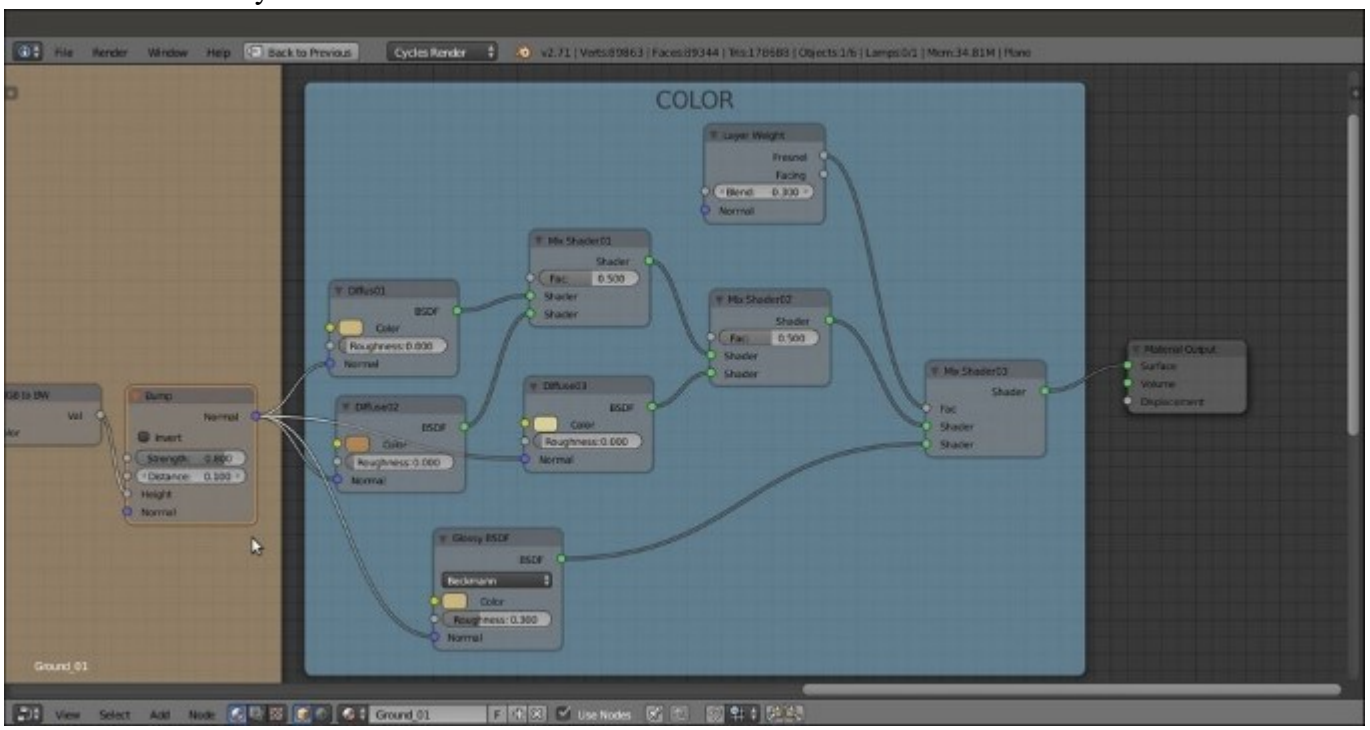
23. Parent the nodes to a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**) and label it `BUMP`, as shown in the following screenshot:



The BUMP frame

24. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and a second **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**). Name them `Mix Shader01` and `Diffuse02`, respectively. Then paste the **Mix Shader01** node between the **Diffuse01** and the **Material Output** nodes, and connect the **Diffuse02** node to the second **Shader** input socket of the **Mix Shader01** node.
25. Once again, add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and a **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**). Label them `Mix Shader02` and `Diffuse03`. Then paste the **Mix Shader02** node between the **Mix Shader01** node and the **Material Output** node, and connect the **Diffuse03** node to the second **Shader** input socket.
26. Connect the **Bump** node output to the **Normal** input of the **Diffuse02** and **Diffuse03** shader nodes.
27. Change the **Diffuse01** color values to `0.593` for **R**, `0.460` for **G**, and `0.198` for **B**; the **Diffuse02** color values to `0.423` for **R**, `0.234` for **G**, and `0.092` for **B**; and the **Diffuse03** color values to `0.700` for **R**, `0.620` for **G**, and `0.329` for **B**.
28. Once more, add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and a **Glossy BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**). Label the first node `Mix Shader03` and paste it between the **Mix Shader02** and the **Material Output** nodes. Connect the **Glossy BSDF** shader to the second **Shader** input socket of the **Mix Shader03** node, and set its **Roughness** value to `0.300` and the color values to `0.593` for R, `0.460` for **G**, and `0.198` for **B**, just like the **Diffuse01** color.
29. Connect the **Bump** node output to the **Normal** input socket of the **Glossy BSDF** node.

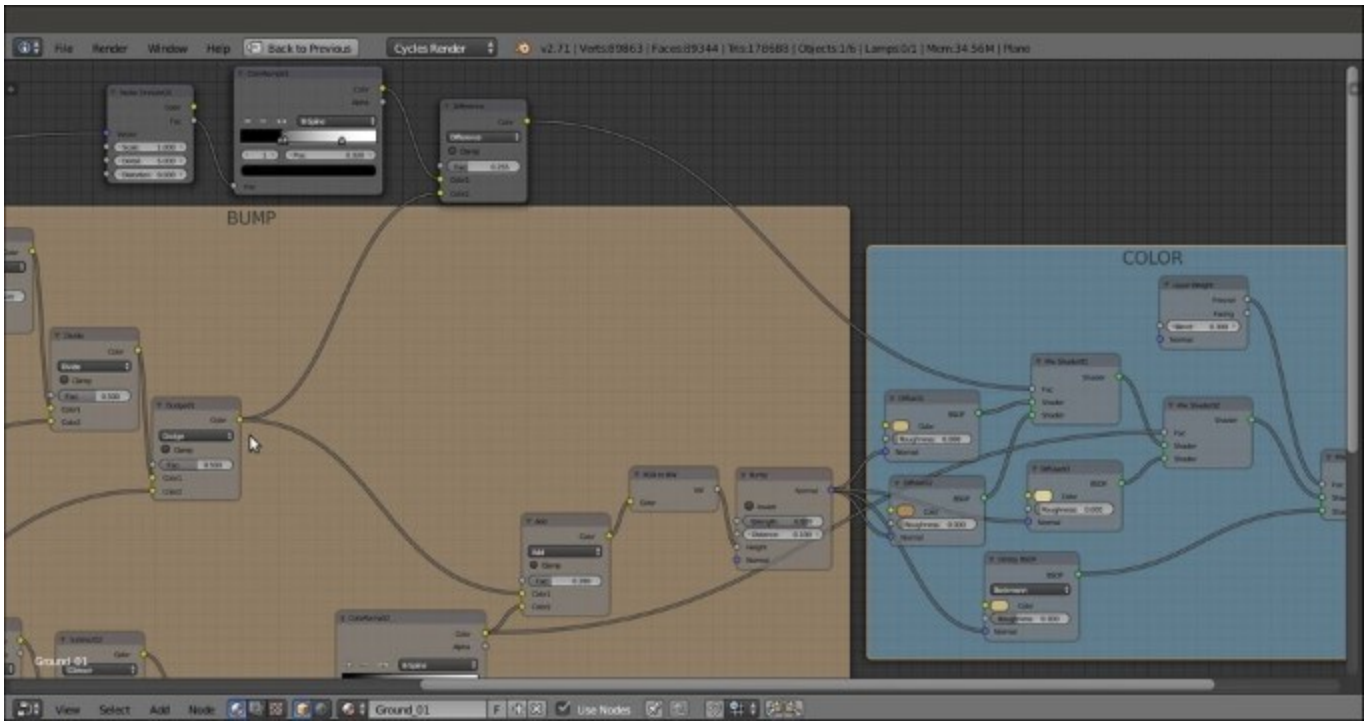
30. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), connect the **Fresnel** output to the **Fac** input socket of the **Mix Shader03** node, and set the **Blend** value to `0.300`.
31. Parent these recently added nodes to a new **Frame** and label it `COLOR`.



The COLOR frame

32. Add one more **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and a new **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Mapping** node output to the **Noise Texture** node's **Vector** input (label it `Noise Texture03`) and the **Fac** output of **Noise Texture** to the **Fac** input of the **ColorRamp** node (label it `ColorRamp03`).
33. For the last time, add a **MixRGB** node and set the **Blend Type** to **Difference**. Then connect the **Color** output of the **ColorRamp03** node to the **Color1** input socket of the **Difference** node, and the **Color** output of the **Difference** node to the **Fac** input socket of the **Mix Shader01** node. Set the **Fac** value of the **Difference** node to `0.255`.
34. Set the **Noise Texture03** node's scale to `1.000` and the **Detail** value to `5.000`. Switch the **ColorRamp03** node's **Interpolation** to **B-Spline**, move the **0** value of **color stop** to position `0.285`, move the **1** color stop to position `0.740`, and click on the **+** icon to add a new color stop. Set its color to black and move it to position `0.320`.
35. Connect the output of the **Dodge01** node inside the **BUMP** frame to the **Color2** input socket of the **Difference** node. Connect the **Color** output of the **ColorRamp02** node inside the **BUMP** frame to the **Fac** input socket of the **Mix Shader02** node inside the **COLOR** frame.

And we're done! Here is a screenshot of what the Blender UI will now look like:

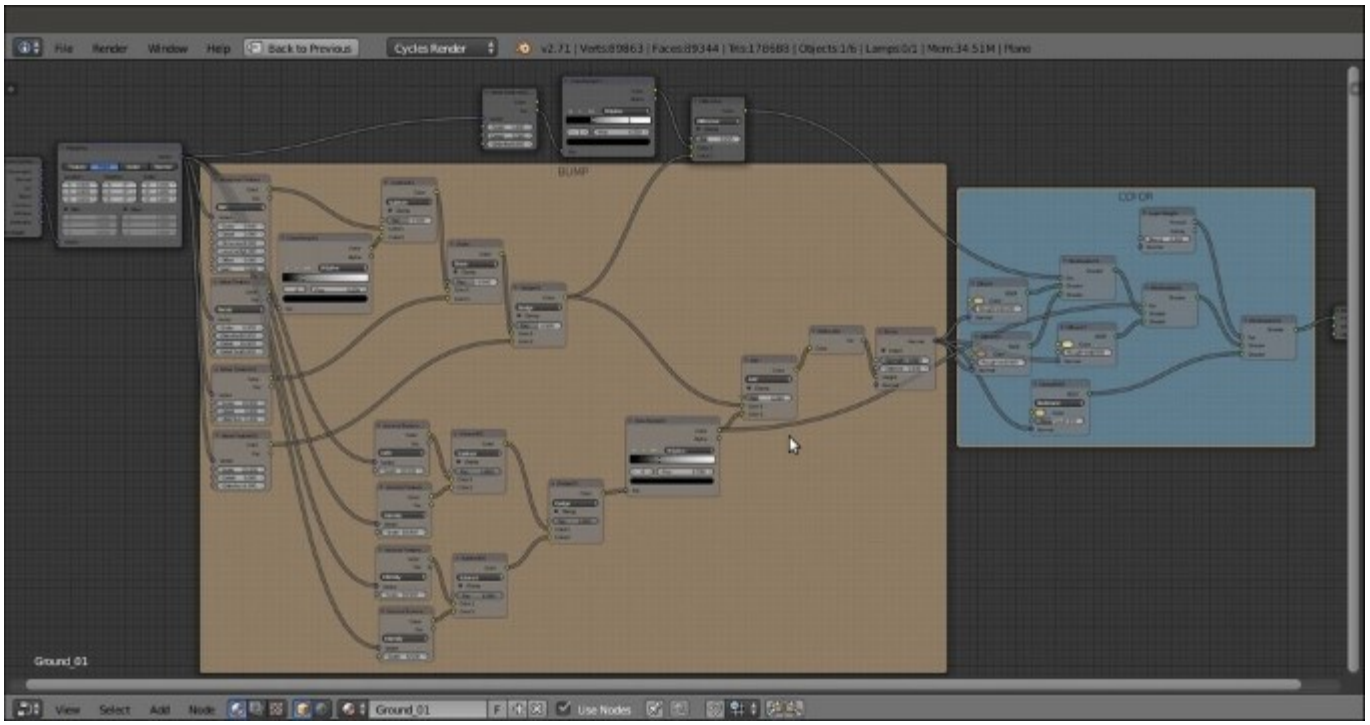


Part of the bump output is connected to the COLOR frame by the three upper nodes and the bottom ColorRamp node

How it works...

The way this material works is very similar to the sand material of the previous recipe, although a lot simpler:

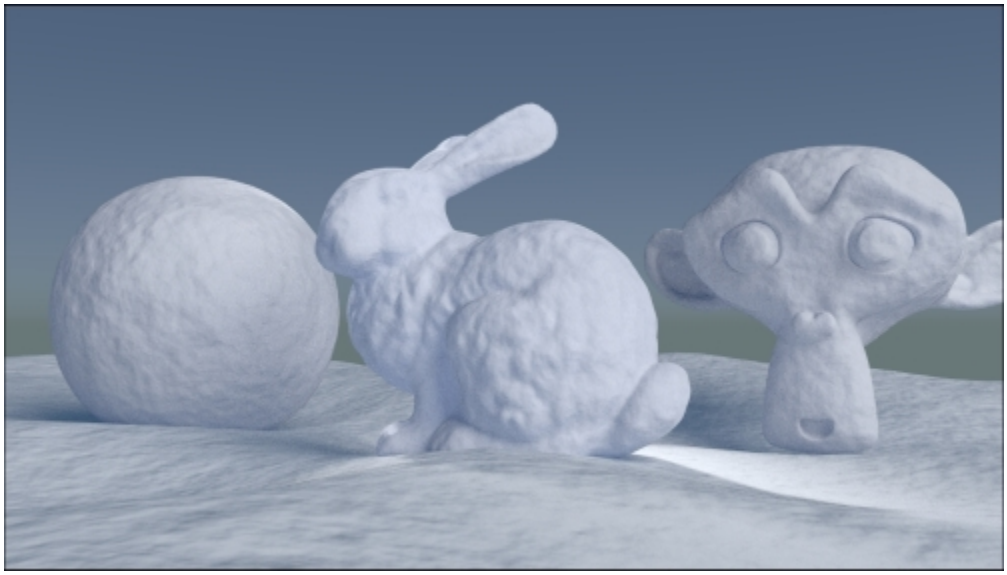
- We mixed two slightly different colors using the values of a **Noise Texture** node as the stencil factor, then mixed a third, similar color on the ground of the bump output to obtain the whitish, pebble-like effect you see in the rendered image. We created the ground roughness using an ensemble of procedural textures mixed in several ways, whose total sum was then connected to the **Normal** input sockets of the three **Diffuse BSDF** nodes and of the **Glossy BSDF** shader, as shown in the following screenshot:



The overall vision of the ground material network

Creating a snow material using procedural textures

In this recipe, we will create a snow material, as shown in the following screenshot, and also fake a slight and cheap Subsurface Scattering effect:



The snow material as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_Snow_start.blend` file.

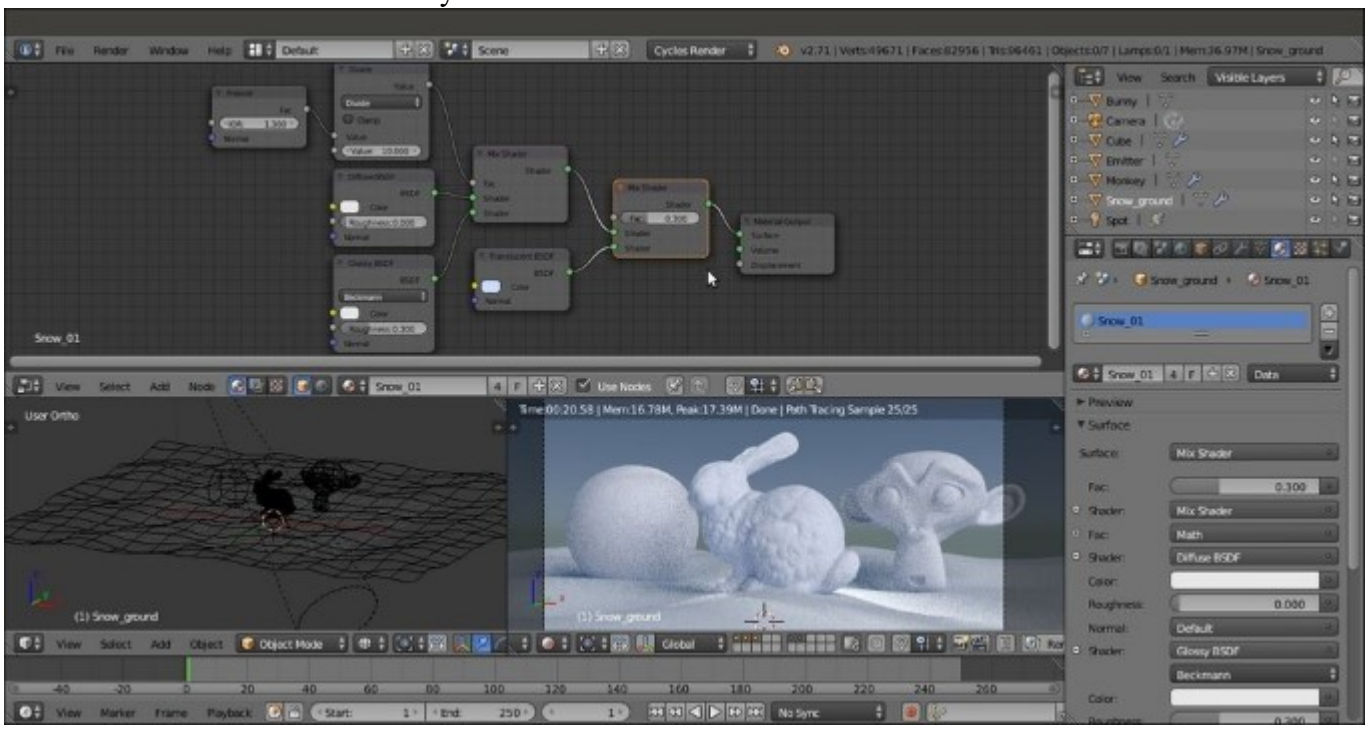
In this file, there is a prepared scene with a Spheroid (the usual Cube with a four-level **Subdivision Surface** modifier), a Suzanne (press *Shift* + *A* and navigate to **Add** | **Mesh** | **Monkey**) with a **Subdivision Surface** modifier as well, and the famous Stanford bunny (http://en.wikipedia.org/wiki/Stanford_bunny), leaning on a subdivided, displaced, and smoothed Plane renamed `Snow_ground`. Suzanne is Blender's mascot and an alternative to free test models such as the Stanford bunny itself. By the way, I thought of grouping them in the same scene to have different shapes to test the material.

In the file, there is also a Plane working as mesh-light and a Spot pointing in the opposite direction to try to enhance the translucency of the snow.

How to do it...

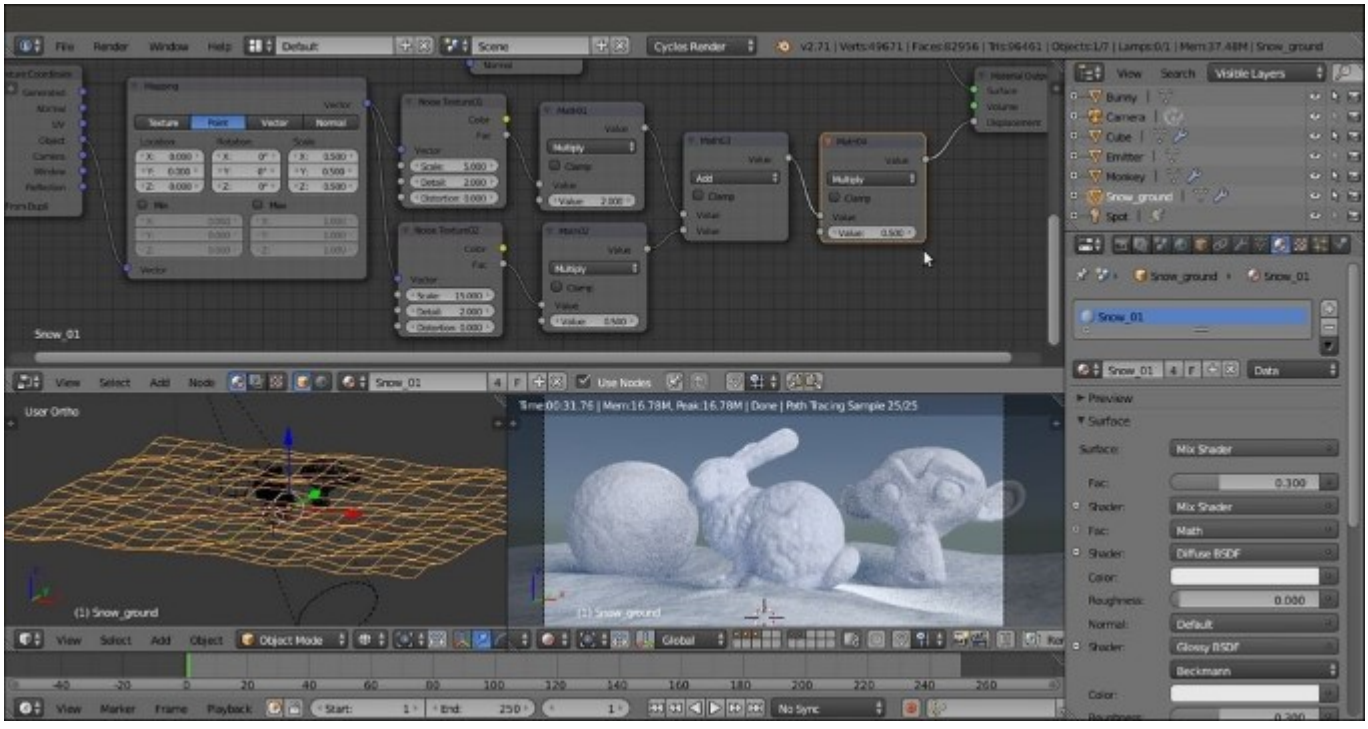
Let's start creating the snow material:

1. Go to the **World** window and click on the **New** button. Then click on the little square with a dot on the right side of the **Color** slot. From the menu, select **Sky Texture**.
2. Go to the **Material** window and select the **Snow_ground** item in the **Outliner**. Click on the **New** button in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material as `Snow_01`.
3. Press *Shift* and select the Spheroid, Suzanne, and the Stanford bunny. Then select the Plane to have it as the active object. Press *Ctrl* + *L* and go to **Materials**.
4. Put the mouse cursor in the **Camera** view and press *Shift* + *Z* to set the **Viewport Shading** mode to **Rendered**.
5. In the **Material** window under the **Properties** panel to the right, under the **Surface** subpanel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** shader, and in the second slot, select a **Glossy BSDF** shader.
6. Set the **Roughness** value of the **Glossy BSDF** shader to `0.300`.
7. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set the **IOR** (short for **Index Of Refraction**) value of the **Fresnel** node to `1.300`. Then connect its **Fac** output to the first **Value** socket of the **Math** node. Set the second **Value** to `10.000` and the operation mode to **Divide**. Finally, connect its **Value** output to the **Fac** input socket of the **Mix Shader** node.
8. Add a **Translucent BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Translucent BSDF**). Set its color values to `0.598` for **R**, `0.721` for **G**, and `1.000` for **B**.
9. Select the **Mix Shader** node, press *Shift* + *D* to duplicate it, and paste it between the first **Mix Shader** node and the **Material Output** node. Connect the **Translucent BSDF** node's output to the second input socket. Set the **Fac** value of the second **Mix Shader** node to `0.300`. Here is a screenshot of the basic shader for your reference:



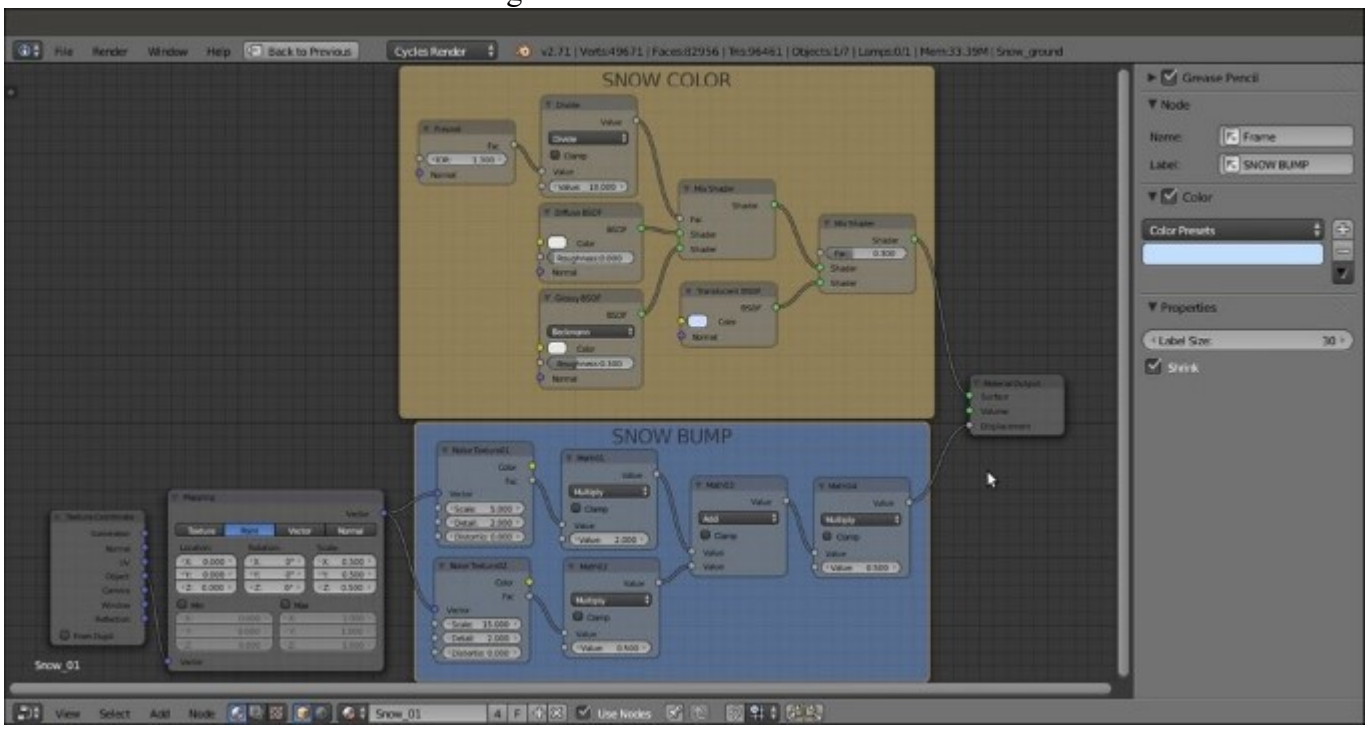
The basic shader for the snow material

10. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and press *Shift* + *D* to duplicate it. In the **Properties** panel of the **Node Editor** window (press the *N* key to make this appear if necessary), label them `Noise Texture01` and `Noise Texture02`.
11. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node. Then connect the **Vector** output of the **Mapping** node to both the **Vector** input sockets of the **Noise Texture** nodes.
12. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and press *Shift* + *D* to duplicate it three times so that you obtain four **Math** nodes. Label them `Math01`, `Math02`, `Math03`, and `Math04`.
13. Connect the **Noise Texture01** node's **Fac** output (the gray one) to the first **Value** input of the **Math01** node and set the second **Value** to `2.000`. Set the **Operation** to **Multiply**.
14. Connect the **Fac** output of the **Noise Texture02** node to the first **Value** input of the **Math02** node and let its second **Value** be the default, which is `0.500`. Set the **Operation** to **Multiply**.
15. Now connect both the output of the two previous **Math** nodes to the input **Value** sockets of the **Math03** node. Set the **Operation** to **Add**.
16. Connect the output of the **Math03** node to the first **Value** input of the **Math04** node. Set its **Operation** to **Multiply** and let the second **Value** be the default, which is `0.500`.
17. Connect the **Math04** node output to the **Displacement** input socket of the **Material Output** node.
18. Now go to the **Noise Texture02** node and change the **Scale** value to `15.000`. Leave the other values (also for the **Noise Texture01** node) as they are (that is, `5.000` for **Scale**, `2.000` for **Detail**, and `0.000` for **Distortion**).
19. Go to the **Mapping** node and set the **Scale** value to `0.500` for all three axes. Now the Blender UI will look like what is shown in this screenshot:



The bump pattern

20. Add two **Frames**, label them `SNOW COLOR` and `SNOW BUMP`, and parent the appropriate nodes to them as shown in the following screenshot:



The overall vision of the snow material network

How it works...

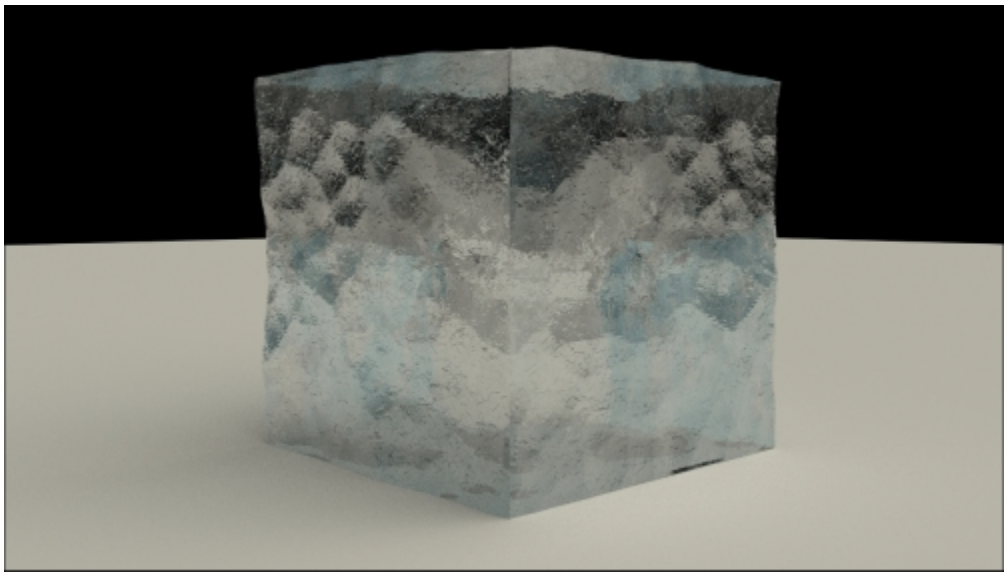
As usual, to understand the creation of this material more easily, we will divide it into two stages: the first stage for the general color and consistency of the snow, and the second stage to add bumpiness to the surface. These stages are explained in detail as follows:

- **First stage**: We just made a basic shader by mixing the **Diffuse BSDF** and the **Glossy BSDF** shaders by the **IOR** value of the **Fresnel** node. The **Fresnel** output value is divided by the **Math-Divide** node to obtain a softer transition (try to change the second value from `10.000` to `1.000` to see a totally different effect). Then we also mixed a bluish **Translucent** shader but gave predominance to the basic shader by setting the factor value in the second **Mix Shader** node to `0.300`. The **Translucent** shader gives the appearance of light seeping through snow and showing in the shadowed areas of the object, working as a very fast and cheap Subsurface Scattering effect.
- **Second stage**: We added two **Noise Texture** nodes with different scale values to simulate the bumpiness of soft snow. The first two **Multiply-Math** nodes set the influence of each noise separately. These values were merged by the **Add-Math** node and piped in one more **Math** node, set to **Multiply** as well, to establish the overall weight of the bump effect that, being

directly connected to the **Displacement** input in the **Material Output** node, affects all the shaders in the network.

Creating an ice material using procedural textures

In this recipe, we will create a semi-transparent ice material that will look like this:

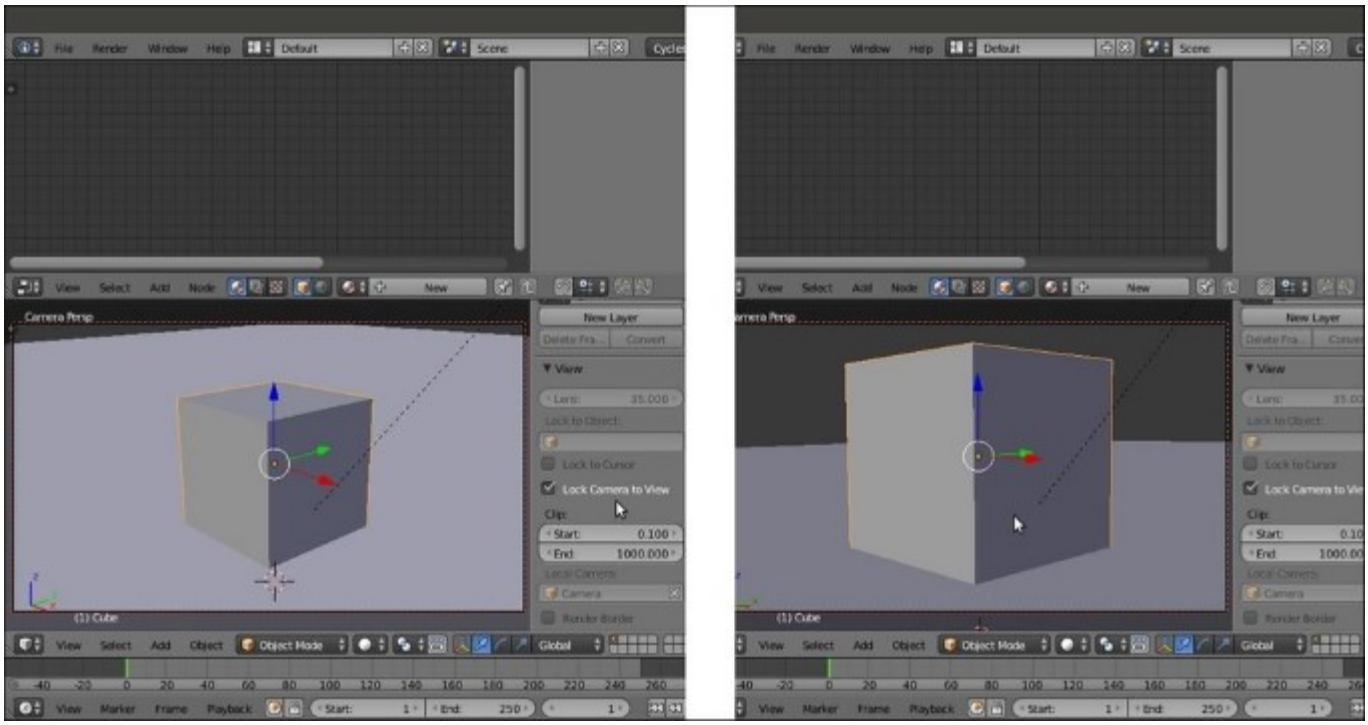


The ice material as it appears in the final rendering

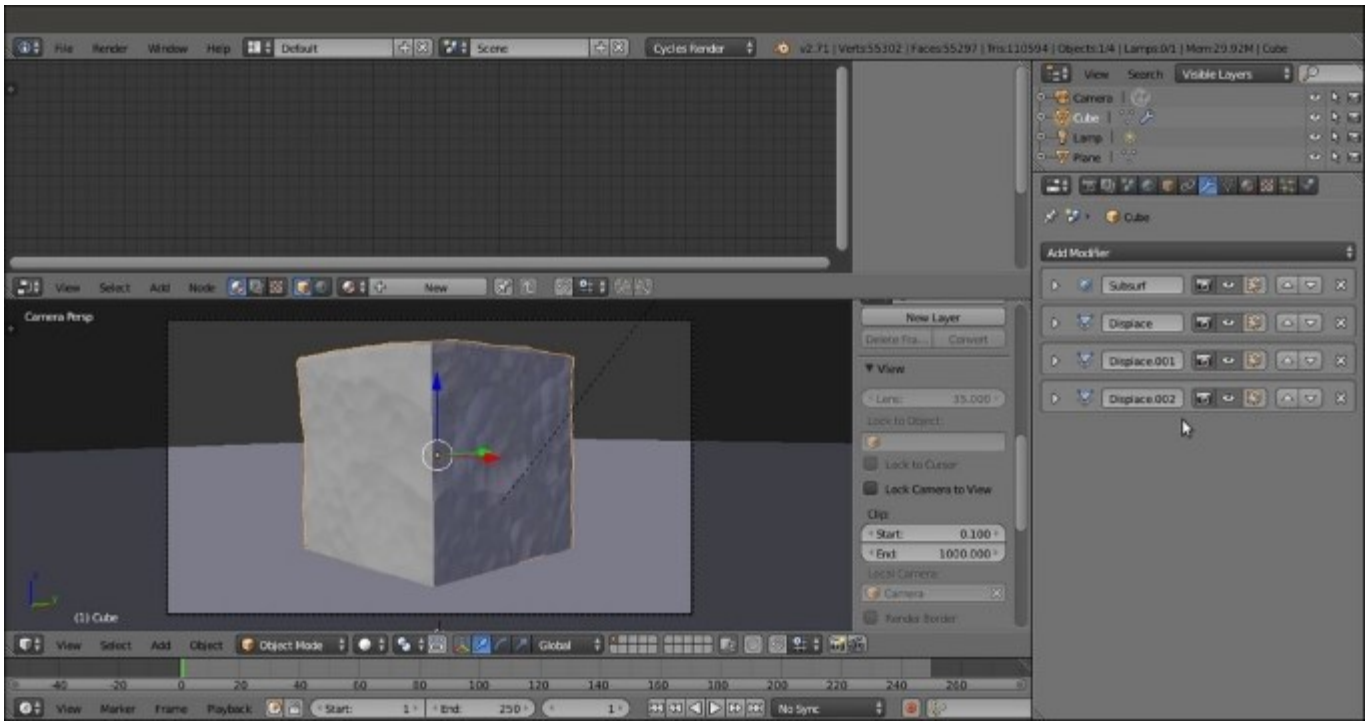
Getting ready

Start Blender, load the `9931OS_start.blend` file, and perform the following steps:

1. Delete the **UV/Image Editor** window by joining it with the 3D view.
2. Select the **Plane** item, go to **Edit Mode**, and scale it eight times bigger (press *Tab*, then press *S*, enter the digit *8*, and press *Enter*). Go out of **Edit Mode** and move the Plane 1 unit upward (press *Tab*, then press *G*, enter the digit *1*, press *Z*, and finally, press *Enter*).
3. Select the **Cube** and press *N* to make the **Properties** panel visible. Go to the **View** subpanel and check the **Lock Camera to View** item. The borders of the **Camera** view turn red, which mean that you can directly use the mouse to move, zoom in, and adjust the position of the Camera around the selected object (the Cube in this case) to obtain a view similar what is shown in the right half of this screenshot:



4. Next, uncheck the **Lock Camera to View** item.
5. Go to the **World** window and set **Color** to black.
6. Select **Sun Lamp** in the **Outliner**, and in the **Object data** window, set the **Strength** value to `3.000`, **Size** to `1.000`, and the **Color** values to `0.900` for **R**, `0.872` for **G**, and `0.737` for **B**.
7. Select the **Cube**, go to **Edit Mode**, and press the *W* key. In the **Specials** pop-up menu, select **Subdivide**. Press the *F6* key, and in the **Subdivide** pop-up panel under the 3D Cursor position, set **Number of Cuts** to `2`. Go out of **Edit Mode**.
8. Go to the **Object modifier** window and assign a **Subdivision Surface** modifier to the Cube. Switch from **Catmull-Clark** to **Simple**. Set the **Subdivisions** levels to `5` for both **View** and **Render**. Check the **Optimal Display** item.
9. Assign a **Displace** modifier, and in the **Textures** window, click on **New** and select the **Voronoi** texture. Set the **Size** value to `1.20`. Back in the **Object modifiers** window, set the displacement **Strength** to `0.050`.
10. Assign a new **Displace** modifier and select **Voronoi** texture again, but this time, set the **Size** value to `0.80`. Set the displacement **Strength** value to `0.075`.
11. Assign a third **Displace** modifier, select the **Voronoi** texture, and leave the default size (`0.25`) as it is. Set the displacement **Strength** value to `0.020`. Here is a screenshot of the displaced Cube primitive for your reference:



A screenshot in the Solid viewport shading mode of the displaced Cube primitive

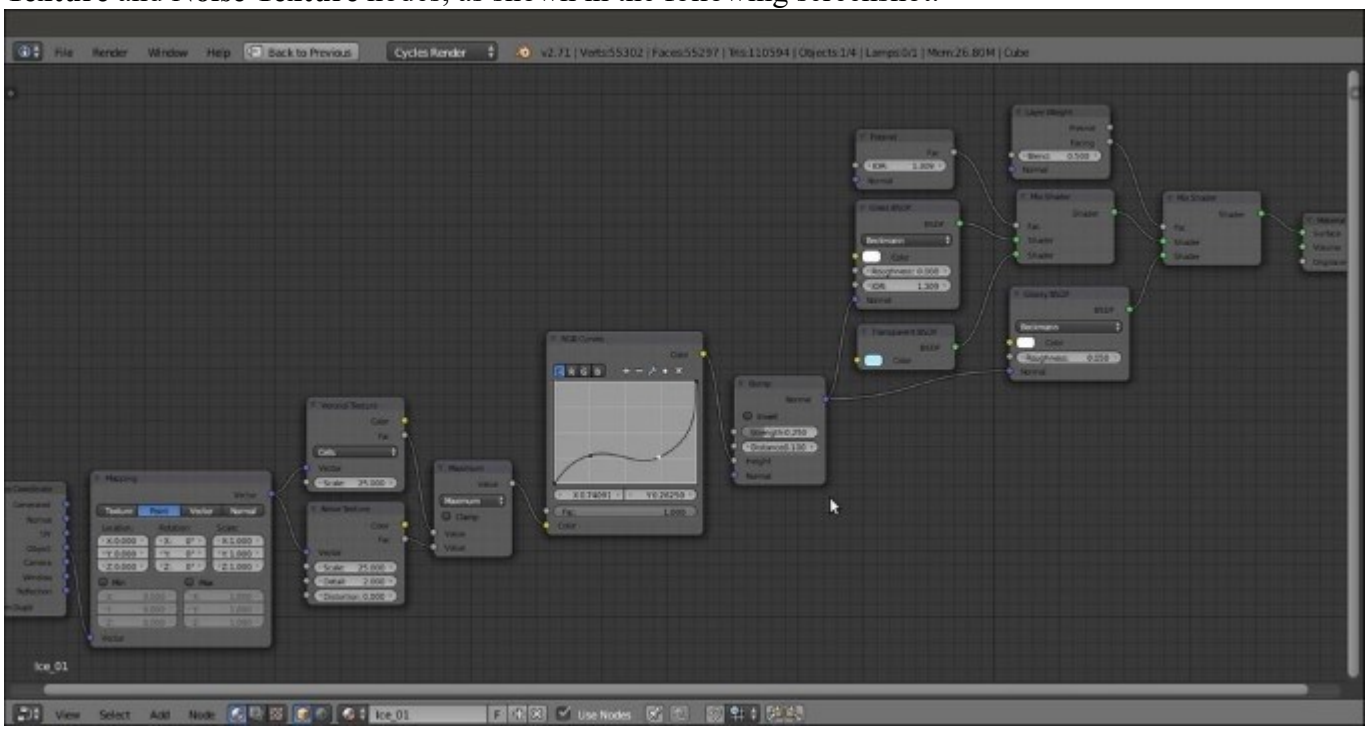
12. Switch the Camera's **Viewport Shading** to the **Rendered** mode.

How to do it...

After preparing the scene, we are going to create the material:

1. Select the **Cube** and click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Ice_01`.
2. In the **Material** window to the right of the screen, under the **Surface** subpanel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Glass BSDF** shader, and in the second slot, select a **Transparent BSDF** shader.
3. Set the **Glass BSDF** shader's color totally white and the **IOR** value to `1.309`. Set the **Transparent BSDF** shader's color values to `0.448` for **R**, `0.813` for **G**, and `1.000` for **B**.
4. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader** node. Then set the **IOR** value to `1.309`.
5. Add a **Glossy BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**). Set the color to pure white and the **Roughness** value to `0.050`.
6. Select the **Mix Shader** node and press *Shift* + *D* to duplicate it. Connect the output of the first `Mix Shader` node to the first `Shader` input socket of the duplicated one, and the **Glossy BSDF** shader output to the second **Shader** input socket. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect the **Facing** output to the **Fac** socket of the second **Mix Shader** node.

7. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**). Set **Coloring** to **Cells** and the **Scale** value to `25.000`.
8. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and set only the **Scale** value to `25.000`.
9. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and set **Operation** to **Maximum**. Connect the **Fac** output of the **Voronoi Texture** and **Noise Texture** nodes to the first and the second **Value** input of the **Math** node.
10. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Maximum-Math** node output to the **Height** input of the **Bump** node, and its **Normal** output to the **Normal** input sockets of the **Glass BSDF** and **Glossy BSDF** shaders.
11. Set the **Strength** value of the **Bump** node to `0.250`.
12. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**) and paste it between the **Maximum-Math** and the **Bump** nodes. Set the point in the little window of the node interface at these coordinates: `0.25455` for **X** and `0.28125` for **Y**. Click on the little window to create a new point and set its coordinates to `0.74091` for **X** and `0.26250` for **Y**.
13. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input**| **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node. Then connect the **Vector** output of the **Mapping** node to the **Vector** input sockets of both the **Voronoi Texture** and **Noise Texture** nodes, as shown in the following screenshot:



The quite simple network of nodes for the ice material

How it works...

This time, we started by mixing a **Glass BSDF** shader and a **Transparent** shader node, modulated by a **Fresnel** node, and we set the **IOR** values of both the **Fresnel** and the **Glass BSDF** to the refraction value of ice. We also added a **Glossy BSDF** shader to provide specularity, mixed by a **Layer Weight** node set on **Facing** (because the more a mesh normal faces the point of view, the more evident the specular effect is).

Then, using mixed procedural textures, we created the bump effect to perturb the surface of the object (note that the bump also affects the material's refraction).

See also

Here are some links to lists of IOR values that can be used in mixing the **Diffuse BSDF** component with the **Glossy BSDF** component through a **Fresnel** node:

- http://blenderartists.org/forum/showthread.php?71202-Material-IOR-Value-reference
- http://blenderartists.org/forum/showthread.php?117271-The-IOR-of-diferent-materials

The following is a computational and scientific search engine that allows you to quickly research the IOR of a given substance by typing its name:

- http://www.wolframalpha.com/

Chapter 4. Creating Man-made Materials in Cycles

In this chapter, we will cover the following recipes:

- Creating a generic plastic material
- Creating a Bakelite material
- Creating an expanded polystyrene material
- Creating a clear (glassy) polystyrene material
- Creating a rubber material
- Creating an antique bronze material with procedurals
- Creating a multipurpose metal node group
- Creating a rusty metal material with procedurals
- Creating a wood material with procedurals

Introduction

On most occasions, artificial materials are quite easy to recreate in Cycles.

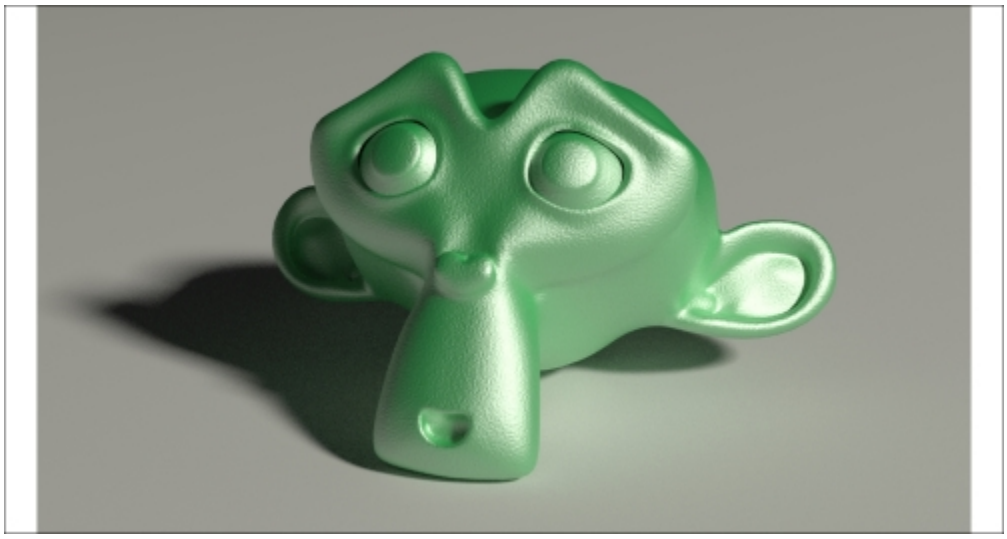
In the previous chapters we discussed the mechanics of building materials through procedural textures using the Cycles render engine. In this chapter, we'll discuss some artificial materials. Starting with one or two examples of simple materials, such as plastic, we will progress to more complex materials. We'll also take a look at the decayed material shaders and treat them as worn or rusty metals.

Note that in Cycles, it's not actually necessary to add the nodes for the texture mapping coordinates to any shader network. This is because, by default and if not otherwise specified, Cycles automatically uses the **Generated** mapping coordinates for procedural textures and any existing UV coordinate layer for the image textures.

Anyway, I think it's a good habit to add both the **Texture Coordinate** and the **Mapping** nodes to all the materials to permit easy reutilization of the shaders on different objects with different mapping options, scales, and locations.

Creating a generic plastic material

In this recipe, we will create a generic plastic shader and add slight granularity (optional) to the surface, as shown in the following screenshot:



The generic plastic material as it appears in the final rendering

Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file. This is a prepared scene, with Suzanne (the monkey head primitive that is Blender's mascot) leaning on a white Plane, a Camera, a mesh-light emitting slightly yellowish light, and a low-intensity gray World.

Note

We'll use a lot this file as starting point for several of our recipes.

How to do it...

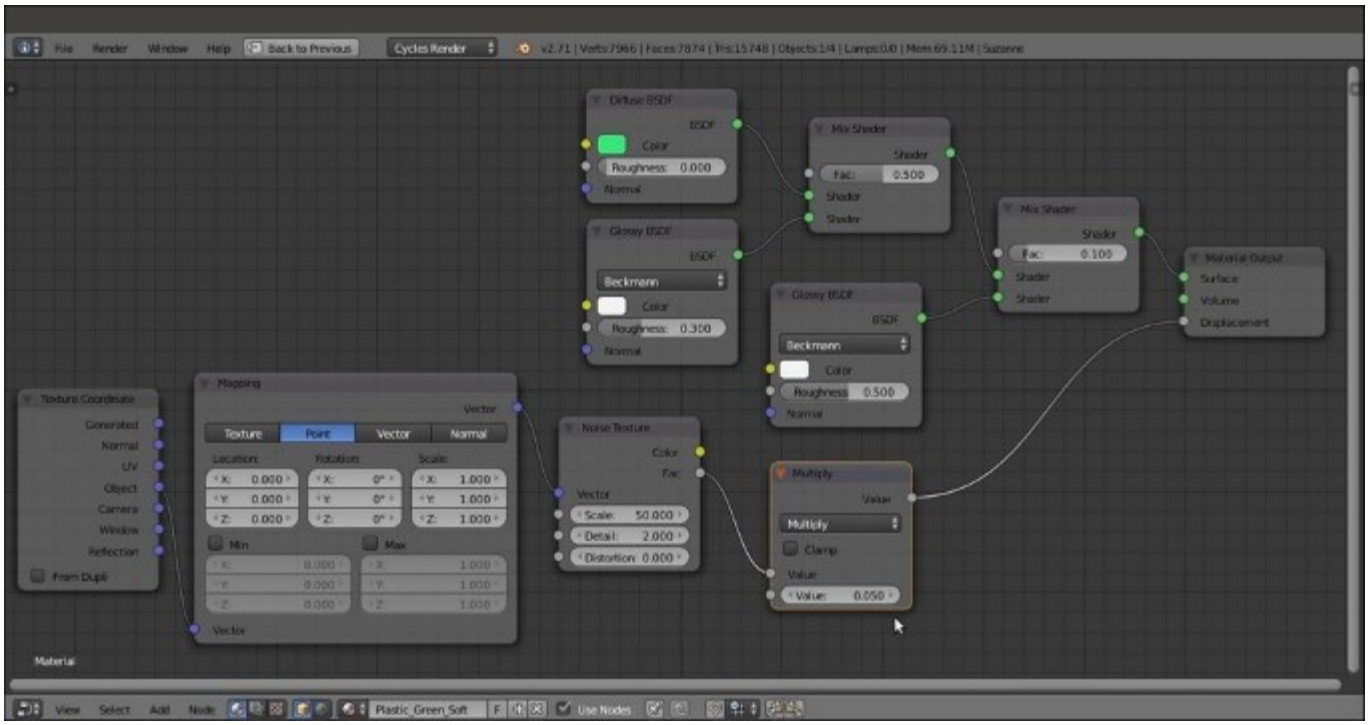
Now we will go straight to creation of the material, so follow these steps:

1. Select **Suzanne** and click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Plastic_Green_Soft`.
2. Set the **Viewport Shading** mode of the **Camera** view to **Rendered** by moving the mouse into the 3D view and pressing *Shift* + *Z*.
3. In the **Material** window under the **Properties** panel, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, select a **Glossy BSDF** node.
4. Change the **Diffuse BSDF** color to bright green (change the values of **R** to `0.040`, **G** to `0.800`, and **B** to `0.190`) and the **Glossy BSDF** shader's **Roughness** value to `0.300`.

5. Press *Shift* + *D* to duplicate the **Mix Shader** node, and paste it between the first **Mix Shader** node and the **Material Output** node. Set the **Fac** value to `0.100`.
6. Duplicate the **Glossy BSDF** node and connect its output to the second input socket of the second **Mix Shader** node. Set its **Roughness** value to `0.500`, as shown in the following screenshot:

A screenshot of the entire Blender interface with the basic shader nodes in the Node Editor window at the top

7. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**).
8. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and the output of this node to the input of the **Noise Texture** node.
9. Set the **Noise Texture** node's **Scale** value to `50.000`. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Connect the **Noise Texture** node's **Fac** output to the first **Value** input of the **Math** node. Set the **Math** node's **Operation** to **Multiply** and second **Value** to `0.050`. Connect its **Value** output to the **Displacement** input socket of the **Material Output** node, as shown in the following screenshot:



The very simple bump effect added to the shader nodes by connecting the output of the Noise Texture node to the Displacement input socket of the Material Output node

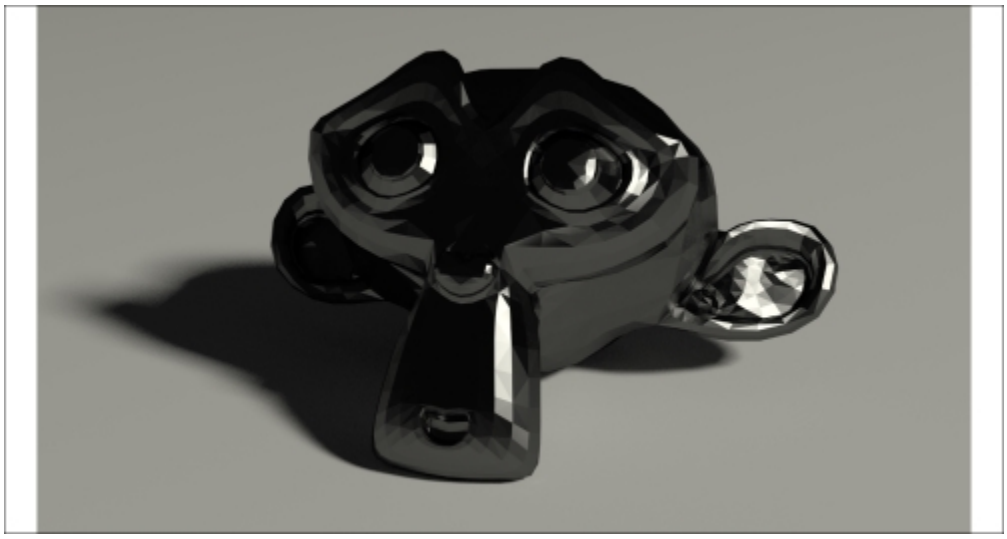
10. Save the file as `Plastic_soft.blend`.

How it works...

This is one of the simplest materials you can build in Cycles. It consists of a colored **Diffuse BSDF** component mixed at 50 percent with a white **Glossy BSDF** shader and another low **Glossy BSDF** shader to make the specular effect more diffused. A tiny **Noise Texture** node, connected directly to the **Displacement** input of the **Material Output** node, adds a slightly dotted bump effect to the whole material, as if it is some kind of industrial plastic used for toys.

Creating a Bakelite material

Bakelite is a very common type of plastic and can be found in a lot of different colors and patterns. In this recipe, we will create the black type (which was once really common), as shown in this screenshot:



The black Bakelite material as it appears in the final rendering

Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file again:

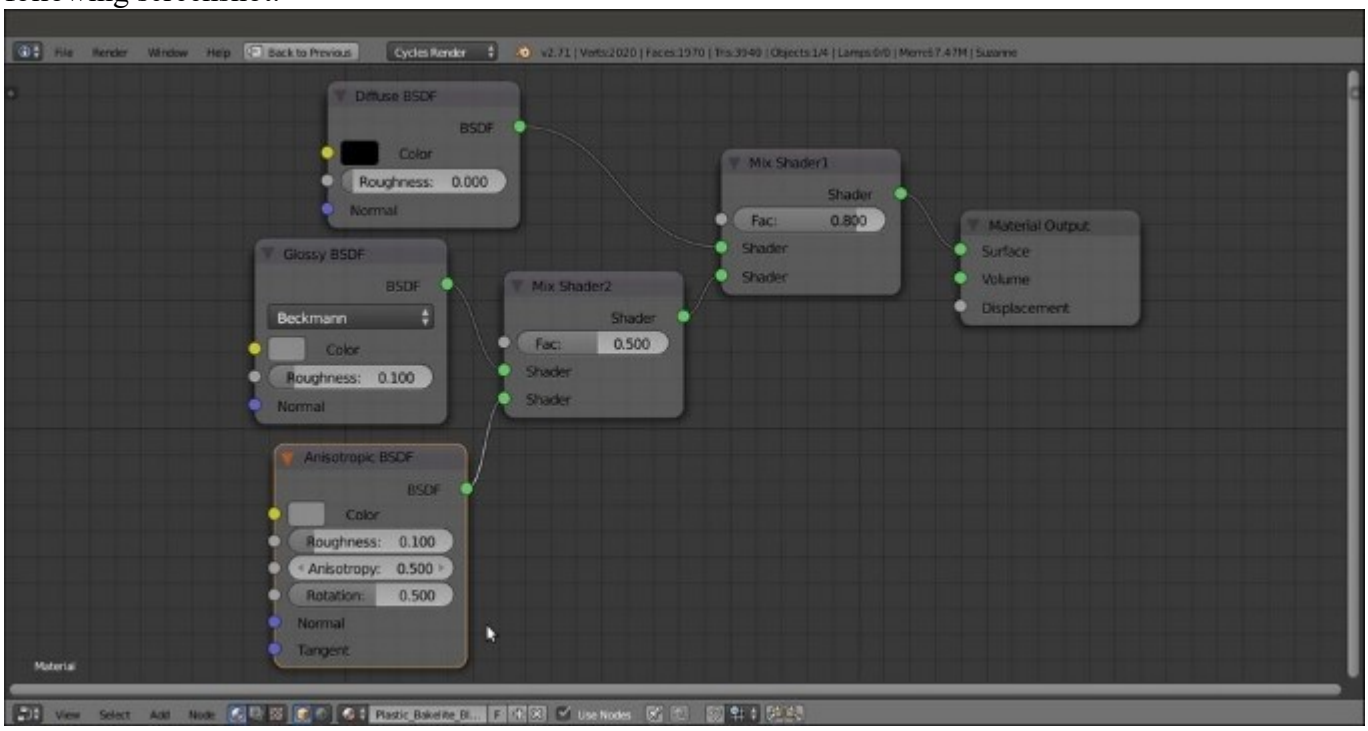
1. With the mouse arrow in the **Camera** view, press the *T* key. Select the **Suzanne** mesh. Go to the **Tools** tab under the **Tool Shelf** panel on the left. Select **Flat** under **Shading**. Press *T* again to close the **Tool Shelf** panel.
2. Go to the **Object modifiers** window in the **Properties** panel. Expand the **Subdivision Surface** modifier panel and set the levels both for **View** and **Render** to `1`.

How to do it...

Now we are going to create the material by performing the following steps:

1. Go to the **Material** window and click on **New** (or do this as usual, in the **Node Editor** toolbar). Rename the material `Plastic_Bakelite_Black`.
2. Set the **Viewport Shading** mode of the **Camera** view to **Rendered**.
3. Switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, select a **Glossy BSDF** node.
4. Change the **Diffuse BSDF** color to pure black and the **Glossy BSDF** shader color to light gray (**RGB** to `0.253`). Set the **Roughness** value of the **Glossy BSDF** shader to `0.100` and the **Fac** value of the **Mix Shader** node to `0.800`.

5. Press *Shift* + *D* to duplicate the **Mix Shader** node, and paste it between the **Glossy BSDF** shader and the first **Mix Shader** node.
6. With the mouse arrow in the **Node Editor** window, press *N*. Select the first **Mix Shader** node, and in the **Label** slot in the **Active Node** panel on the right, write `Mix Shader1`. Select the second **Mix Shader** node, and in the **Label** slot, write `Mix Shader2`.
7. Add an **Anisotropic BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Anisotropic BSDF**) and connect its output to the second input socket of the **Mix Shader2** node.
8. Set the **Mix Shader2** node's **Fac** value to `0.500`. Set the **Anisotropic BSDF** node's color to light gray, and set the same color for the **Glossy BSDF** shader (that is, **RGB** to `0.253`). Set the **Glossy BSDF** shader's **Roughness** value to `0.100` and **Rotation** to `0.500` as shown in the following screenshot:



The simple shader network for the basic Bakelite material

9. Save the file as `Plastic_Bakelite.blend`.

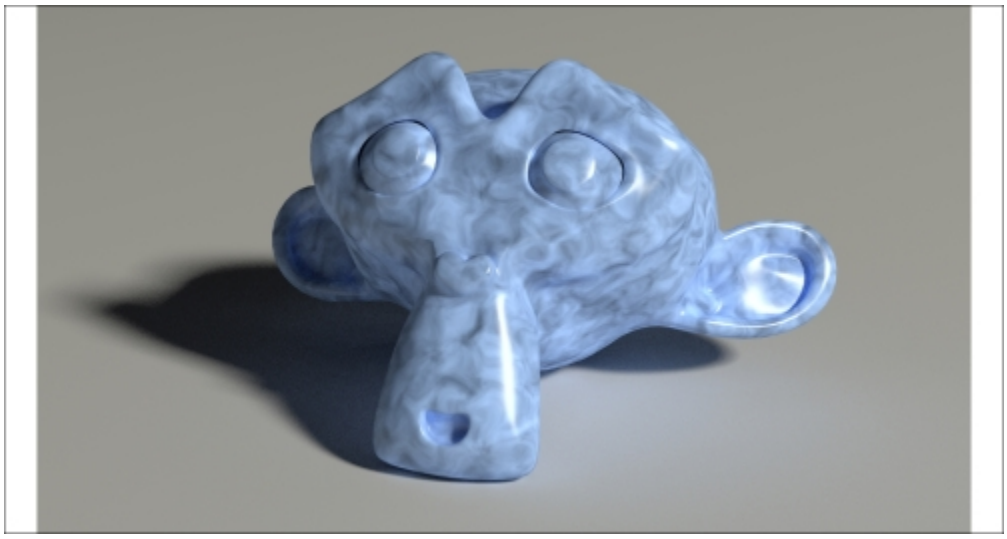
How it works...

Basically, we made the same kind of material as the green plastic material, but this time, we enhanced the reflectivity (mirror) by lowering the **Roughness** value. We also added an **Anisotropic BSDF** specularity effect with the same roughness and color as those for the **Glossy BSDF** shader. The **Rotation** value of the **Anisotropic BSDF** shader sets the flow of the highlights on the mesh. The direction of the specularity rotates as this value increases from `0.000` to `1.000`.

Anisotropy is a method of enhancing image quality of textures on surfaces that are far away and steeply angled with respect to the point of view. An anisotropic surface will change in appearance as it rotates about its geometric normal.

There's more...

Starting from the black material, let's now try to make a differently processed Bakelite material, as shown in the following screenshot:



A different type of Bakelite

First, we'll make a node group of the Bakelite material by performing the following steps:

1. Click on the material name and rename it `Plastic_Bakelite2`. Then save the file as `Plastic_Bakelite2.blend`.
2. Select the **Diffuse BSDF**, **Glossy BSDF**, **Anisotropic BSDF**, and two **Mix Shader** nodes and press *Ctrl* + *G* to make a group.
3. Click and drag the **Diffuse BSDF** node's **Color** socket into the empty socket of the **Group Input** node. Drag the **Fac** socket of the **Mix Shader2** node, and in the **Interface** subpanel of the **Properties** panel of the **Node Editor** window, rename it `Aniso`. This will drive the influence of the anisotropic shader on the glossy shader. Click and drag the **Fac** socket of the **Mix Shader1** node to the empty socket of **Group Input** node. Rename it `Spec`. This will drive the amount of final specularity of the shader. Here is a screenshot of the creation of the Bakelite node group for your reference:

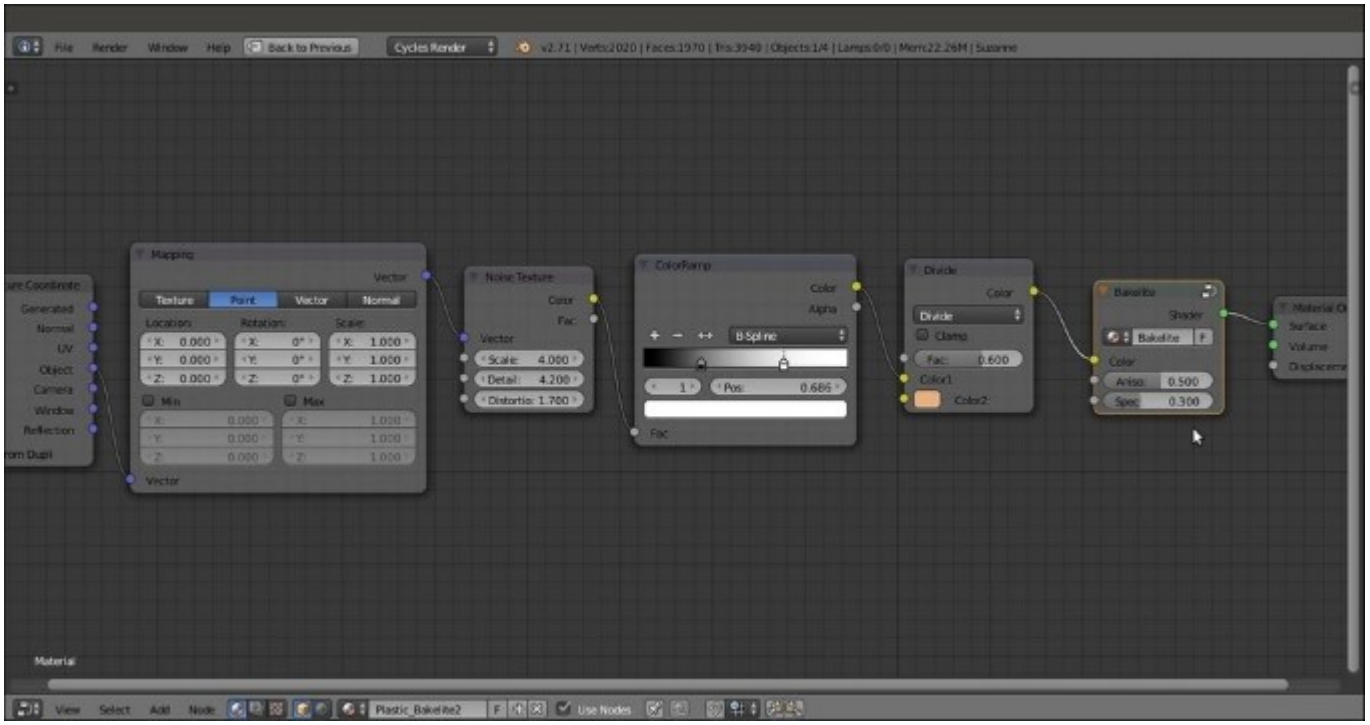
The creation of the Bakelite node group

4. Press *Tab* to close the group. Rename it `Bakelite`.

Now we'll add the nodes needed to create the differently colored material that we decided at the beginning of this section:

1. In the **Node Editor** window, add the following nodes in linear sequence from left to right: a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**).
2. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and the output of this node to the input of the **Noise Texture** node. Connect the **Color** output of the **Noise Texture** node to the **ColorRamp** input socket, and the **Color** output of the **ColorRamp** node to the **Color1** input socket of the **MixRGB** node. Connect the **Color** output of the **MixRGB** node to the **Color** input of the **Bakelite** node group.
3. Set the **Noise Texture** node's **Scale** to `4.000`, **Detail** to `4.200`, and **Distortion** to `1.700`.
4. Set the **ColorRamp** node's **Interpolation** to **B-Spline**. Move the black color marker to position `0.277` and the white color marker to `0.686`.
5. Set **MixRGB** node's **Blend Type** to **Divide** (but remember to experiment with the other types as well) and the **Fac** value to `0.600`. Change the **Color2** values of **R** to `0.799`, **G** to `0.442`, and **B** to `0.220`.

6. In the **Bakelite** node group interface, set the **Spec** value to `0.300`, as shown in the following screenshot:



Adding texture details to the Bakelite node group

7. You can also smooth the **Suzanne** mesh in the **Tool Shelf** panel (press *T*) and increase the **Subdivision** levels of the **Subdivision Surface** modifier to `2`.
8. Save the file.

Creating an expanded polystyrene material

In this recipe, we will create a classic white expanded polystyrene material, as shown in this screenshot:

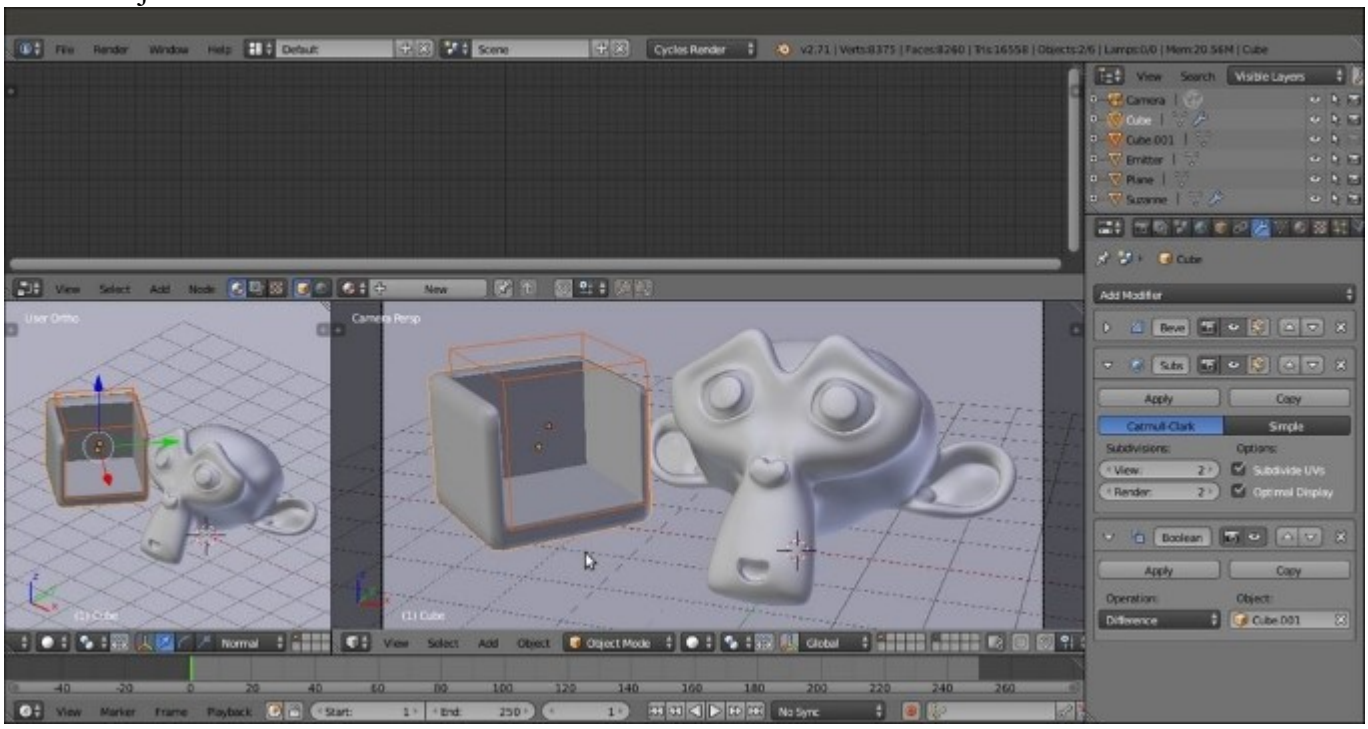
The white expanded polystyrene material as it appears in the final rendering

Getting ready...

First, let's prepare the scene:

1. Start Blender and load the `9931OS_Suzanne_start.blend` file. Add a Cube primitive to the scene and place it leaning on the Plane, close to Suzanne. Move it upwards by 1 Blender unit.
2. With the mouse arrow in the **Camera** view, press *Shift* + *F* to enter **Walk Mode** (in this mode, you can press the *W* key to go forward, press *S* to go back, move the mouse to decide the direction, and click or press *Enter* to confirm). Adjust the **Camera** position so as to center the two objects in the frame.
3. Select the **Cube** object and go to the **Object modifiers** window. Assign a **Boolean** modifier.
4. Press *Shift* + *D* to duplicate the Cube, and move it a bit upward (press *G*, then press *Z*, enter *.4*, and press *Enter*). Reselect the first **Cube**, and in the **Object** field of the **Boolean** modifier panel, select the second Cube (**Cube.001**). Set **Operation** to **Difference**. Go to **Edit Mode** and scale all the vertices a bit larger on the *x* and *y* axes (press *S*, then press *Shift* + *Z*, enter *1.200*, and press *Enter*).
5. Exit **Edit Mode** and reselect **Cube.001**. Move it a bit on the *x* axis (press *G*, then press *X*, enter *.4*, and press *Enter*).
6. Go to the **Object** window and set **Maximum Draw Type** to **Wire**. Then go to the **Ray Visibility** subpanel (usually at the bottom) and uncheck all the items. This way, **Cube.001** becomes visible in the 3D view, but is not yet rendered in the preview.

7. Just to be sure that the second cube is not visible (the previous step should be enough for the final rendering), go to **Outliner** and click on the camera icon to the right of the **Cube.001** item.
8. Select the first **Cube** and assign a **Bevel** modifier. Set the **Width** value to `0.0200`. Move it higher in the stack of modifiers and place it before the **Boolean** modifier.
9. Assign a **Subdivision Surface** modifier and set both the **Subdivisions** levels to `2`. Check the **Optimal Display** item and move it higher in the stack. Place it before the **Boolean** modifier but after the **Bevel** modifier.
10. Press *T* to call the **Tool Shelf** panel. Set the **Cube** shading to **Smooth**.
11. Press *Shift*, select both **Cube** and the **Cube.001** objects, and rotate them on *z* axis towards the **Camera** (press *R*, then press *Z*, enter *-40*, and then press *Enter*).
12. Press *T* to close the **Tool Shelf** panel. The following screenshot shows the process of building the box object:



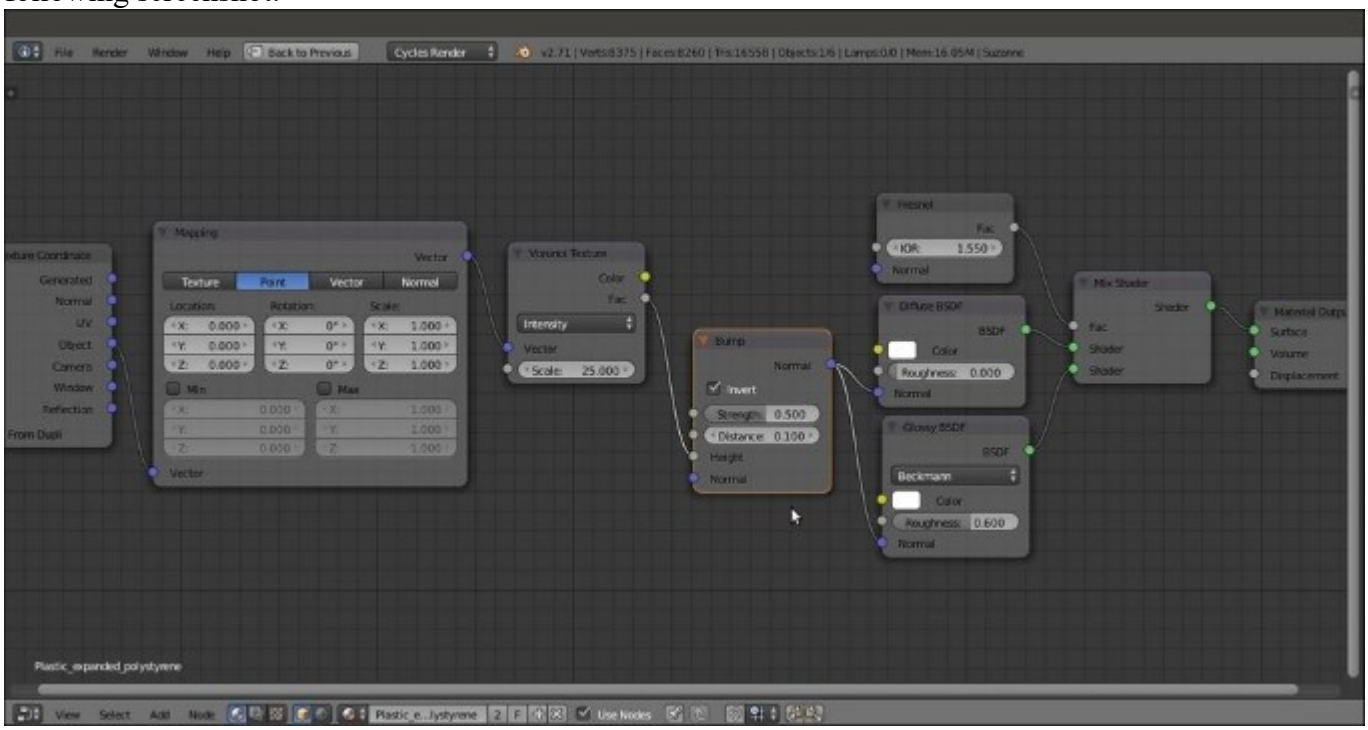
Building the box object by a Boolean modifier

13. Select the **Plane** object, and in the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. Then, in the **Shader** slots, select a **Diffuse BSDF** node and a **Glossy BSDF** shader node. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect the **Facing** output to the **Fac** input socket of the **Mix Shader** node. Set the color of the **Diffuse BSDF** node as follows: **R** to `0.530`, **G** to `0.800`, and **B** to `0.800`.

How to do it...

Now we are going to create the material by performing the following steps:

1. Select **Suzanne** and click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Plastic_expanded_polystyrene`.
2. Switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, select a **Glossy BSDF** node.
3. Set the **Diffuse BSDF** shader color and the **Glossy** shader color to pure white. Set the **Roughness** value of the **Glossy BSDF** shader to `0.600`. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**). Connect its output to the **Fac** input socket of the **Mix Shader** node. Set the **IOR** value to `1.550`.
4. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**), a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**).
5. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and the output of this node to the **Vector** input of the **Voronoi Texture** node.
6. Set the **Voronoi Texture** node's **Scale** value to `25.000`. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Fac** output of the **Voronoi Texture** node to the **Height** input socket of the **Bump** node, and the output of this node to the **Normal** input sockets of the **Diffuse BSDF** and **Glossy BSDF** shader nodes.
7. Check the **Invert** item on the **Bump** node and set the **Strength** value to `0.500`, as shown in the following screenshot:



The white expanded polystyrene material network

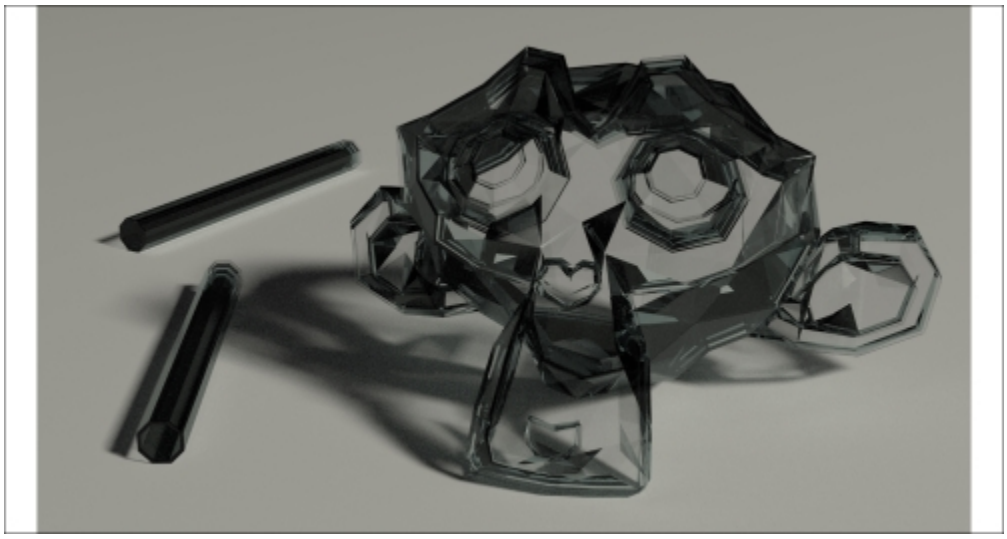
8. Press *Shift* and select the **Cube** object and **Suzanne**. Then press *Ctrl* + *L*, and in the **Make Links** pop-up menu, select the **Material** item to assign the material of the active object to the other object.
9. Save the file as `Plastic_expanded_polystyrene.blend`.

How it works...

You have probably noticed that this recipe is simply a variation of the generic plastic shader. We changed the color to white, and instead of **Noise Texture**, we used a **Voronoi Texture** node with a different scale to add the typical polystyrene pattern. Then, by increasing the **Roughness** value of the **Glossy BSDF** shader, we made the specularity more diffused.

Creating a clear (glassy) polystyrene material

In this recipe, we will create a glassy polystyrene material (which you find on the body of ballpoint pens), as shown in the following screenshot:



The glassy polystyrene material as it appears in the final rendering

Getting ready...

First, we need the usual preparation:

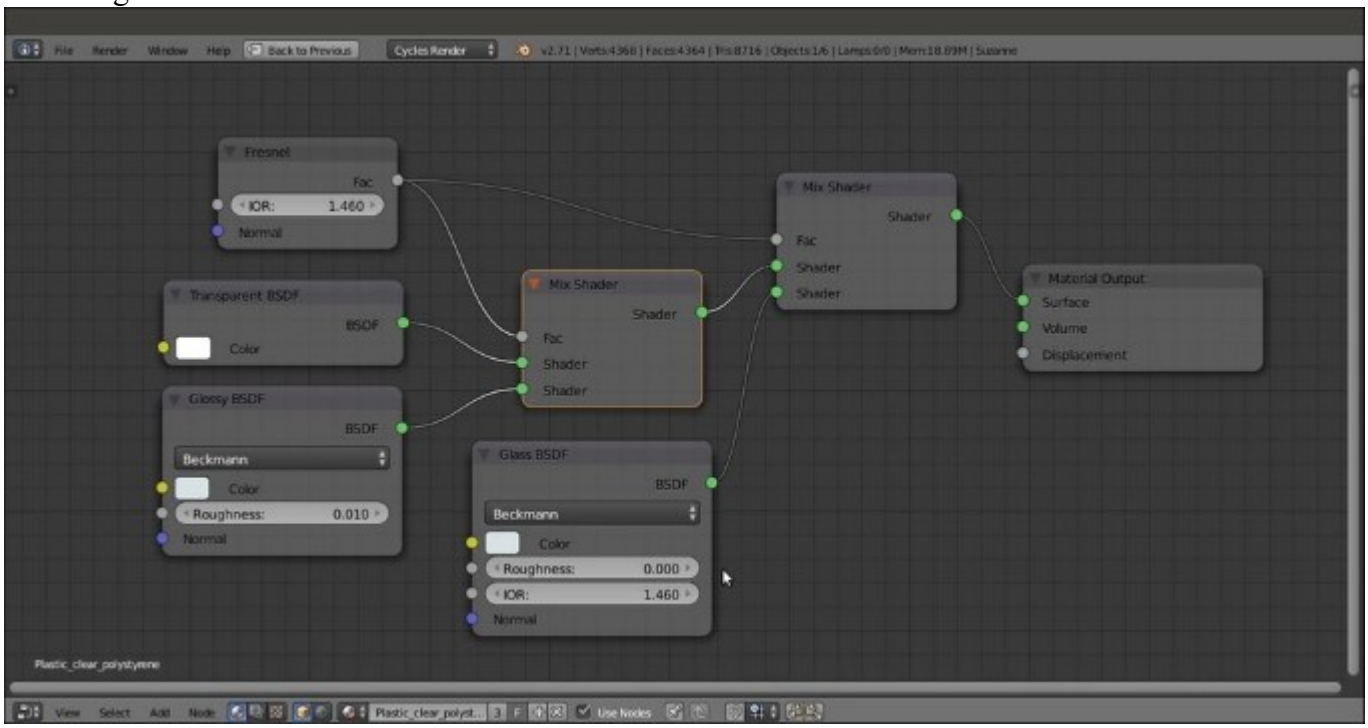
1. Start Blender and load the `9931OS_Suzanne_start.blend` file.
2. Select the **Suzanne** mesh and press *T*. In the **Tool Shelf** panel on the left side, select **Flat** under **Shading**. Press *T* again to close the **Tool Shelf** panel.
3. Go to the **Object modifiers** window in the **Properties** panel and delete the **Subdivision Surface** modifier. Add a **Solidify** modifier and set the **Thickness** value to `0.0350`. Add a **Bevel** modifier and set the **Width** value to `0.0050`. Uncheck the **Clamp Overlap** item.

How to do it...

Now we are going to create the material by performing the following steps:

1. Go to the **Material** window and click on **New** (or as usual, go to the **Node Editor** toolbar). Rename the material `Plastic_clear_polystyrene`.
2. Set the **Viewport Shading** mode of the **Camera** view to **Rendered**.
3. Switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Mix Shader** node again. In the second **Shader** slot, select a **Glass BSDF** node. Set its **IOR** value to `1.460`. Change the values of **R** to `0.688`, **G** to `0.758`, and **B** to `0.758`.

4. Go to the second **Mix Shader** node, and in its first **Shader** slot, select a **Transparent BSDF**. In the second **Shader** slot, select a **Glossy BSDF** node. Change the **Glossy BSDF** node color values for **R** to `0.688`, **G** to `0.758`, and **B** to `0.758`. Change the **Roughness** value to `0.010`.
5. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input sockets of both the **Mix Shader** nodes. Set the **IOR** value to `1.460`, as shown in the following screenshot:



The completed network for the glassy polystyrene material

6. Save the file as `Plastic_clear_polystyrene.blend`.

Creating a rubber material

In this recipe, we will create a generic rubber shader, as shown in this screenshot:

The rubber material as it appears in the final rendering

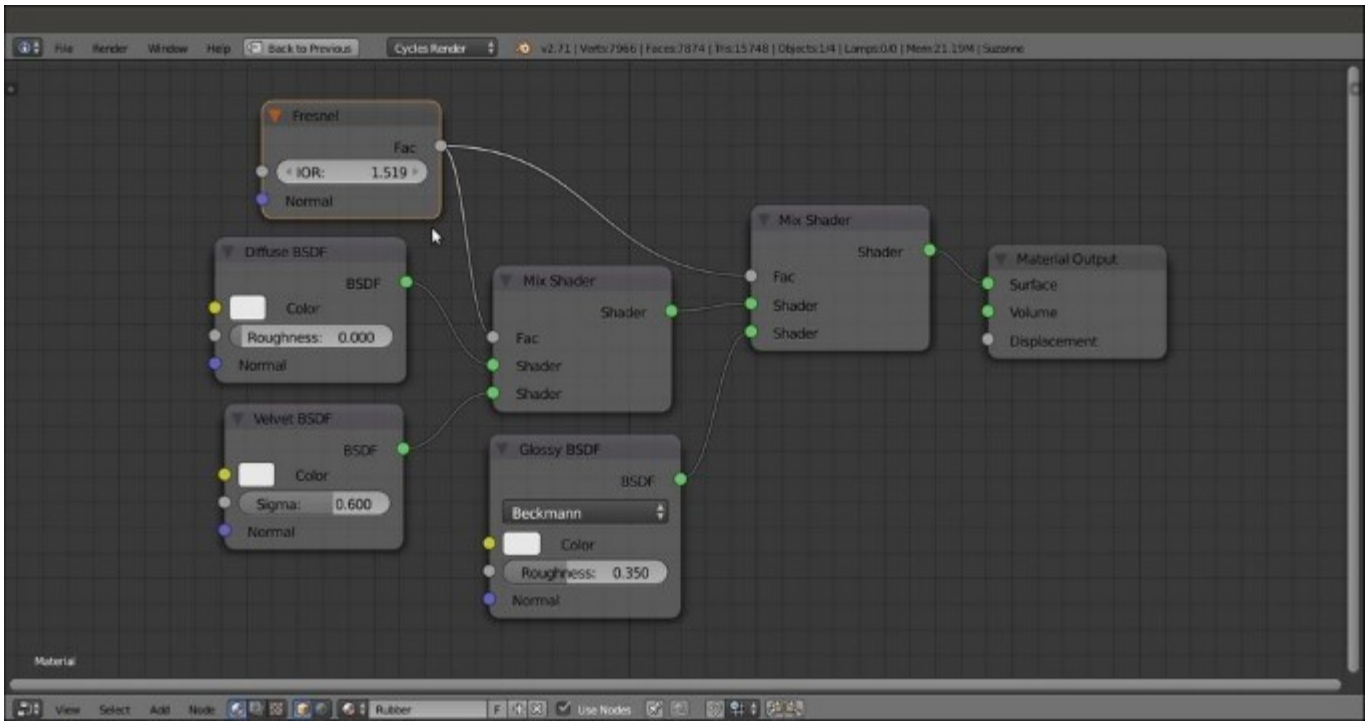
Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file.

How to do it...

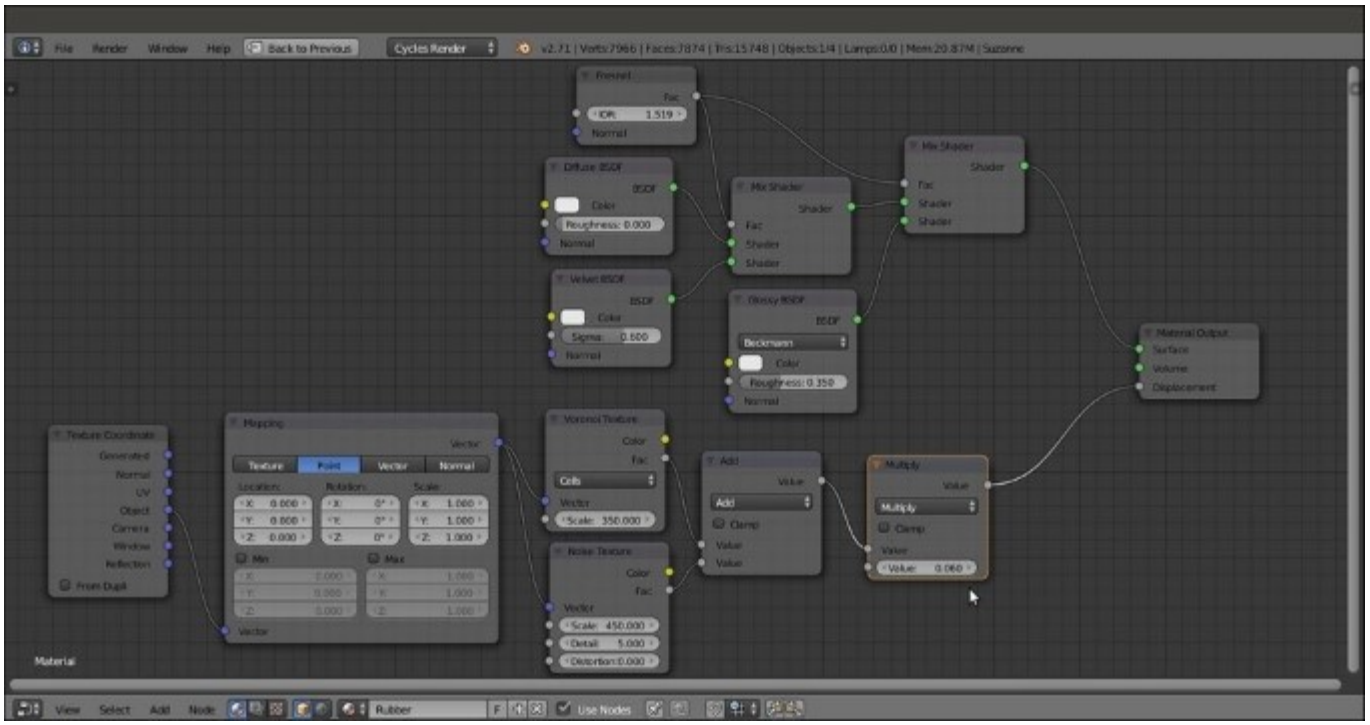
Now we are going to create the material by performing the following steps:

1. Click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Rubber`.
2. With the mouse arrow in the **Camera** view, press *Shift* + *Z* to set it to **Rendered** mode.
3. Switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the second **Shader** slot, select a **Glossy BSDF** node. In the first **Shader** slot, select a new **Mix Shader** node. Set the **Glossy BSDF** node's **Roughness** value to `0.350`.
4. Go to the second **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** node. In the second **Shader** slot, select a **Velvet BSDF** node. Set the **Velvet BSDF** shader node's **Sigma** value to `0.600`.
5. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input of both the **Mix Shader** nodes. Set the **IOR** value to `1.519`, as shown in the following screenshot:



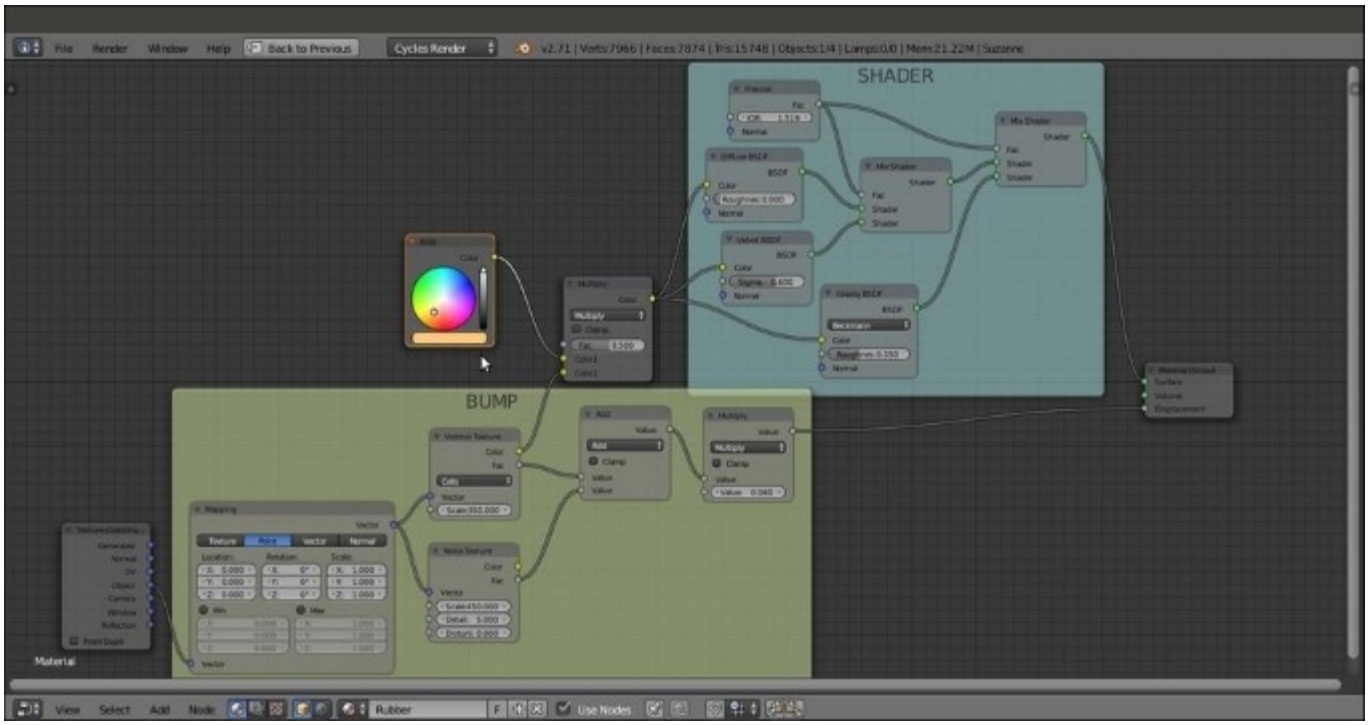
The basic shader network

6. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), a **Voronoi Texture** node, and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**, do the same to add **Noise Texture** node).
7. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and the latter's output to the **Vector** input sockets of the two texture nodes.
8. Set the **Voronoi Texture** node's **Coloring** to **Cells** and the **Scale** value to `350.000`. Set the **Noise Texture** node's **Scale** value to `450.000` and **Detail** to `5.000`.
9. Add two **Math** nodes (press *Shift* + *A* and navigate to **Converter** | **Math**). Set the **Operation** of the second node to **Multiply**. Connect the **Fac** output of the **Voronoi Texture** node to the first **Value** input socket of the **Add-Math** node. Connect the **Fac** output of the **Noise Texture** node to the second **Value** input socket of the **Add-Math** node.
10. Connect the **Add-Math** node output to the first **Value** input socket of the **Multiply-Math** node. Set second **Value** to `0.060` and connect the output to the **Displacement** input socket of the **Material Output** node, as shown in the following screenshot:



The slight bump effect added to the network

11. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and move it close to the **Voronoi Texture** node. Set the **Blend Type** to **Multiply**. Connect the **Voronoi Texture** node's **Color** output to the **Color2** input socket of the **Multiply-MixRGB** node. Then connect the **Color** output of this node to the **Color** input sockets of the **Diffuse BSDF**, **Velvet BSDF**, and **Glossy BSDF** shaders.
12. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and connect it to the **Color1** input socket of the **Multiply-MixRGB** node, as shown in this screenshot:



The overall view of the network

13. Save the file as `Rubber.blend`.

How it works...

We built the shader in steps 1 to 5. We added a slight bump effect in steps 6 to 10. In the last two steps, we just added the **RGB** node, a control used to set the color of the material.

Creating an antique bronze material with procedurals

In this recipe, we will create a bronze shader that looks similar to a ruined, corroded, and antique statue, as shown in the following screenshot:

The antique bronze material as it appears in the final rendering when assigned to the poor Suzanne mesh!

Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file. Then perform the following steps:

1. With **Suzanne** selected, click on the **Object Mode** button in the **Camera** view toolbar. Choose **Vertex Paint**.
2. Click on the **Paint** item in the toolbar and select **Dirty Vertex Colors**, the first option at the top. Then press *T*, and in the **Option** panel at the bottom of the **Tool Shelf** panel, set **Blur Strength** to `0.01` and **Dirt Angle** to `90`. Check the **Dirt Only** item, as shown in this screenshot:

The Dirty Vertex Colors setting and the effect on the Suzanne mesh

3. Go to the **Object data** window under the **Properties** panel. Double-click on the **Col** item in the **Vertex Colors** subpanel and rename it `V_col`.
4. Return in **Object Mode** and press *T* to close the **Tool Shelf** tabs.
5. Save the file as `9931OS_Suzanne_vcol.blend`. We will use this file for other recipes.

How to do it...

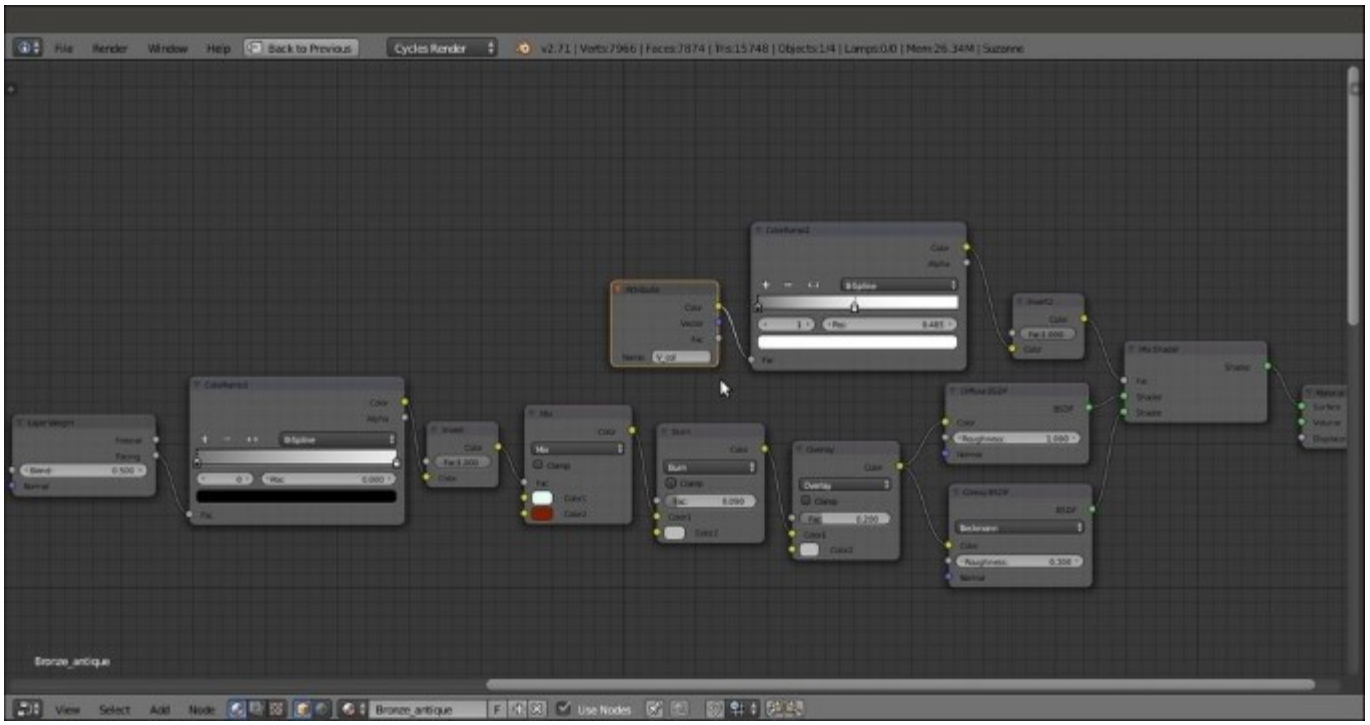
Now we are going to create the material by performing the following steps:

1. First, save the file as `Bronze_antique.blend`.
2. Click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Bronze_antique`.
3. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, select a **Glossy BSDF** node. Set the **Diffuse BSDF** shader node's **Roughness** value to `1.000` and the **Glossy BSDF** node's **Roughness** value to `0.300`.
4. Now add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). In the **Properties** panel of the **Node Editor** window (press *N* to make it appear), label the **ColorRamp** node as `ColorRamp1`. Set its **Interpolation** mode to `B-Spline`.
5. Connect the **Facing** output of the **Layer Weight** node to the **Fac** input of the **ColorRamp1** node, and its **Color** output to the **Fac** input socket of the **MixRGB** node.

6. Set the **Color1** value of the **MixRGB** node as **R** to `0.771`, **G** to `1.000`, and **B** to `0.848`. Set the **Color2** values as **R** to `0.222`, **G** to `0.013`, and **B** to `0.000`.
7. Add an **Invert** node (press *Shift* + *A* and navigate to **Color** | **Invert**). Paste it between the **ColorRamp1** and the **MixRGB** nodes'.
8. Press *Shift* + *D* to duplicate the **MixRGB** node, and set **Blend Type** to **Burn**. Set the **Fac** value to `0.090`. Connect the **Mix-MixRGB** node's **Color** output to the **Color1** input socket of the **Burn-MixRGB** node.
9. Press *Shift* + *D* to duplicate the **MixRGB** node again. Set the **Blend Type** to **Overlay** and the **Fac** value to `0.200`. Connect its **Color** output to both the **Color** input sockets of the **Diffuse BSDF** and **Glossy BSDF** shaders. Now connect the **Color** output of the **Burn-MixRGB** node to the **Color1** input socket of the **Overlay-MixRGB** node, as shown in the following screenshot:

The shader part of the material

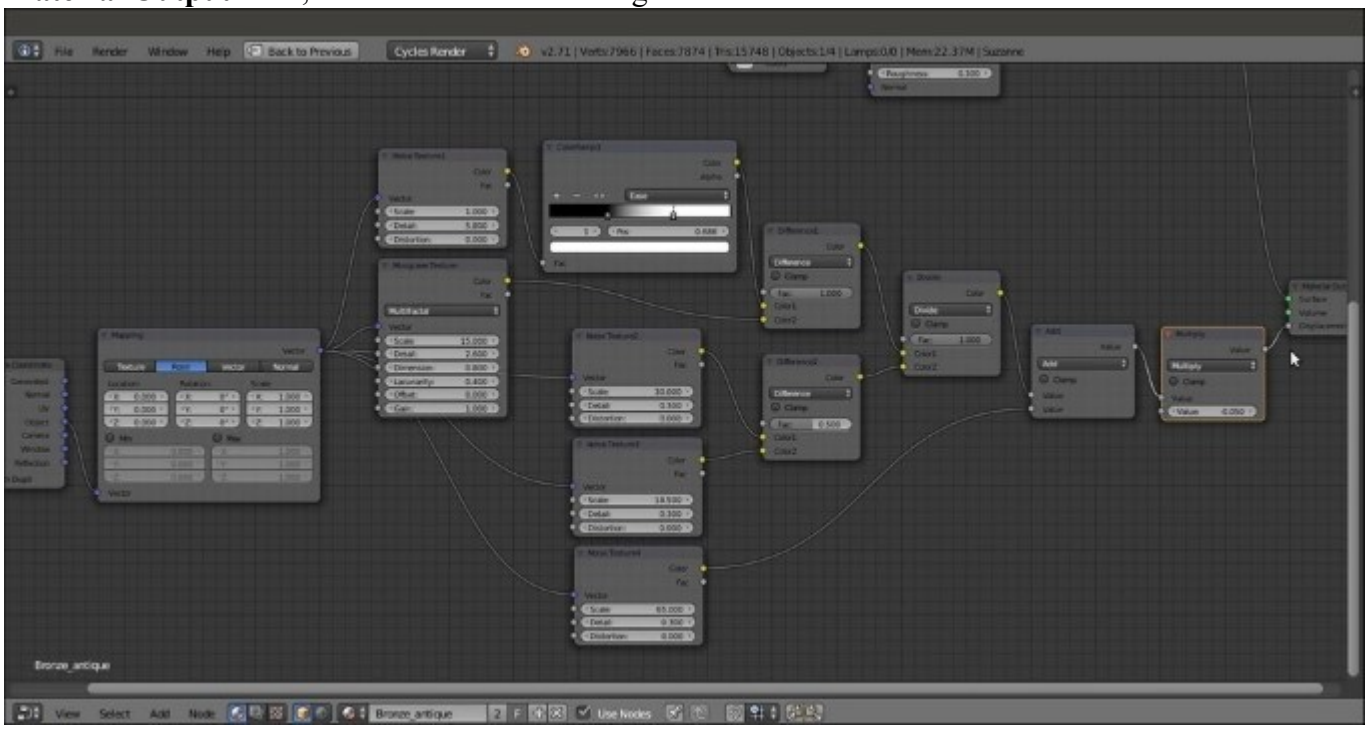
10. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**). Select and press *Shift* + *D* to duplicate the **ColorRamp** and the **Invert** nodes. Label them as `ColorRamp2` and `Invert2` , respectively, and move them close to the **Attribute** node.
11. Write the name of the Vertex Color layer, `V_col`, in the **Name** slot of the **Attribute** node. Then connect its **Color** output to the **Fac** input of the **ColorRamp2** node. Move the white color stop to position `0.485`.
12. Connect the **Color** output of the **ColorRamp** node to the Color input of the **Invert2** node, then connect the **Color** output of the **Invert2** node to the **Fac** input socket of the **Mix Shader** node, as shown in the following screenshot:



The shader modulated by the Dirty Vertex Colors output

13. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), four **Noise Texture** nodes (press *Shift* + *A*, navigate to **Textures** | **Noise Texture**, and press *Shift* + *D* to duplicate them), and a **Musgrave Texture** node (press *Shift* + *A* and navigate to **Textures** | **Musgrave Texture**).
14. Label the four **Noise Texture** nodes as `Noise Texture1`, `Noise Texture2`, `Noise Texture3`, and `Noise Texture4`.
15. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node. Connect the **Vector** output of this node to the **Vector** input sockets of all the five texture nodes.
16. For the **Noise Texture1** node, set **Scale** to `1.000` and **Detail** to `5.800`. For the **Noise Texture2** node, set **Scale** to `30.000` and **Detail** to `0.300`. For the **Noise Texture3** node, set **Scale** to `18.500` and **Detail** to `0.300`. Finally, for the **Noise Texture4** node, set **Scale** to `65.000` and **Detail** to `0.300`.
17. For the **Musgrave Texture** node, set **Type** to **Multifractal**, **Scale** to `15.000`, **Detail** to `2.600`, **Dimension** to `0.800`, and **Lacunarity** to `0.400`.
18. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set **Blend Type** to **Difference**, and label it as `Difference1`. Press *Shift* + *D* to duplicate it. Label the duplicate as `Difference2`.
19. Connect the **Color** output of the **Noise Texture1** node to the **Color1** input socket of the **Difference1** node, and set the **Fac** output of this node to `1.000`. Connect the **Color** output of the **Musgrave Texture** node to the **Color2** input socket of the **Difference1** node.

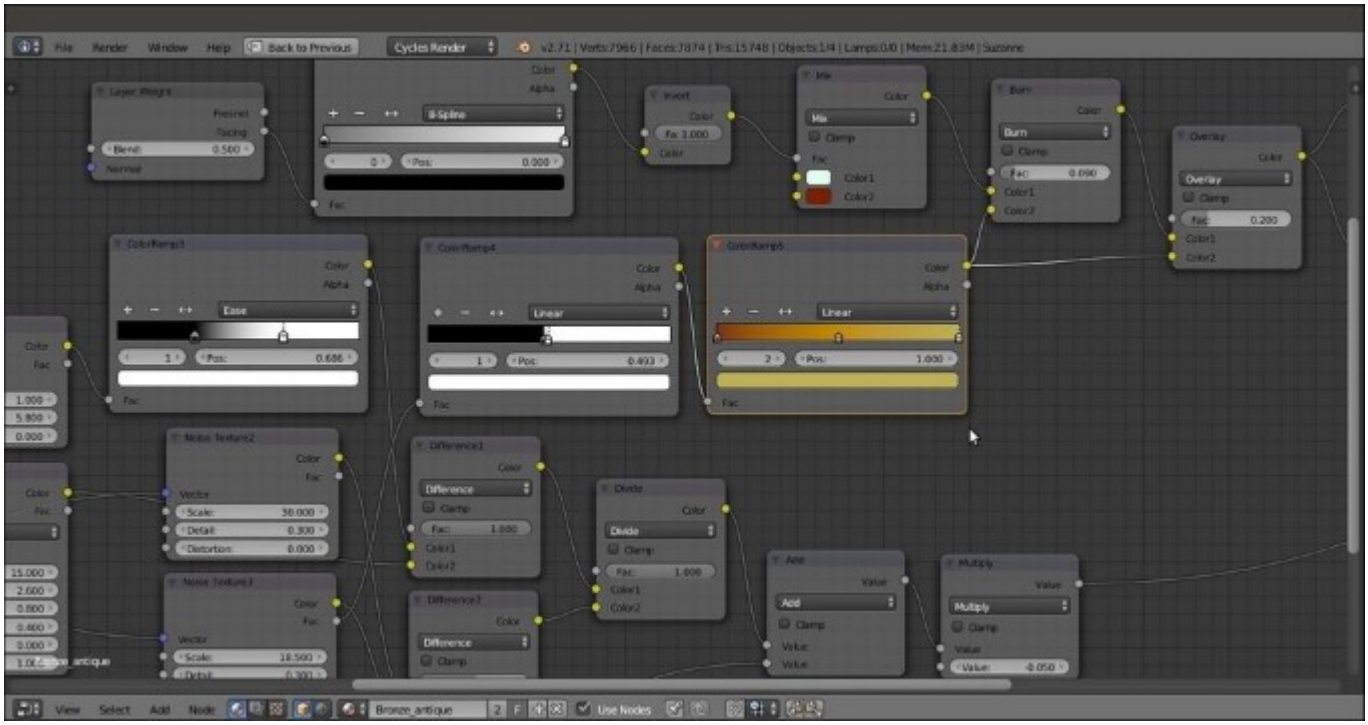
20. Connect the **Color** output of the **Noise Texture2** node to the **Color1** input socket of the **Difference2** node, and the **Color** output of the **Noise Texture3** node to the **Color2** input socket of the **Difference1** node.
21. Press *Shift* + *D* to duplicate the **Difference1** node, and set the **Blend Type** to **Divide**. Connect the **Color** output of the **Difference1** node to the **Color1** input socket of the **Divide** node, and the **Color** output of the **Difference2** node to the **Color2** input socket.
22. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Connect the output of the **Divide** node to the first **Value** input socket, and the **Color** output of the **Noise Texture4** node to the second **Value** socket.
23. Press *Shift* + *D* to duplicate the **Math** node, and set the **Operation** to **Multiply**. Connect the **Value** output of the **Add-Math** node to the first input socket of the **Multiply-Math** node. Set second **Value** to `-0.050`.
24. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), paste it between the **Noise Texture1** node and the **Difference1** node, and label it as `ColorRamp3`. Set **Interpolation** to **Ease**. Move the black color stop to the `0.318` position and the white color stop to the `0.686` position.
25. Connect the output of the **Multiply-Math** node to the **Displacement** input socket of the **Material Output** node, as shown in the following screenshot:



The bump pattern's nodes

26. Add two new **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Label them as `ColorRamp4` and `ColorRamp5`.
27. Connect the **Fac** output of the **Noise Texture3** node to the **ColorRamp4** node. Set the color black stop to the `0.479` position and the white stop to the `0.493` position.

28. Connect the **Color** output of the **ColorRamp4** node to the **ColorRamp5** node's input socket, and the **Color** output of this node to the **Color2** input sockets of the **Burn** and **Overlay** nodes.
29. Click on the + icon on the **ColorRamp5** node to add a new color stop in the middle of the color slider. Set the black stop color (index `0`) for **R** to `0.216`, **G** to `0.027`, and **B** to `0.007`; the middle stop color (index `1`) for **R** to `0.539`, **G** to `0.261`, and **B** to `0.000`; and the white color stop color (index `2`) for **R** to `0.515`, **G** to `0.433`, and **B** to `0.088`, as shown in the following screenshot:

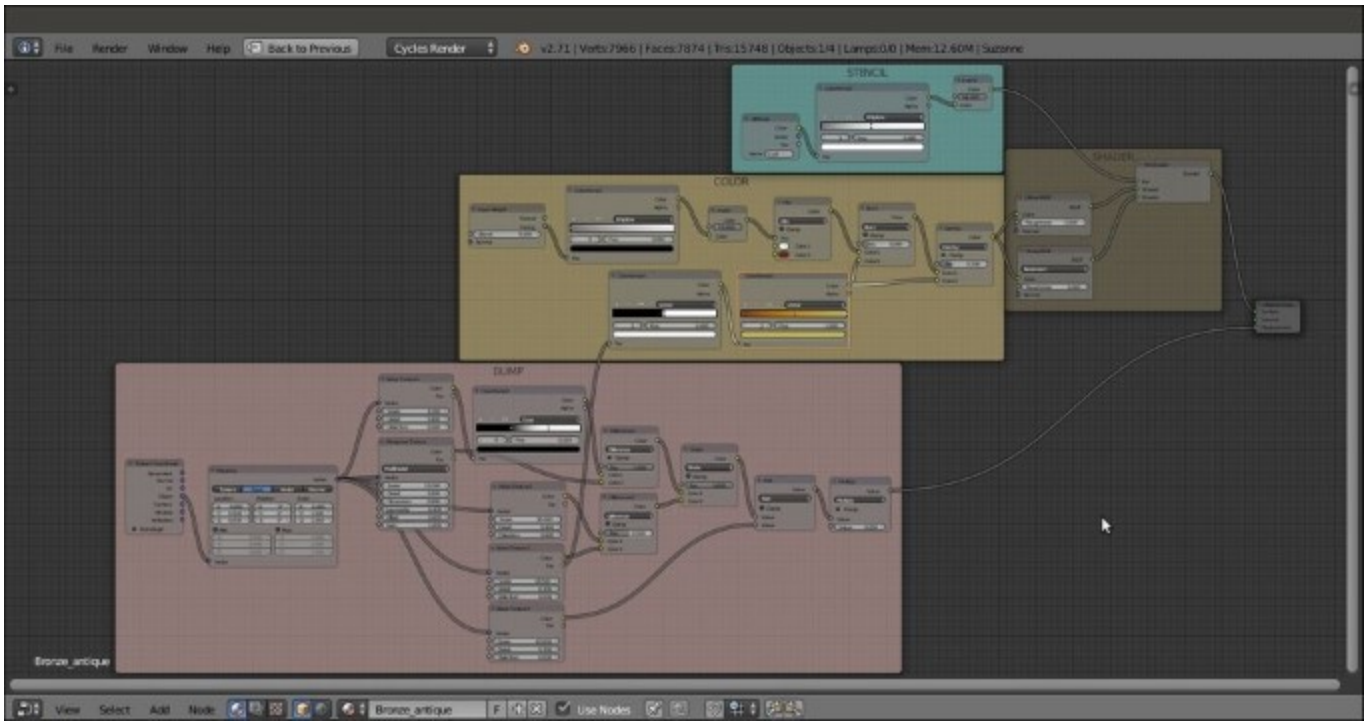


Adding color details to the ground of the bump textures

How it works...

We use the Vertex Color layer set in the *Getting ready* section as a stencil map to distribute both the colored **Diffuse BSDF** and the **Glossy BSDF** shaders, driven by the **Facing** option of the **Layer Weight** input node.

Most of the bump effect is created by the **Noise Texture** and **Musgrave Texture** nodes, which are mixed and clamped in several ways by the **ColorRamp** nodes. Here is a screenshot of the entire material network:



The overall view of the antique bronze material network

As usual, the last **Math** node, which is set to **Multiply**, establishes the strength of the bump.

Creating a multipurpose metal node group

All the metal materials you can see in the following screenshot (pewter, gold, silver, chromium, and aluminum) were obtained from a single shader node group linked and applied to each Suzanne with different interface settings.

To take a look at the scene, open the `9931OS_04_metals.blend` file. In this recipe, we will build the generic **Metal** node group shader. You can find it in the `9931OS_04_metal_group.blend` file, as shown in the following screenshot:

Some examples of different metal materials created by the same node group

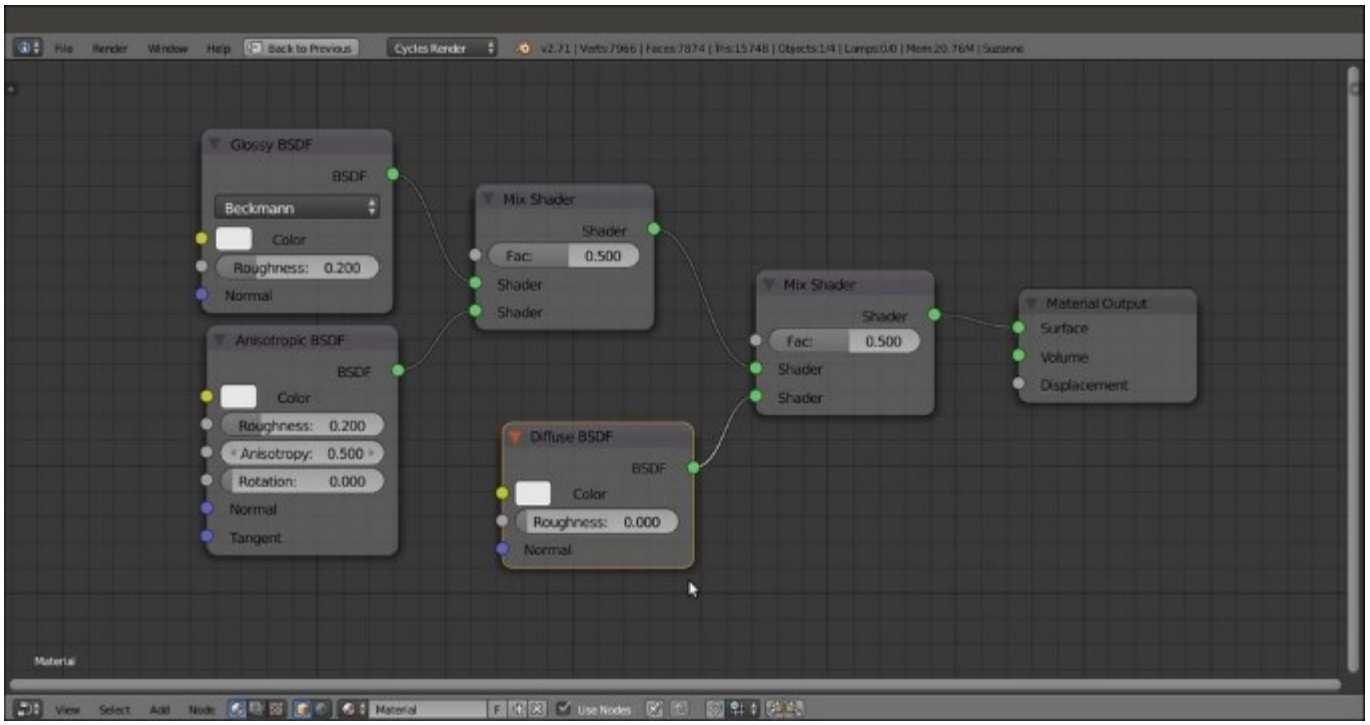
Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file.

How to do it...

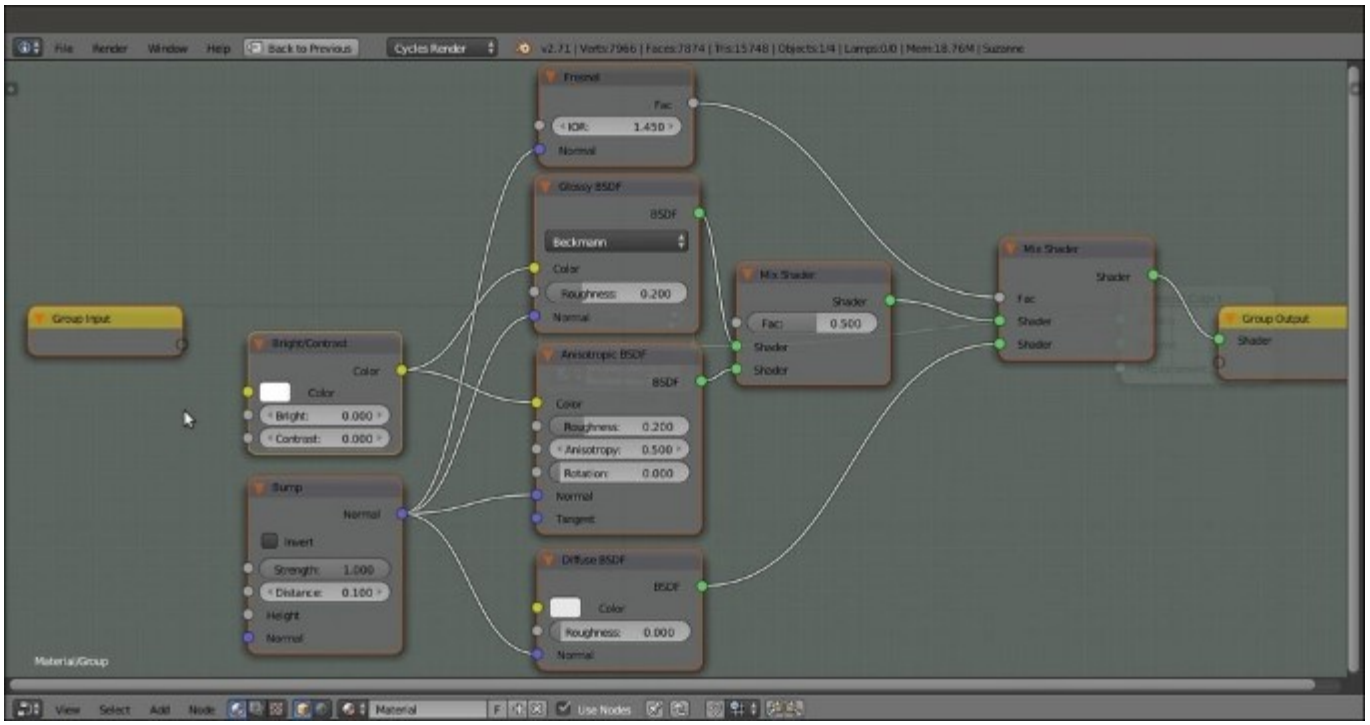
Now we are going to create the node group by performing the following steps:

1. Click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Glossy BSDF** shader. In the second **Shader** slot, select an **Anisotropic BSDF** node.
3. Press *Shift* + *D* to duplicate the **Mix Shader** node, and paste it just after the first **Mix Shader** node. Add a **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**) and connect it to the second **Shader** input of the second **Mix Shader** node, as shown in the following screenshot:



The basic metal shader network

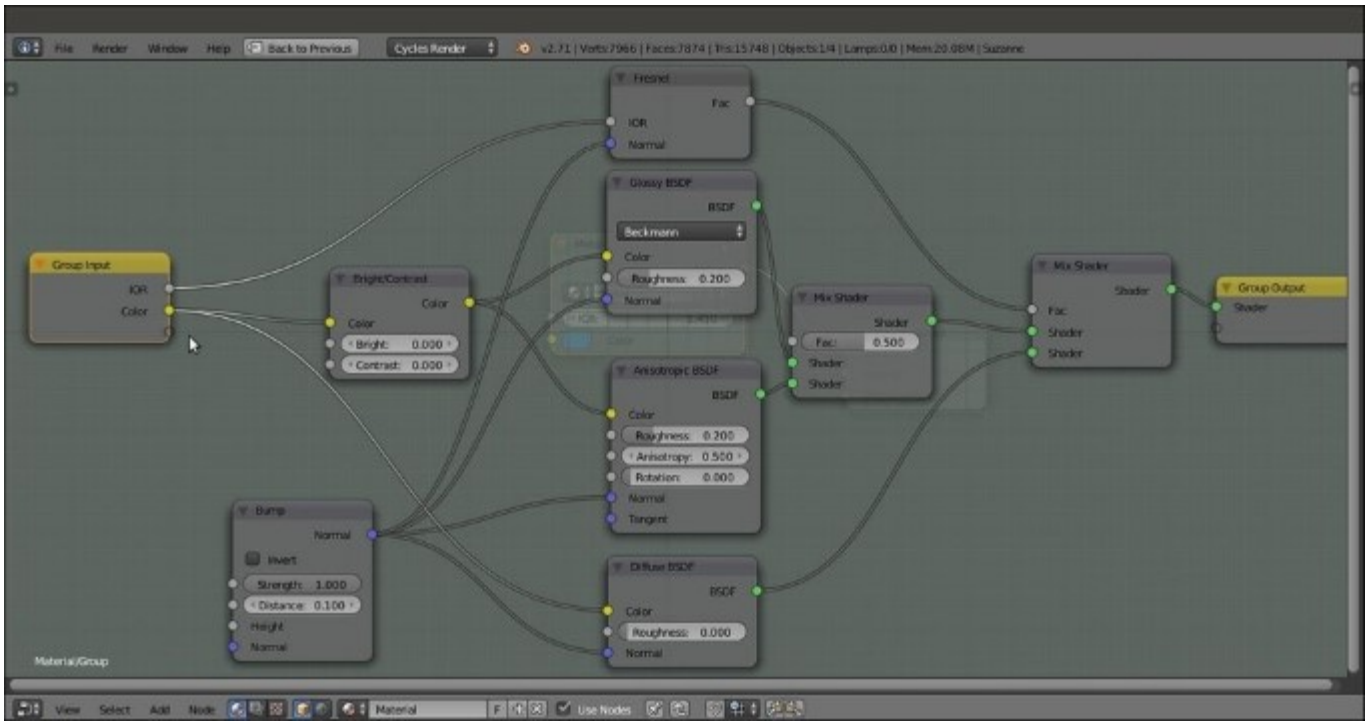
4. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input of the second **Mix Shader** node.
5. Add a **Bright Contrast** node (press *Shift* + *A* and navigate to **Color** | **Bright Contrast**). Connect its **Color** output to the **Color** input sockets of the **Glossy BSDF** and **Anisotropic BSDF** shader nodes.
6. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect its **Normal** output to the **Normal** input sockets of the **Fresnel**, **Diffuse BSDF**, **Glossy BSDF**, and **Anisotropic BSDF** shader nodes.
7. Select all the nodes except the **Material Output** node, and press *Ctrl* + *G* to create a group, as shown in the following screenshot:



The nodes inside the node group

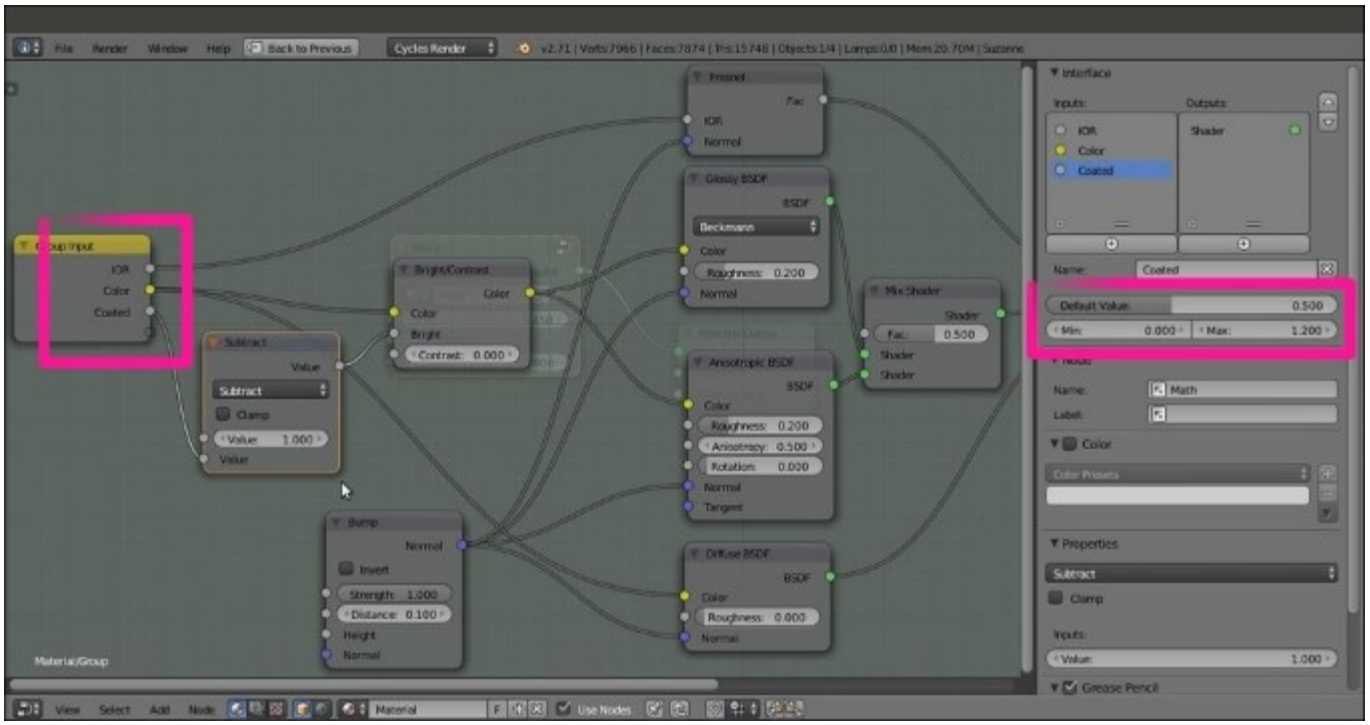
Now we must expose all the values necessary to tweak the node group for the different types of metal:

1. Click and drag the **IOR** input socket of the **Fresnel** node into the empty socket of the **Group Input** node.
2. Repeat step 1 with the **Color** input socket of the **Diffuse BSDF** shader node. Then drag the **Color** socket of the **Bright/Contrast** node and connect it to the same **Color** socket on the **Group Input** node, as shown in this screenshot:



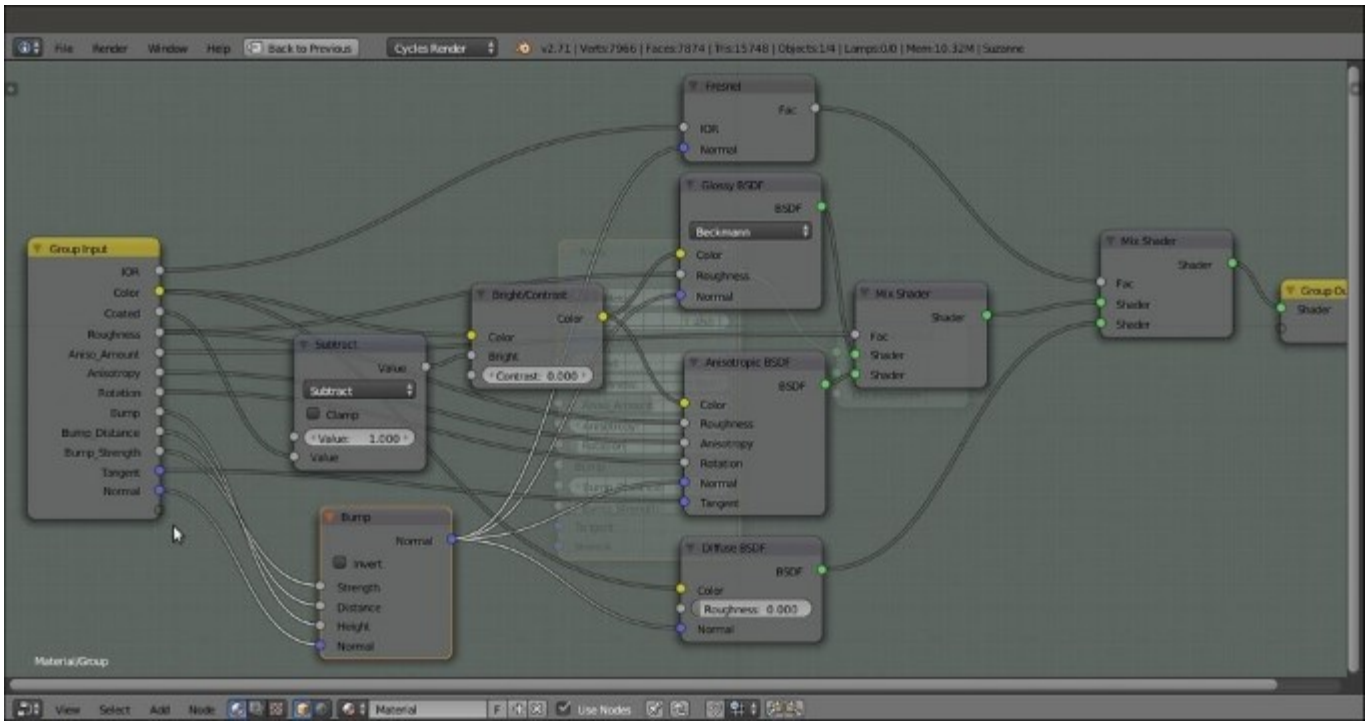
Creating the exposed sockets

3. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set **Operation** to **Subtract** and first **Value** to `1.000`. Drag its second **Value** input socket to the **Group Input** node, label the new socket as `Coated`, and then connect the **Value** output to the **Bright** input socket of the **Bright/Contrast** node. In the **Interface** subpanel under the **Properties** panel, set the **Max** value for the **Coated** socket to `1.200`, as shown in the following screenshot:



The setting of the Min and Max values through to the Interface subpanel

4. Drag the **Roughness** input socket of the **Glossy BSDF** shader. Then drag the **Roughness** socket of the **Anisotropic BSDF** shader and connect it to the same socket on the **Group Input** node.
5. Click and drag the **Fac** socket of the first **Mix Shader** node into a new, empty socket. Rename it `Aniso_Amount`. Click and drag the **Anisotropy** socket of the **Anisotropic BSDF** shader node into a new, empty socket. Repeat this step for the **Rotation** input socket.
6. Now click and drag the **Height** socket of the **Bump** node into a new, empty socket. Rename it `Bump`. Repeat this step for the **Distance** and the **Strength** sockets, and rename the sockets `Bump_Distance` and `Bump_Strength`, respectively.
7. Also repeat for the **Normal** socket of the **Bump** node.
8. Finally, click and drag the **Tangent** input socket of the **Anisotropic BSDF** shader.
9. Use the arrows in the top-right corner of the **Interface** subpanel to order the sockets in the **Group Input** node (the same order should be used for the **Group Output** node), as shown in the following screenshot:



The final layout of the completed node group in Edit Mode

10. Exit **Edit Mode** by pressing *Tab*. Rename the group `Metal`. Although this is not "strictly necessary here, you can also click on the **F** icon on the interface to activate the *fake user* for the node group.
11. Save the file as `Metal_group.blend`.

How it works...

The effect of this node group is mainly based on the IOR value (the refractive index of a material is a number that describes how light propagates through that material or gets reflected on its surface). This value can be quite different for each kind of metal. In the node, the exposed **IOR** value drives the amount of blending of the Diffuse component with the Mirror component made by the **Glossy BSDF** and **Anisotropic BSDF** shader nodes combined, but that can also be mutually blended accordingly to the **Aniso_Amount** value.

The **Anisotropy** and **Rotation** values of the **Anisotropic BSDF** shader are exposed as well. and the same for the **Tangent** input if a particular mapping option must be used (for example, a layer of UV coordinates).

Textures must be connected to the **Bump** input socket on the **Bump** node. The **Bump_Strength** socket establishes the amount of bump influence. The **Bump_Distance** socket is a multiplier for the strength of influence. The **Bump** node output is piped to all the **Normal** input of the **Fresnel**, **Diffuse BSDF**, **Glossy BSDF**, and **Anisotropic BSDF** nodes to keep a consistent effect among all the components.

Similarly, both the **Glossy BSDF** and **Anisotropic BSDF** nodes' **Roughness** values are driven by a single-interface input.

Finally, let's discuss the color of the metal. The color that arrives at the **Diffuse BSDF** shader by passing through the **Bright/Contrast** node gets modified by a **Coated** value larger than `0.000`. The result is a different input for the mirror component. The **Subtract-Math** node simply inverts the effect of the numeric input of the **Coated** socket.

Besides the links provided at the end of the previous chapter, for a list of IORs, you can take a look at these links:

- http://refractiveindex.info/
- http://www.robinwood.com/Catalog/Technical/Gen3DTuts/Gen3DPages/RefractionIndexList.html
- http://forums.cgsociety.org/archive/index.php/t-513458.html

Note

Note that for some materials (especially metals), different lists report different IOR values.

Creating a rusty metal material with procedurals

In this recipe, we will create a rusty shader that will be mixed with the metal shader by a stencil factor, as shown in the following screenshot:

The rusty metal material as it appears in the final rendering

Getting ready...

Start Blender and load the `9931OS_Suzanne_vcol.blend` file. Then perform the following steps:

1. Go to the **World** window. Click on the dotted little box to the right of the **Color** slot under the **Surface** subpanel. In the pop-up menu, select the **Environment Texture** item.
2. Click on the **Open** button and browse to the `textures` folder to load the `Barce_Rooftop_C_3K.hdr` image.
3. Set the **Strength** value to `0.200`. Then go back to the **Material** window.

How to do it...

Now we are going to create the shader by performing the following steps:

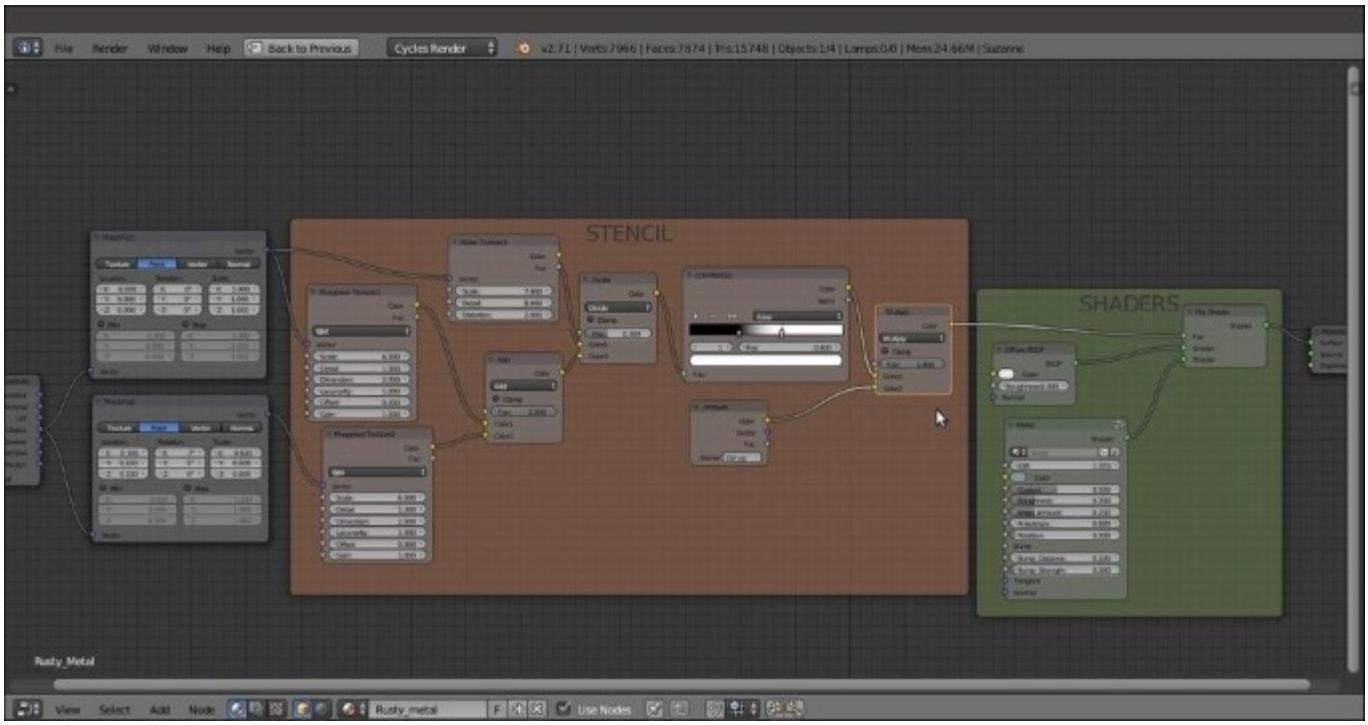
1. Click on the **File** item in the upper main header. Select the **Link** item. If necessary, browse to the folder where you stored all the blend files. Select the `9931OS_04_metal_group.blend` file. From there, click on the **NodeTree** entry and then select the **Metal** item. Click on the **Link/Append from Library** button to link the node group.
2. Click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Rusty_metal`.

3. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, under **Group** in the pop-up menu, load the linked **Metal** node group, as shown in the following screenshot:

The Properties pop-up menu used to select different nodes

4. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select the **Diffuse BSDF** shader, the **Metal** node group, the **Mix Shader** node, and then the **Frame**. Press *Ctrl* + *P* to parent them. In the **Properties** panel of the **Node Editor** window (press the *N* key to make it appear), label the **Frame** as `SHADERS`.
5. In the **Metal** group, set the **IOR** value to `1.370`. Change the Color values for **R** to `0.229`, **G** to `0.307`, and **B** to `0.299`. Set the **Roughness** value to `0.200`, **Aniso_Amount** to `0.200`, and **Anisotropy** to `0.600`.
6. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and two **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of both the **Mapping** nodes. Label them as `Mapping1` and `Mapping2`.
7. Now add a **Musgrave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Musgrave Texture**) and label it as `Musgrave Texture1`. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), label it as `Noise Texture1`, and add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Label this node as `ColorRamp1`.
8. Set the **Musgrave Texture** node's **Scale** value to `6.000` and **Detail** to `1.300`. Press *Shift* + *D* to duplicate it, and label the duplicate as `Musgrave Texture2`. Set the **Noise Texture** node's **Scale** value to `7.800`, **Detail** to `8.000`, and **Distortion** to `2.000`.

9. Connect the **Vector** output of the **Mapping1** node to the **Vector** input sockets of the **Musgrave Texture1** and **Noise Texture1** nodes. Then connect the **Vector** output of the **Mapping2** node to the **Vector** input sockets of the **Musgrave Texture2** node. In the **Mapping2** node, set the **Location** value to `0.100` and **Scale** to `0.600` for the three axes. Then change the **Rotation** value of **X** to `7` and **Y** to `-5`.
10. Connect the **Fac** output of the **Noise Texture** node to the **Fac** input of the **ColorRamp1** node. Set its **Interpolation** to **Ease**. Move the black color stop to the `0.321` position and the white color stop to the `0.600` position.
11. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Fac** value to `1.000` and **Blend Type** to **Add**. Then connect the color output of the **Musgrave Texture1** node to the **Color1** input socket and the **Color** output of the **Musgrave Texture2** node to the **Color2** input socket of the **Add-MixRGB** node.
12. Press *Shift* + *D* to duplicate the **Add-MixRGB** node. Set **Blend Type** to **Divide** and the **Fac** value to `0.309`. Connect the **Fac** output of the **Noise Texture1** node to the **Color1** input socket and the output of the **Add-MixRGB** node to the **Color2** input socket of the **Divide-MixRGB** node.
13. Connect the output of the **Divide-MixRGB** node to the **Fac** input socket of the **ColorRamp1** node. Press *Shift* + *D* to duplicate the **Divide-MixRGB** node, and change the **Blend Type** to **Multiply**. Set the **Fac** value to `1.000` and connect the output of the **ColorRamp1** node to the **Color1** input socket.
14. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**) and connect its **Color** output to the **Color2** input socket of the **Multiply-MixRGB** node. In its **Name** slot, write the name of the Vertex Color layer, (`Col_vp`).
15. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select the **Musgrave Texture** node, the **Noise Texture** node, the **ColorRamp1** node, the three **MixRGB** nodes, the **Attribute** node, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `STENCIL`.
16. Connect the output of the **Multiply-MixRGB** node under the **STENCIL** frame to the **Fac** input socket of the **Mix Shader** node under the **SHADERS** frame, as shown in the following screenshot:

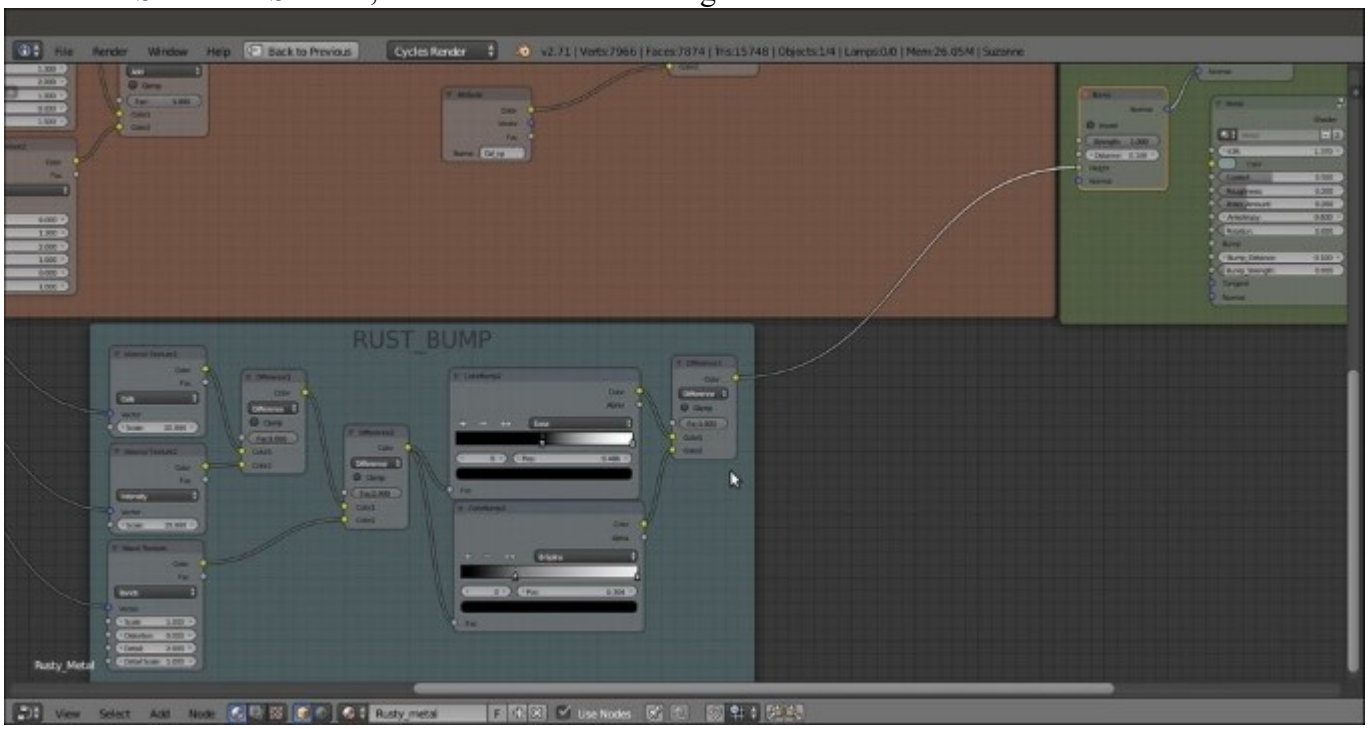


The first two frames of the material, SHADERS and STENCIL

17. Add two **Voronoi Texture** nodes (press *Shift* + *A*, navigate to **Texture** | **Voronoi Texture**, and label them as `Voronoi Texture1` and `Voronoi Texture2`) and a **Wave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**). In the **Voronoi Texture1** node, set the **Coloring** to **Cells** and the **Scale** value to `20.000`. In the **Voronoi Texture2** node, set the **Scale** value to `19.000`. Set the **Wave Texture** node's **Scale** value to `1.000`.
18. Connect the **Vector** output of the **Mapping1** node to the **Vector** input sockets of these three new texture nodes.
19. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Difference**, and label it as `Difference1`. Set the **Fac** value to `1.000`. Then connect the **Voronoi Texture1** node's **Color** output to the **Color1** input socket and the second **Voronoi Texture2** node's **Color** output to the **Color2** input socket.
20. Press *Shift* + *D* to duplicate the **Difference1** node, and label the duplicate as `Difference2`. Connect the **Color** output of the **Difference1** node to the **Color1** input socket of the **Difference2** node. Then connect the **Color** output of the **Wave Texture** node to the **Color2** input socket.
21. Add two **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label them as `ColorRamp2` and `ColorRamp3`, and connect the output of the **Difference2** node to their **Fac** input socket. Set the **ColorRamp2** node's **Interpolation** to **Ease** and move the black color stop to the `0.486` position. Set the **ColorRamp3** node's **Interpolation** to **B-Spline** and move the black color stop to the `0.304` position.
22. Press *Shift* + *D* to duplicate the **Difference2** node, and label the duplicate as `Difference3`. Place it after the **ColorRamp2** node and **ColorRamp3** nodes. Connect the **ColorRamp2** node's

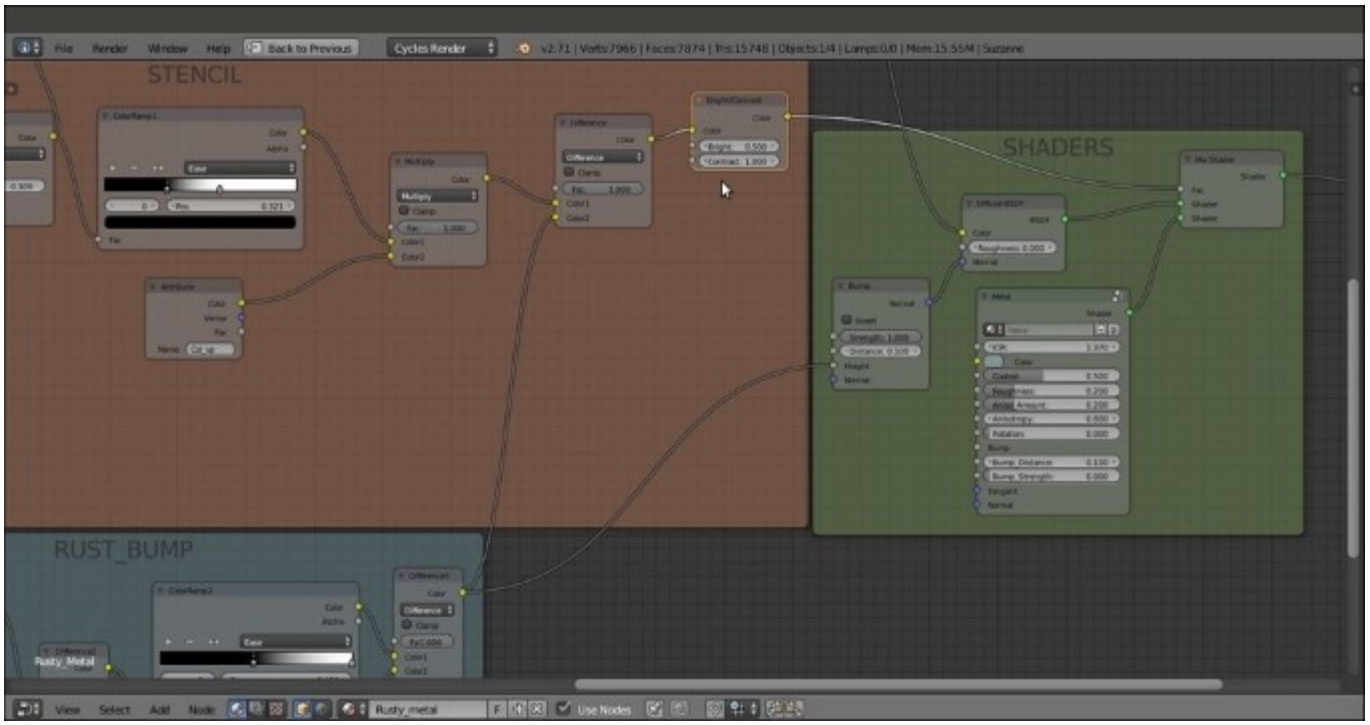
Color output to the **Color1** input socket and the **ColorRamp3** node's **Color** output to the **Color2** input socket of the **Difference3** node.

23. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select these lastly added nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame `RUST_BUMP`.
24. Select the **SHADERS** frame and add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Difference3** node's output to the **Height** input socket of the **Bump** node. Connect the **Normal** output of this node to the **Normal** input socket of the **Diffuse BSDF** node inside the **SHADERS** frame, as shown in the following screenshot:



The frame for the bump of the rust

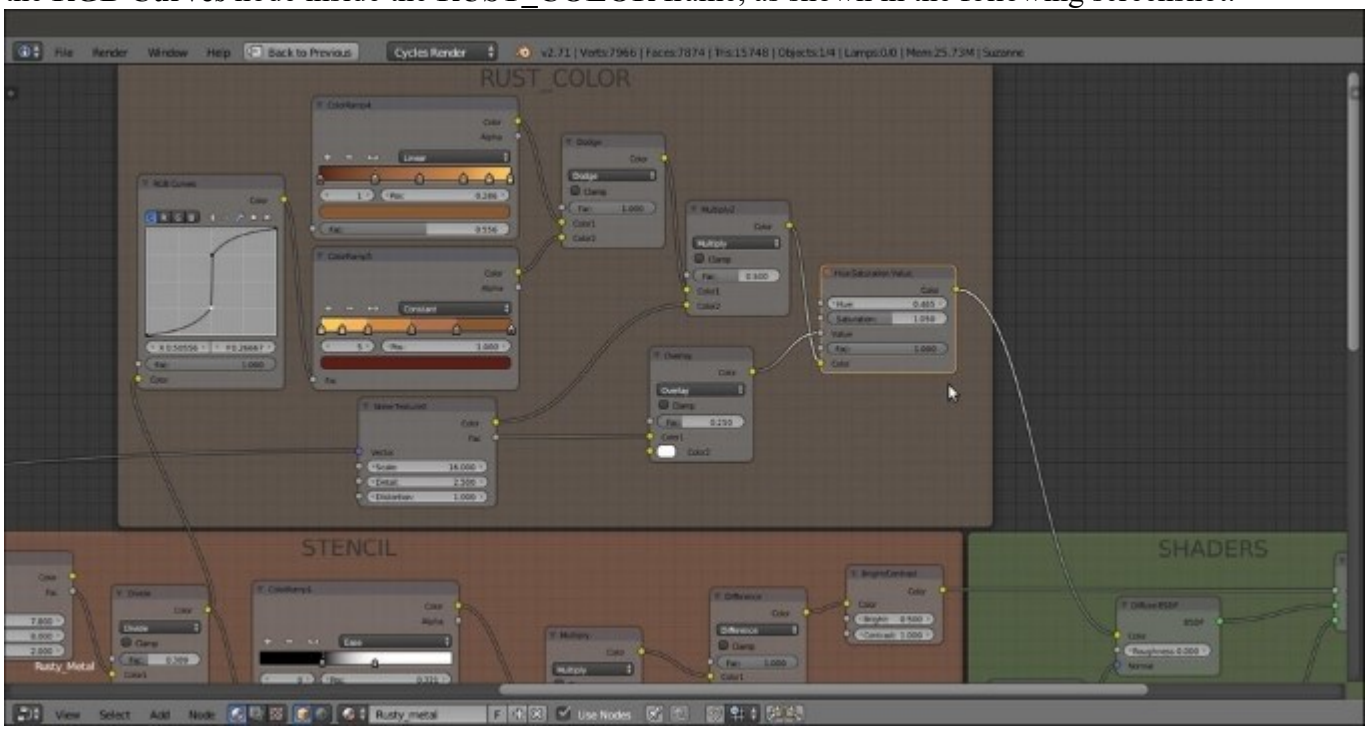
25. Select the **STENCIL** frame and add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and a **Bright/Contrast** node (press *Shift* + *A* and navigate to **Color** | **Bright Contrast**). Paste the **MixRGB** node right after the **Multiply-MixRGB** node. Set **Blend Type** to **Difference** and then connect the output of the **Difference3** node inside the **RUST_BUMP** frame to the **Color2** input socket. Paste the **Bright/Contrast** node right after this last **Difference** node. Set the **Bright** value to `0.500` and the **Contrast** value to `1.000`, as shown in the following screenshot:



The RUST_BUMP output added to the stencil to separate the metal surface from the rusty surface

26. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**), two **ColorRamp** nodes (press *Shift* + *A*, navigate to **Converter** | **ColorRamp**, and label them as `ColorRamp4` and `ColorRamp5`), a **Noise Texture** node (press *Shift* + *A*, navigate to **Texture** | **Noise Texture**, and label it as `Noise Texture2`), and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**).
27. In the **RGB Curves** node's interface, click on the diagonal line to add a control point. In the **X** and **Y** slots at the bottom, set the values to `0.50000` and `0.26000`, respectively. Click again to add a new control point, and set **X** to `0.51000` and **Y** to `0.75000`.
28. Connect the **Color** output of the **RGB Curves** node to the **Fac** input of the **ColorRamp5** node.
29. Go to the **ColorRamp4** node. Select the black color stop and change the color values of **R** to `0.991`, **G** to `0.591`, and **B** to `0.084`. Select the white color stop and change the color values of **R** to `0.105`, **G** to `0.013`, and **B** to `0.010`.
30. Click four times on the **+** icon on the **ColorRamp4** node interface to add four new color stops. Select **4** as the index number and move it to the `0.889` position. Change the color values of **R** to `0.930`, **G** to `0.456`, and **B** to `0.105`. For index **3**, set **Pos** to `0.754`, **R** to `0.624`, **G** to `0.250`, and **B** to `0.053`. For index **2**, set **Pos** to `0.521`, **R** to `0.418`, **G** to `0.159`, and **B** to `0.068`. Finally, for index **1**, set **Pos** to `0.286`, **R** to `0.246`, **G** to `0.098`, and **B** to `0.034`.
31. With the mouse arrow inside the colorband on the **ColorRamp4** node, press *Ctrl* + *C* to copy it. Move the mouse arrow to the colorband of the **ColorRamp5** node. Press *Ctrl* + *V* to paste the colors and the stops. Set the **ColorRamp5** node's **Interpolation** to **Constant**.

32. Connect the **Color** output of both the **ColorRamp4** and **ColorRamp5** nodes to the **Color1** and **Color2** input sockets of the **MixRGB** node, respectively. Set the **MixRGB** node's **Blend Type** to **Dodge** and the **Fac** value to `1.000`.
33. Press *Shift* + *D* to duplicate the **Dodge-MixRGB** node, set the **Blend Type** to **Multiply**, and label it as `Multiply2`. Lower the **Fac** value to `0.500`. Connect the **Dodge-MixRGB** node's output to the **Color1** input of the **Multiply2** node and the **Color** output of the **Noise Texture2** node to the **Color2** input.
34. Set the **Noise Texture2** node's **Scale** value to `16.000`, **Detail** to `2.500`, and **Distortion** to `1.000`. Connect the **Object** output of the **Mapping1** node to the **Vector** input of the **Noise Texture2** node.
35. Add a **Hue Saturation Value** node (press *Shift* + *A* and navigate to **Color** | **Hue Saturation Value**). Place it right after the **Multiply2** node. Connect the **Multiply2** output to the **Color** input socket of the **Hue Saturation Value** node. Then set the **Hue** value to `0.465` and the **Saturation** value to `1.050`.
36. Press *Shift* + *D* to duplicate the **Multiply2** node, and place the duplicate close to the **Noise Texture2** node. Set the **Blend Type** to **Overlay** and the **Fac** value to `0.250`. Connect the **Fac** output of the **Noise Texture2** node to the **Color1** input socket of the **Overlay-MixRGB** node, and change **Color2** to pure white. Connect the **Overlay-MixRGB** node's output to the **Value** input socket of the **Hue Saturation Value** node.
37. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select these recently added nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame `RUST_COLOR`.
38. Connect the output of the **Hue Saturation Value** node inside the **RUST_COLOR** frame to the **Color** input socket of the **Diffuse BSDF** shader node inside the **SHADERS** frame. Then connect the **Color** output of the **Divide** node inside the **STENCIL** frame to the **Color** input of the **RGB Curves** node inside the **RUST_COLOR** frame, as shown in the following screenshot:



The color of the rusty surface added to the network

39. Save the file as `Metal_rusty.blend`.

How it works...

From step 2 to step 4, we built the basic shader arrangement. From step 6 to step 15, we made the **STENCIL** frame to separate the rust material from the polished metal.

From step 17 to step 23, we built the bump effect for the rust, and from step 26 to step 37, we added the rust color.

There's more...

We used the **Dirty Vertex Colors** layer named `Col_vp` again, this time to give a denser pattern to certain areas of Suzanne compared to other areas. Remember that a Vertex Colors layer can be modified and improved by manual vertex painting on the mesh in **Vertex Paint** mode. We can also use a gray-scale image map, painted in GIMP or in Blender itself and then UV-mapped on the mesh to obtain more precise and localized effects.

Creating a wood material with procedurals

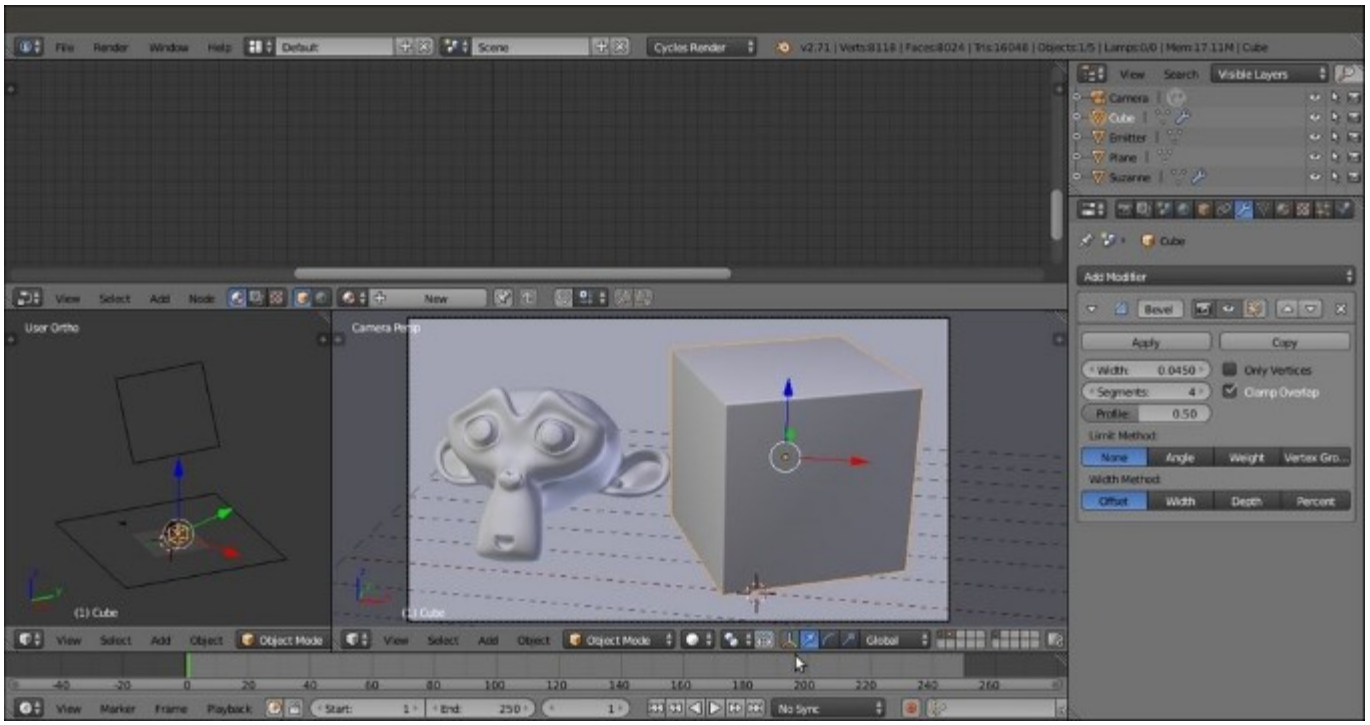
In this recipe, we will create a generic wood material—a shader that can be easily adapted to different situations—as shown in the following screenshot:

The procedural wood material as it appears in the final rendering

Getting ready...

Start Blender and load the `9931OS_Suzanne_start.blend` file. Then perform these steps:

1. Go to the **World** window and click on the button with a dot icon to the right of the **Color** slot under the **Surface** subpanel. In the pop-up menu, select the **Environment Texture** item.
2. Click on the **Open** button and browse to the `textures` folder to load the `Barce_Rooftop_C_3K.hdr` image.
3. Set the **Strength** value to `0.300`. Then go back to the **Material** window.
4. Go to the **Camera** view and add a **Cube** primitive to the scene. Place it leaning on the Plane, to the right of Suzanne. Move it up by 1 Blender unit.
5. With the mouse arrow in the **Camera** view, press *Shift* + *F* to enter **Walk Mode**. Adjust the Camera position to center the two objects in the frame.
6. Select the **Cube**, go to **Edit Mode**, and scale it to at least twice its current size. Exit **Edit Mode**, and using the 3D manipulator widget (which can be enabled in the 3D view toolbar), move the Cube upwards to stay nicely on the Plane. Press *N*, and in the **Properties** panel, select the **Lock Camera to View** item. Then adjust the Camera position framing the two objects.
7. Assign a **Bevel** modifier to the Cube, set **Width** to `0.0450`, and set the **Segments** value to `4`.
8. Press *T* to call the **Tool Shelf** panel. Set the Cube shading to **Smooth**.
9. Select **Suzanne** and rotate it a bit towards the left on the *z* axis.
10. Press *T* to close the **Tool Shelf** panel.



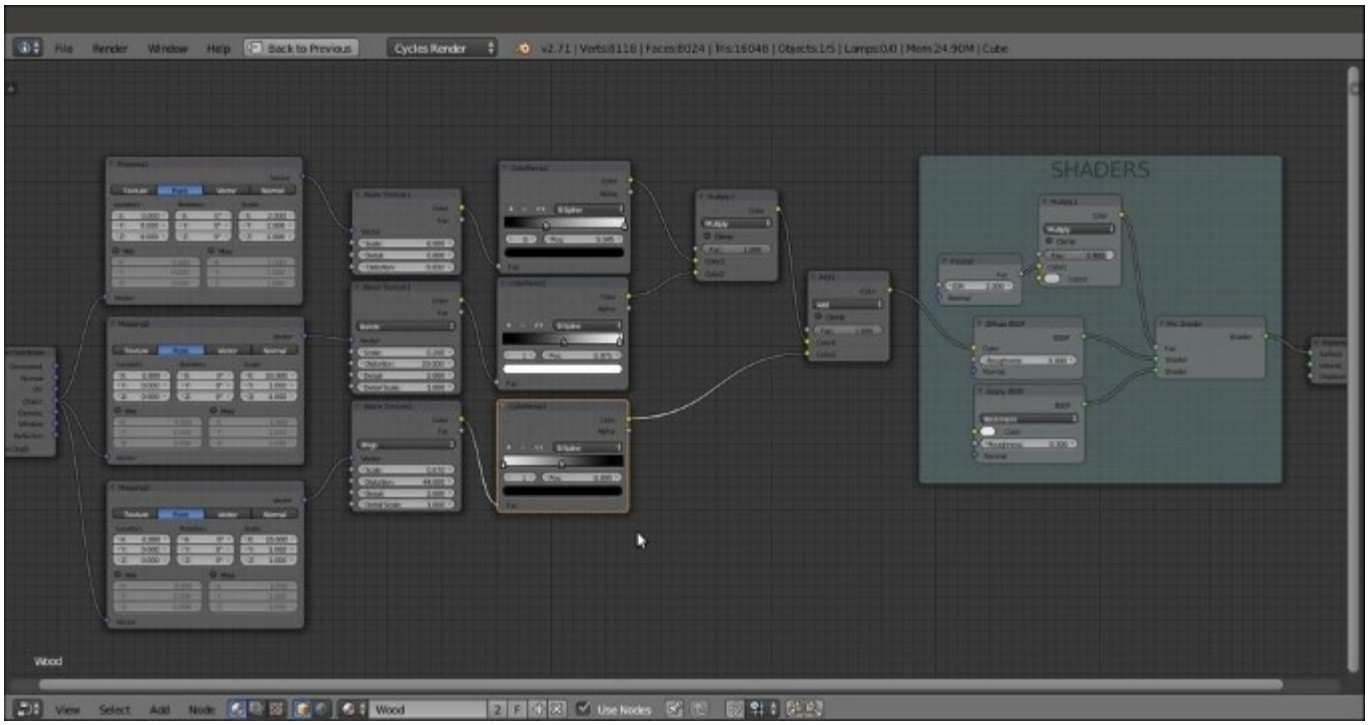
Setting up the scene

How to do it...

Now we are going to create the material by performing the following steps:

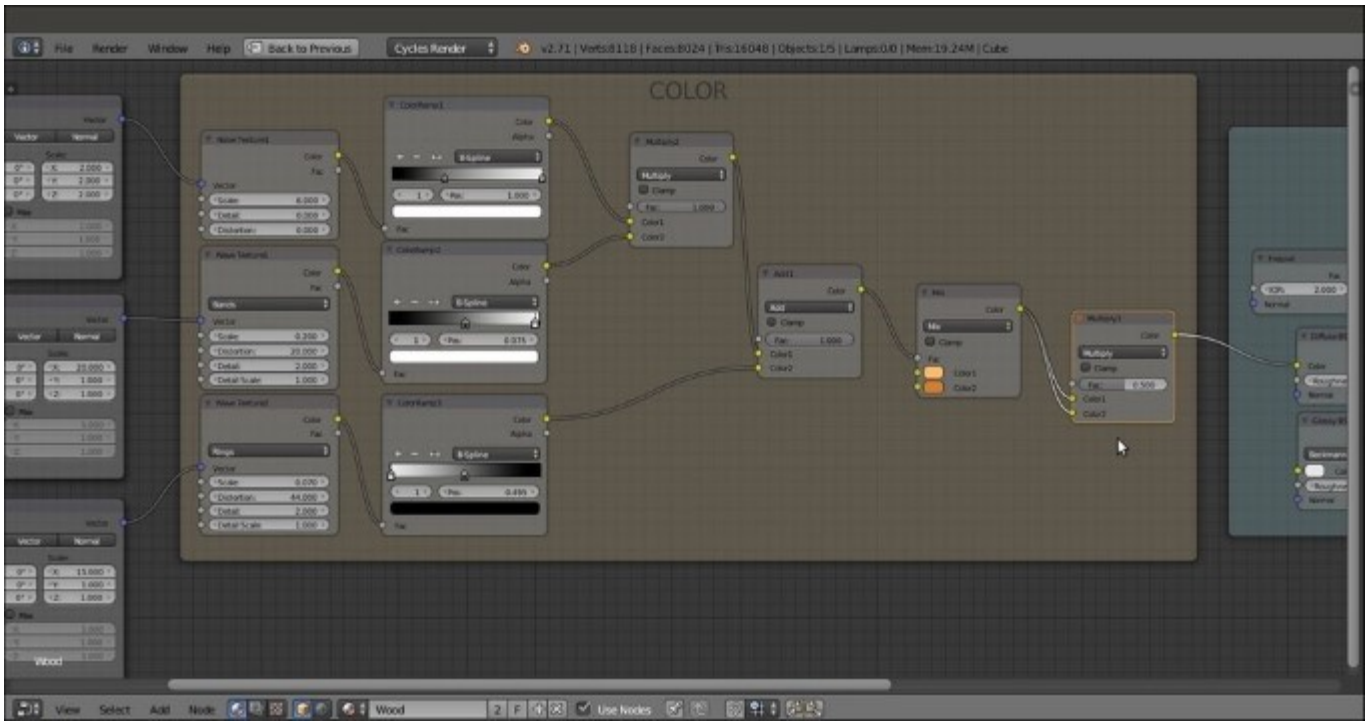
1. Click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material `Wood`.
2. Switch the **Diffuse BSDF** shader with a **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader. In the second **Shader** slot, select a **Glossy BSDF** node. Set the **Glossy BSDF** node's **Roughness** value to `0.300`.
3. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **IOR** value of the **Fresnel** node to `2.000`. Connect its output to the **Color1** input socket of the **MixRGB** node. Set the **MixRGB** node's **Blend Type** to **Multiply**, label it as `Multiply1`, and set the **Fac** value to `0.900`. Connect the **Multiply1** node's output to the **Fac** input socket of the **Mix Shader** node.
4. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select the **Diffuse BSDF**, **Glossy BSDF**, **Mix Shader**, **Multiply1**, and **Fresnel** nodes. Then select the **Frame** and press *Ctrl* + *P* to parent them. Label the frame as `SHADERS`.
5. Add one **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and three **Mapping** nodes (press *Shift* + *A*; navigate to **Vector** | **Mapping**; add the first node; duplicate the other nodes; and then label them as `Mapping1`, `Mapping2`, and `Mapping3`). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input of the three **Mapping** nodes.

6. Set the **Scale** value of the **Mapping1** node to `2.000` for all the three axes. Set the **Scale** value only for the *x* axis of the **Mapping2** node to `20.000`. Then set the **Scale** value only for the *x* axis of the **Mapping3** node to `15.000`.
7. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and two **Wave Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**). Label them as `Noise Texture1`, `Wave Texture1`, and `Wave Texture2`.
8. Set the **Scale** of the `Noise Texture1` node to `6.000` and **Detail** to `0.000`. Connect the **Mapping1** node's output to the **Noise Texture** node's **Vector** input socket.
9. Connect the **Mapping2** node's output to the **Vector** input of the **Wave Texture1** node. Set the **Wave Texture1** node's **Scale** value to `0.200` and **Distortion** to `20.000`.
10. Connect the **Mapping3** node output to the **Wave Texture2** node's **Vector** input socket. Set **Wave Type** to **Rings**, the **Scale** value to `0.070`, and the **Distortion** value to `44.000`.
11. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Blend Type** to **Multiply** (label it as `Multiply1`) and the **Fac** value to `1.000`. Connect the **Noise Texture** node's **Color** output to the **Color1** input socket and the **Wave Texture1** node's **Color** output to the **Color2** input socket.
12. Connect the **Multiply1** node's output to the **Color** input of the **Diffuse BSDF** shader. Press *Shift* + *D* to duplicate it, change the **Blend Type** to **Add**, and paste it between the **Multiply1** node and the **Diffuse BSDF** shader node. Connect the **Wave Texture2** node's **Color** output to the **Color2** input socket of this **Add-MixRGB** node (labelled as `Add1`).
13. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp1`, and paste it right after the **Noise Texture1** node. Set **Interpolation** to **B-Spline** and move the black color stop to the `0.345` position.
14. Press *Shift* + *D* to duplicate the **ColorRamp1** node, paste it right after the **Wave Texture1** node, and label it as `ColorRamp2`. Move the black color stop to the `0.505` position and the white color stop to the `0.975` position.
15. Press *Shift* + *D* to duplicate the **ColorRamp2**, label it as `ColorRamp3`, and paste it right after the **Wave Texture2** node. Move the black color stop to the `0.495` position and the white color stop to the left end of the slider, and set **Pos** as `0.000`, as shown in the following screenshot:



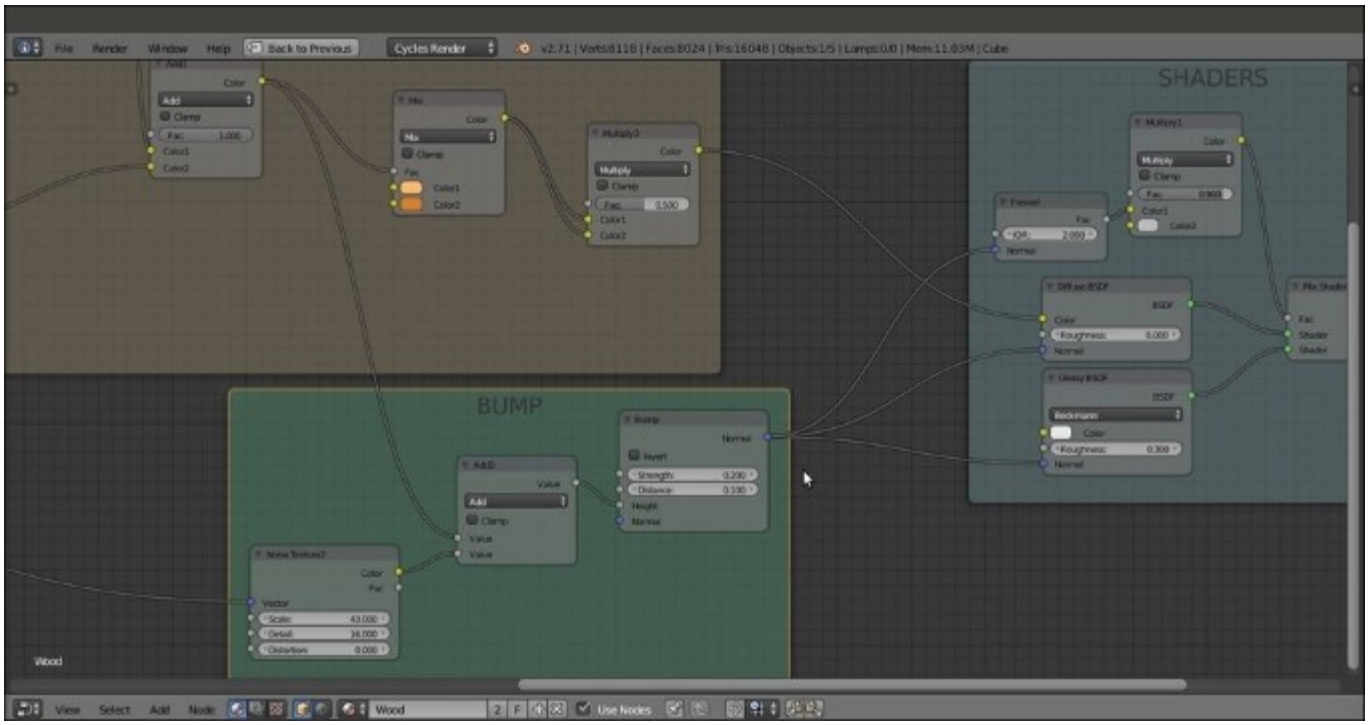
The textures required to draw the wood effect summed and connected to the shader part of the material

16. Now add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect the **Add1** node's **Color** output to its **Fac** input socket. Set the **Color1** values of **R** to `1.000`, **G** to `0.500`, and **B** to `0.150`. Set the **Color2** values of **R** to `0.694`, **G** to `0.205`, and **B** to `0.027`.
17. Press *Shift* + *D* to duplicate the **MixRGB** node. Paste the duplicate right after the original node. Connect the **MixRGB** node's output to the **Color2** input socket, change **Blend Type** to **Multiply**, and label it as `Multiply3`.
18. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the three texture nodes, the three **ColorRamp** nodes, the four **MixRGB** nodes, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `COLOR`, as shown in the following screenshot:



Adding more color to the veining

19. Add a new **Noise Texture** node (press *Shift* + *A*, navigate to **Texture** | **Noise Texture**, and label it as `Noise Texture2`), a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**).
20. Connect the **Mapping3** node's output to the **Vector** input socket of the **Noise Texture2** node. Then connect the **Color** output of this node to the second **Value** input of the **Math** node. Set its **Operation** to **Add**, label it as `Add2`, and connect its output to the **Height** input socket of the **Bump** node.
21. Set the **Bump** node's **Strength** value to `0.200`. Connect the **Normal** output of the **Bump** node to the **Normal** input of the **Fresnel**, **Diffuse BSDF**, and **Glossy BSDF** nodes inside the **SHADERS** frame. Set the **Noise Texture2** node's **Scale** value to `43.000` and **Detail** to `16.000`.
22. Go to the **Add1** node inside the **COLOR** frame, click on the output node, and drag it so that it is connected to the first **Value** input socket of the **Add2-Math** node.
23. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Select the three nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `BUMP`, as shown in the following screenshot:



The bump pattern, based in part on the output of the veining

24. Save the file as `Wood.blend`.

How it works...

From steps 1 to 4, we built the basic shader using the usual **Diffuse BSDF** and **Glossy BSDF** nodes, mixed by a **Fresnel** value and multiplied by the values of a medium gray color.

From steps 5 to 18, we built the color of the wood's veins, adding three procedurals to be used as splitting factors for the two wood colors set in the penultimate **MixRGB** node. Using the last **Multiply3** node, we made the color more saturated (actually, we multiplied the values by themselves).

From steps 19 to 23, we built the bump using a noise grain summed to the veins' values by the **Add2-Math** node. We set a low value for the bump's **Strength** value, but you can use higher values (together with higher roughness values) to obtain less polished surfaces, which can give you different kinds of wood in the output.

Chapter 5. Creating Complex Natural Materials in Cycles

In this chapter, we will cover the following recipes:

- Creating an ocean material using procedural textures
- Creating underwater environment materials
- Creating a snowy mountain landscape with procedurals
- Creating a realistic earth as seen from space

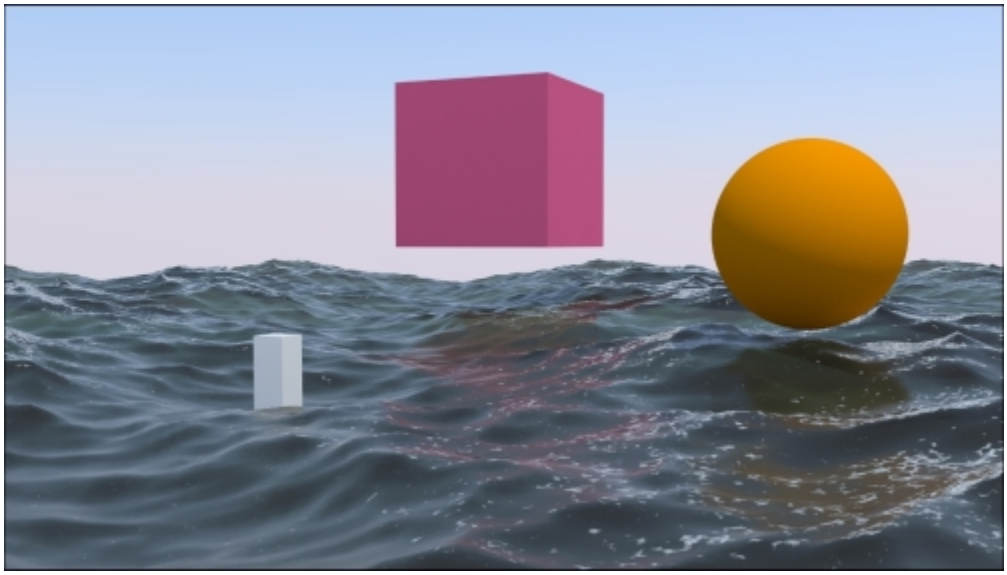
Introduction

In Chapter 3, *Creating Natural Materials in Cycles*, we saw some of the simpler natural materials that are possible to build in Cycles, keeping them out of any landscape context to make them more easily understandable.

Now it's time to deal with more elaborate natural materials. In this chapter, we will examine the way to mix different basic shaders to mimic the look of complex natural objects and their environments (very often, these two things fit together neatly).

Creating an ocean material using procedural textures

In this recipe, we will build an ocean surface material, using the **Ocean** modifier and procedural textures to create the foam, and establish a set of nodes to locate it on the higher parts of the waves:



The final look of the ocean material, with three simple and brightly colored objects to be reflected by the surface

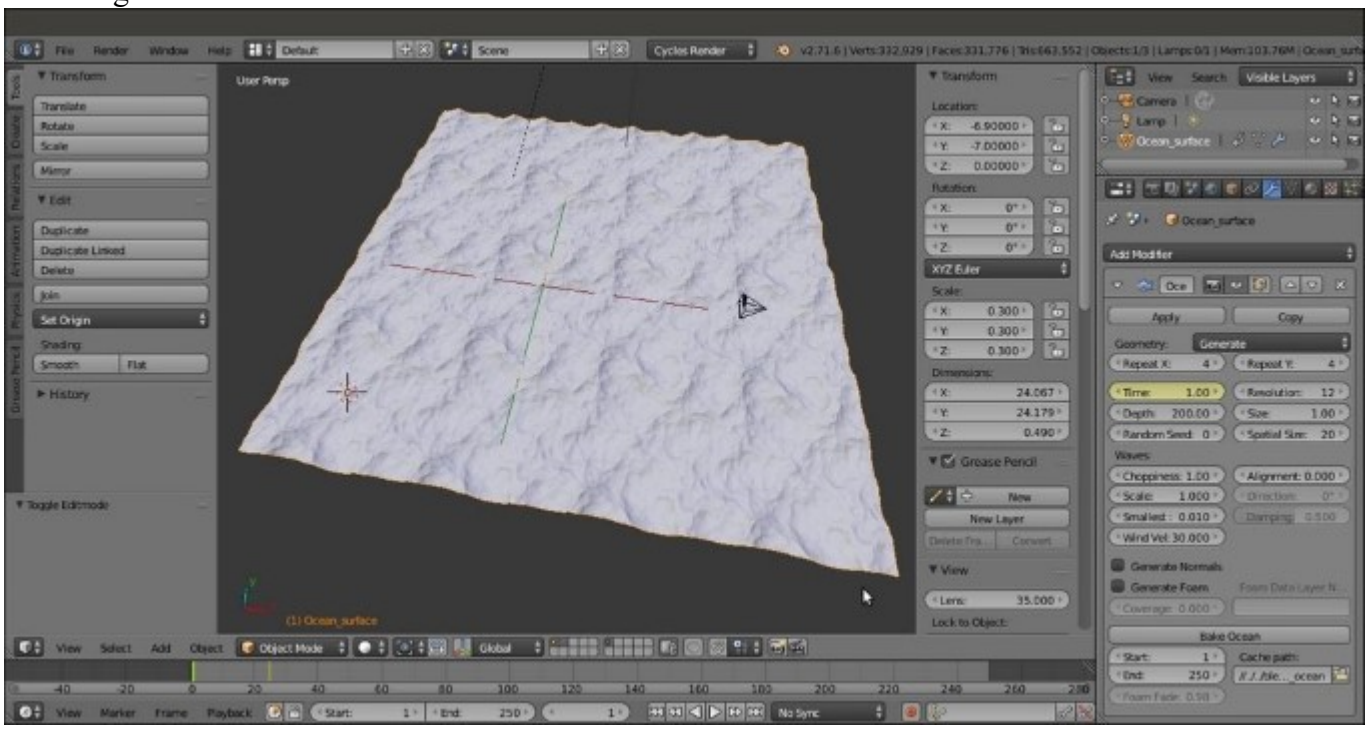
Getting ready

Before we start with creating the shaders, let's prepare the ocean scene:

1. Start Blender and switch to the Cycles rendering engine. Select the default **Cube**, delete it (press *X*), and add a **Plane** (with the mouse arrow in the 3D window, press *Shift* + *A* and navigate to **Mesh** | **Plane**). In **Object Mode**, scale the plane smaller to `0.300`. Don't apply a size.
2. Go to the **Object modifiers** window and assign an **Ocean** modifier. Set these values of **Geometry** to **Generate**, **Repeat X** and **Repeat Y** to `4`, **Spatial Size** to `20`, and **Resolution** to `12`.
3. Press the *N* key, and in the **Transform** panel, set these values for the Plane in **Location** for **X** as `-6.90000`, **Y** as `-7.00000`, and **Z** as `0.00000`.
4. Make sure that you are at frame 1, and with the mouse arrow in the modifier's **Time** slot, press *I* to add a key for the animation. Go to frame 25, change the **Time** value from `1.00` to `2.00`, and press the *I* key again to set a second key.
5. In the **Choose Screen layout** button at the top, switch from **Default** to **Animation**. In the **Graph Editor** window, press *T*. In the **Set Keyframe Interpolation** pop-up menu, select the **Linear** item under **Interpolation**. Then press *Shift* + *E*, and in the **Set Keyframe**

Extrapolation pop-up menu, select the **Linear Extrapolation** item to make the ocean animation constant and continuous.

6. Go back to the **Default** screen and rename the **Plane** as `Ocean_surface`. Have a look at the following screenshot:



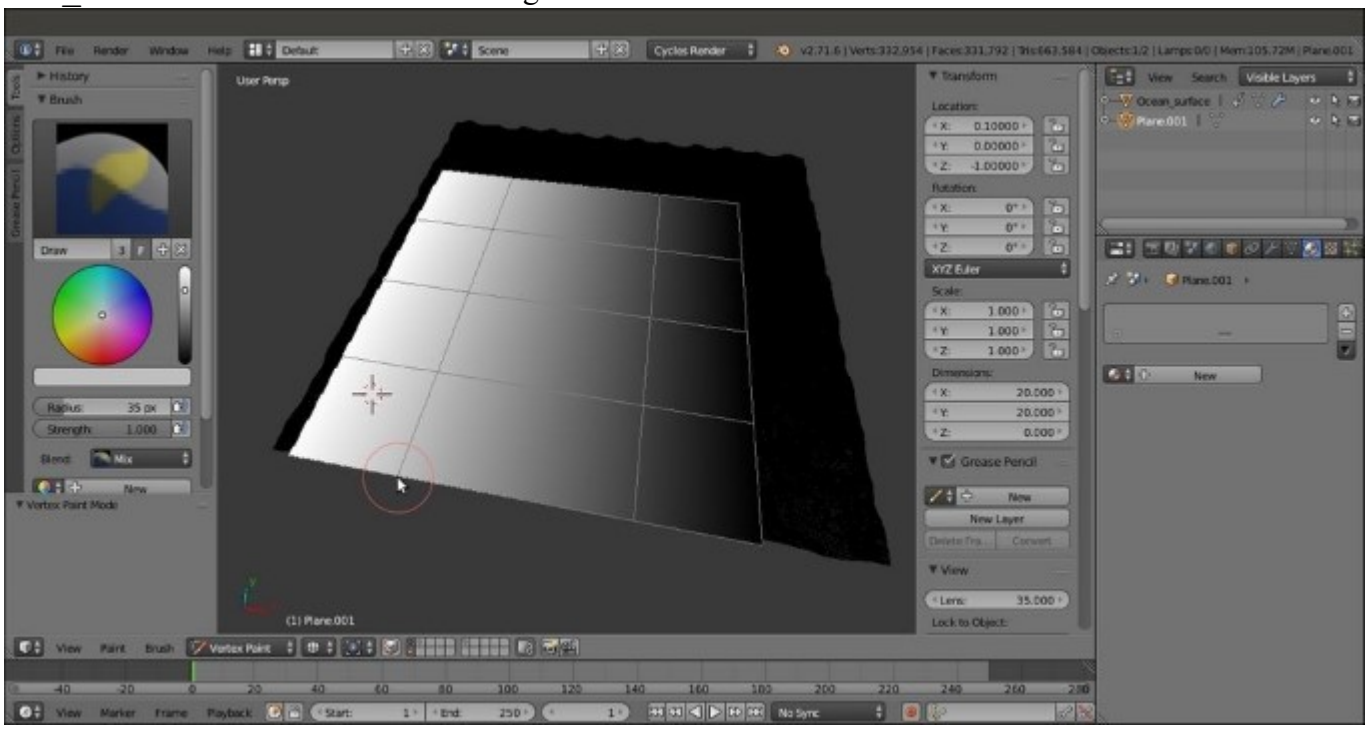
The Plane with the assigned Ocean modifier, and the settings to the right

7. Place the **Camera** to have a nice angle on the ocean, and then go to the **Camera** view (press *0* on the numeric keypad).
8. Add a **Cube** in the middle of the scene and, if you want, a UV Sphere in the foreground. Place these so that they float in the air. Their only purpose is to get reflected by the ocean surface during the shader setup.
9. Go to the **World** window and click on the **Use Nodes** button under the **Surface** subpanel. Then click on the little square with a dot on the right side of the color slot. From the pop-up menu, navigate to **Texture** | **Sky Texture**.
10. Select the **Lamp**, go to the **Object data** window, and click on the **Use Nodes** button under the **Nodes** subpanel. Set a yellowish color for the light (change the value of **R** to `1.000`, **G** to `0.989`, and **B** to `0.700`). Turn it to **Sun**, set the **Size** value to `0.010`, and set the **Strength** value to `2.500`.
11. Go to the **Render** window, and under the **Sampling** subpanel, set both the **Clamp Direct** and **Clamp Indirect** values to `1.00`. Set the samples to `100` for **Render** and `50` for **Preview** (you can obviously change these values according to the power of your machine).

    Now, because the shader we are going to build is largely transparent, we need to simulate the water body as seen from above the surface.

12. Add a new **Plane** in **Edit Mode** and scale it `10` times bigger (20 Blender units per side; press *Tab*, then press *S*, enter digit `10`, and press *Enter*). Exit **Edit Mode** and move the Plane so that it is centered on the ocean Plane location, then move it 1 unit down on the *z* axis. You can do it like this: go to the **Top** view and move the new Plane of 7 Blender units first along the *x* axis and then along the *y* axis. Then press *G*, press *Z*, enter digit `-1`, and press *Enter*.
13. In **Edit Mode**, press *W* to subdivide it by the **Specials** menu. Then press *T* to open the **Tool Shelf** panel on the left, and under **Number of Cuts** in the **Operator** panel at the bottom, and select **3**.
14. Go to the **Vertex Paint** mode and paint a very simple gray-scale gradient, changing from black at the vertices close to the Camera location to a plain white color on the opposite side.

    There are five rows of vertices on the Plane (ideally, all the rows are along the global *x* axis), so you can paint the first row with **RGB** value as `0.000`, second with **RGB** value as `0.250`, third with **RGB** value as `0.500`, fourth with **RGB** value as `0.750`, and fifth with **RGB** value as `1.000` to have a perfect gray-scale gradient.
15. In the **Object data** window, under the **Vertex Colors** tab, rename the Vertex Color layer as `Col_emit`. Have a look at the following screenshot:



The Ocean_Bottom plane with the painted Vertex Colors layer

16. Exit **Vertex Paint** mode and rename this second Plane `Ocean_bottom`.
17. Split the 3D window into two horizontal rows. Change the upper row to a **Node Editor** window.
18. Assign very simple colored materials to the Cube and the UV Sphere; plain **Diffuse BSDF** shaders are enough.

How to do it...

We will be performing this in four parts:

- Creating the water surface and the bottom shaders
- Creating the foam shader
- Creating the stencil material for the location of foam
- Putting everything together

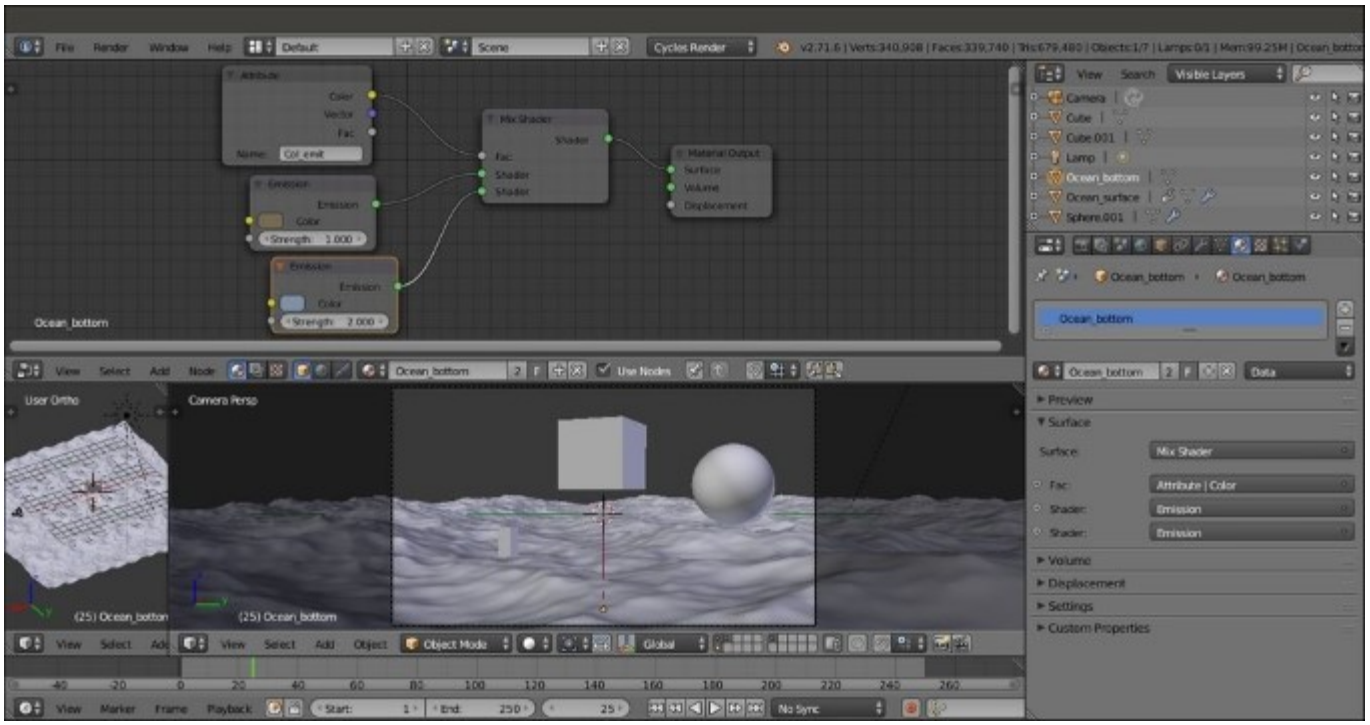
Let's start!

Creating the water surface and the bottom shaders

Let's now create the water surface and the bottom shaders:

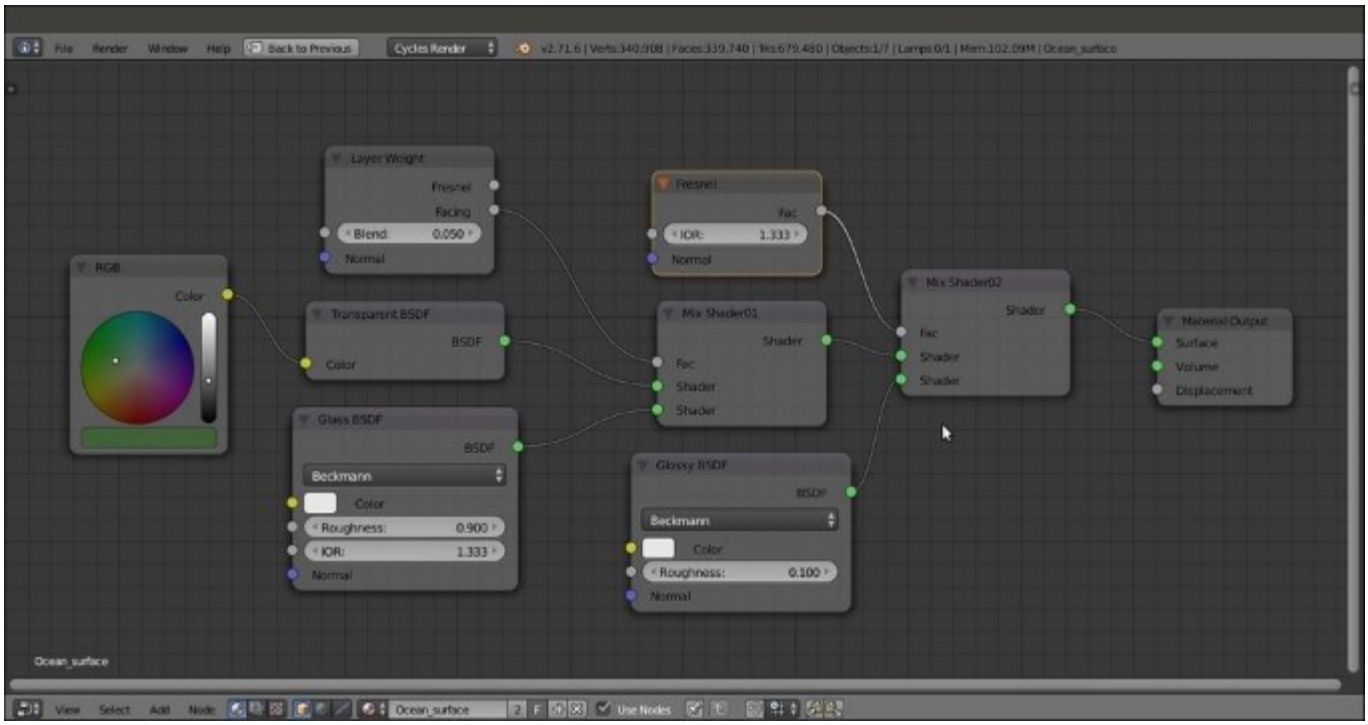
1. Select the **Ocean_bottom** object. Click on the **New** button in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the new material as `Ocean_bottom` as well.
2. Switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first and the second slots, load two **Emission** shaders.
3. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**) and connect the **Color** output to the **Fac** input of the **Mix Shader** node. In the **Name** slot of the **Attribute** node, write `Col_emit`, which is the name of the **Vertex Color** layer.
4. Change the color of the first **Emission** node for **R** to `0.178`, **G** to `0.150`, and **B** to `0.085`. Set the **Strength** value to `1.000`.
5. Change the color values of the second **Emission** node for **R** to `0.213`, **G** to `0.284`, and **B** to `0.380`. Set the **Strength** value to `2.000`.

   The **Ocean_bottom** material is ready. Have look at the following screenshot:



The Ocean_bottom material and the scene visible in the Solid viewport shading mode through the Camera view

6. Now select **Ocean_surface** and click on **New** in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename this material as `Ocean_surface`.
7. Replace the **Diffuse BSDF** node with a **Mix Shader** node, and in the first **Shader** slot, assign a **Transparent BSDF** node. In the second slot, assign a **Glass BSDF** shader. In the **Properties** panel of the **Node Editor** window, label the **Mix Shader** node as `Mix Shader01`.
8. Change the **Transparent BSDF** nodes **Color** values for **R** to `0.055`, **G** to `0.124`, and **B** to `0.042` (you can also do this by connecting an **RGB** node to the **Color** input socket, as shown in the example blend file provided). Set the **Glass BSDF** shader node's **Roughness** value to `0.900` and the **IOR** value to `1.333`.
9. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), connect the **Facing** output to the **Fac** input of the **Mix Shader01** node, and set the **Blend** value to `0.050`.
10. Select the **Mix Shader01** node and press *Shift* + *D* to duplicate it. Add a **Glossy BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**) and connect it to the second **Shader** input socket of the **Mix Shader02** node. Connect the output of the **Mix Shader01** node to the first **Shader** input socket of the **Mix Shader02** node, and the output of this node to the **Surface** input of the **Material Output** node.
11. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**). Connect this to the **Fac** input of the **Mix Shader02** node. Set the **IOR** value to `1.333` as shown in the following screenshot:



The Ocean_surface shader network

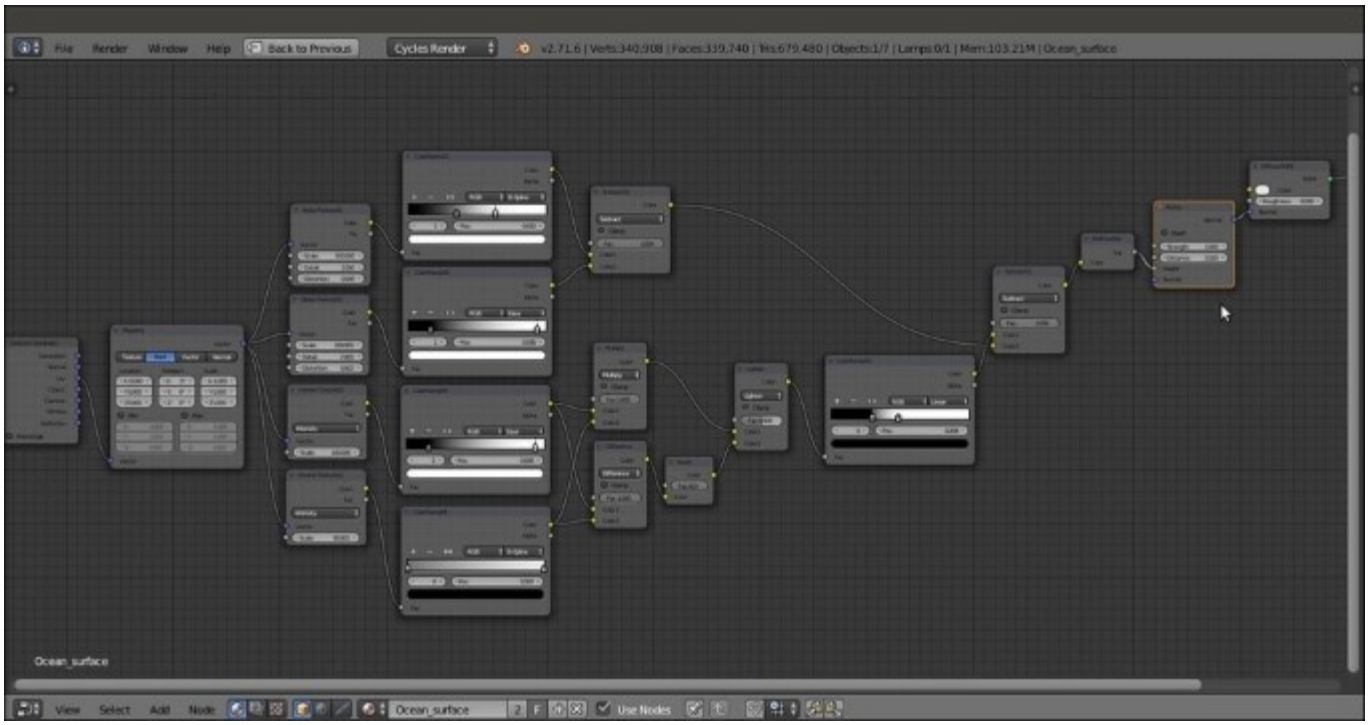
12. Now select all the nodes except the **Material Output** node, and press *Ctrl* + *G* to make a group. Select and delete the **Group Input** node to the left (press the *X* key), and drag the **Mix Shader02** node's output to the empty socket of the **Group Output** node.
13. Press *Tab* to close the node group, and rename it as `Ocean_water`.

Creating the foam shader

Let's now create the shader for the foam:

1. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) nodes. Select them and press *Shift* + *D* to duplicate them. Label them as `Noise Texture01`, `Noise Texture02`, `Voronoi Texture01`, and `Voronoi Texture02`.
2. Add four **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**, then press *Shift* + *D* to duplicate them). Label them as `ColorRamp01`, `ColorRamp02`, `ColorRamp03`, and `ColorRamp04`. Place the four texture nodes in a vertical column and arrange the **ColorRamp** nodes to their side. Connect the **Color** output of each texture node to the **Fac** input of the respective **ColorRamp** node.
3. Set **Interpolation** of the **ColorRamp01** node to **B-Spline**, **ColorRamp02** and **ColorRamp03** to **Ease**, and **ColorRamp04** to **B-Spline** again.
4. Go to the **ColorRamp01** node. Move the black color stop to position `0.345` and the white color stop to position `0.633`.

5. Go to the **ColorRamp02** and **ColorRamp03** nodes. Move the black color stop to position `0.159` and the white color stop to position `0.938` for both nodes. Leave the **ColorRamp04** color stops as they are.
6. Set the **Scale** value of the **Noise Texture01** node to `500.000`, the **Noise Texture02** node to `100.000`, the **Voronoi Texture01** node to `100.000`, and the **Voronoi Texture02** node to `90.000`.
7. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node. Then connect the **Vector** output of this node to the **Vector** input sockets of the four texture nodes.
8. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Subtract**, and label it as `Subtract01`. Set the **Fac** value to `1.000`. Connect the **Color** outputs of the **ColorRamp01** and **ColorRamp02** nodes to the **Color1** and **Color2** input sockets of the **Subtract01** node.
9. Select the **Subtract01** node, press *Shift* + *D* to duplicate it, and set the **Blend Type** to **Multiply**. Label it as `Multiply` and connect the **Color** outputs of the **ColorRamp03** and **ColorRamp04** nodes to its **Color1** and **Color2** input sockets.
10. Duplicate a **MixRGB** node again, set the **Blend Type** to **Difference**, and name it `Difference` as well. Then connect the **Color** outputs of the **ColorRamp03** and **ColorRamp04** nodes to the **Color1** and **Color2** input sockets of this **Difference** node.
11. Duplicate one of the **MixRGB** nodes one more time. Set the **Blend Type** to **Lighten** and label it as `Lighten`. Lower the **Fac** value to `0.500`. Connect the **Color** output of the **Multiply** node to the **Color1** input of the **Lighten** node, and the **Color** output of the **Difference** node to the **Color2** input socket.
12. Add an **Invert** node (press *Shift* + *A* and navigate to **Color** | **Invert**) and move it on the link connecting the **Difference** and the **Lighten** nodes to be automatically pasted in between.
13. Add a new **ColorRamp** node, label it as `ColorRamp05`, and connect the **Lighten** node output to its **Fac** input. Then move the black color stop to position `0.298` and the white color stop to position `0.486`.
14. Add a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Subtract**, and label it as `Subtract02`. Set the **Fac** value to `1.000`. Connect the **ColorRamp05** node's **Color** output to the **Color1** input of **Subtract02** node and the output of the **Subtract01** node to the **Color2** input socket.
15. Add an **RGB to BW** node (press *Shift* + *A* and navigate to **Converter** | **RGB to BW**), a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**), and a **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**).
16. Connect the **Subtract02** node's output to the **RGB to BW** node, the output of this node to the **Height** input socket of the **Bump** node, and the **Normal** output of the **Bump** node to the **Normal** input of the **Diffuse BSDF** shader. Set the **Bump** node's **Strength** value to `1.000` as shown in the following screenshot:



The network for the foam shader

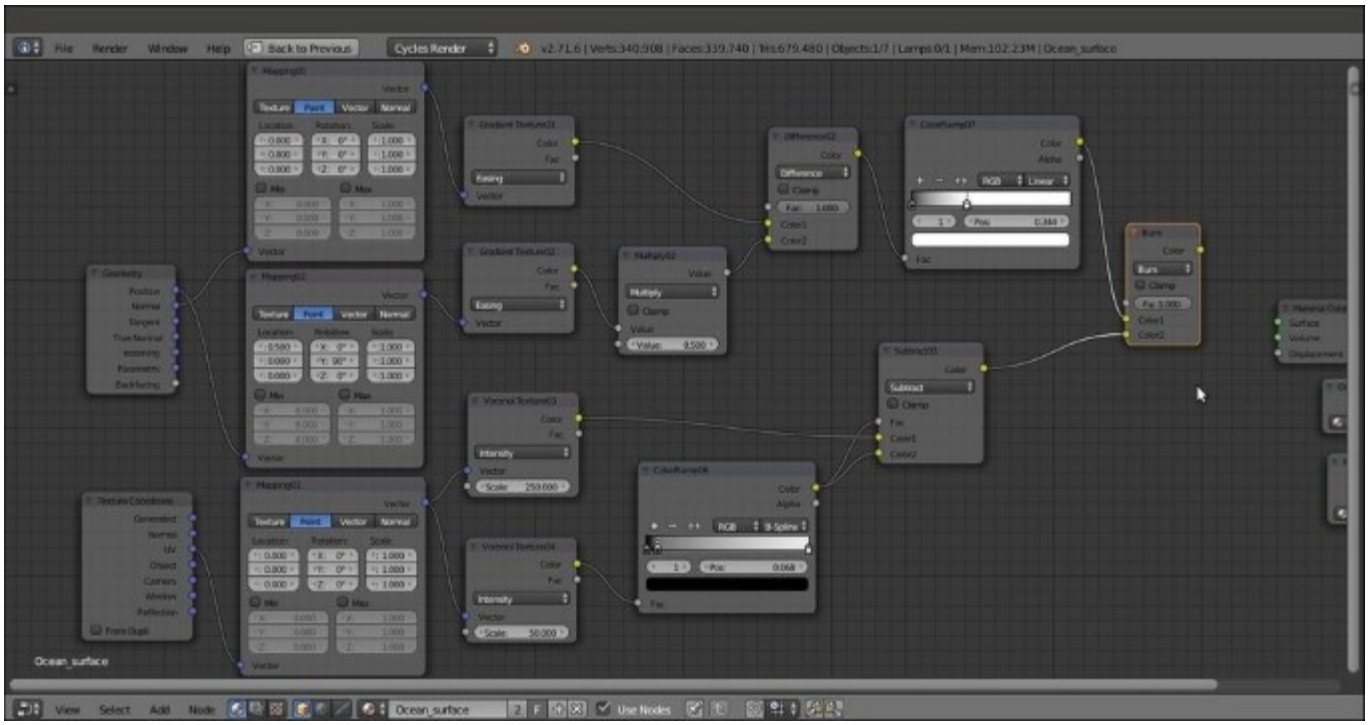
17. Select all of these nodes and press *Ctrl* + *G*. Delete the **Group Input** node on the left.
18. Drag the **BSDF** output of the **Diffuse BSDF** shader onto the right side, to the empty socket of the **Group Output** node. Repeat this for the **Color** output of the **Subtract02** node.
19. Press *Tab* to close the group. Rename it as `Foam`.

Creating the stencil material for the foam location

What we need now is a way to limit both the amount and the presence of the foam in the upper parts of the waves:

1. Add **Gradient** (press *Shift* + *A* and navigate to **Texture** | **Gradient Texture**) and **Voronoi Texture** (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) texture nodes, select them, and press *Shift* + *D* to duplicate them. Label them as `Gradient Texture01`, `Gradient Texture02`, `Voronoi Texture03`, and `Voronoi Texture04`.
2. Set the **Gradient Types** to **Easing** for both the nodes, the **Scale** value of the **Voronoi Texture03** node to `250.000`, and the Scale value of the **Voronoi Texture04** node to `50.000`.
3. Add three **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**, then press *Shift* + *D* to duplicate them). Label them as `Mapping01`, `Mapping02`, and `Mapping03`.
4. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Geometry** node (press *Shift* + *A* and navigate to **Input** | **Geometry**).
5. Connect the **Normal** output of the **Geometry** node to the **Vector** input of the **Mapping01** node. Next, connect the **Position** output of the **Geometry** node to the **Vector** input of the **Mapping02** node. Then connect the **UV** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping03** node.

6. In the **Mapping02** node, change the **Location** value of **X** to `0.500` and the **Rotation** value of **Y** to `90°`.
7. Connect the output of the **Mapping01** to the input of the **Gradient Texture01**, the output of the **Mapping02** to the input of the **Gradient Texture02**, and the output of the **Mapping03** to both the **Vector** inputs of the last two **Voronoi Texture** nodes.
8. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp06`, and connect the **Color** output of the **Voronoi Texture04** node to its **Fac** input. Set **Interpolation** to **B-Spline**. On the **ColorRamp06** node, click on the little **+** icon to add a new color stop (medium gray) in the middle of the slider. Change its color to total black and move it to position `0.068`.
9. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), set the **Operation** to **Multiply**, and label it as `Multiply02`. Connect the **Color** output of the **Gradient Texture02** node to the first **Value** input socket of the **Multiply02** node.
10. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Difference**, and label it as `Difference02`. Set the **Fac** value to `1.000`. Connect the **Color** output of the **Gradient Texture01** to the **Color1** input socket, and the **Value** output of the **Multiply02** node to the **Color2** input socket of the **Difference02** node.
11. Press *Shift* + *D* to duplicate the **MixRGB** node, set the **Blend Type** to **Subtract**, and label it as `Subtract03`. Connect the **ColorRamp06** node's **Color** output to both the **Color2** and to the **Fac** input sockets. Then connect the **Color** output of the **Voronoi Texture03** node to the **Color1** input socket.
12. Add a new **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp07`, and connect the output of the **Difference02** node to the **Fac** input. Then move the white color stop to position `0.344`.
13. Duplicate one of the **MixRGB** nodes, set the **Blend Type** to **Burn**, and label it as `Burn` as well. Connect the **Color** output of the **ColorRamp07** node to the Color1 input of the **Burn** node. Then connect the output of the **Subtract03** node to the **Color2** input socket of the **Burn** node as shown in the following screenshot:



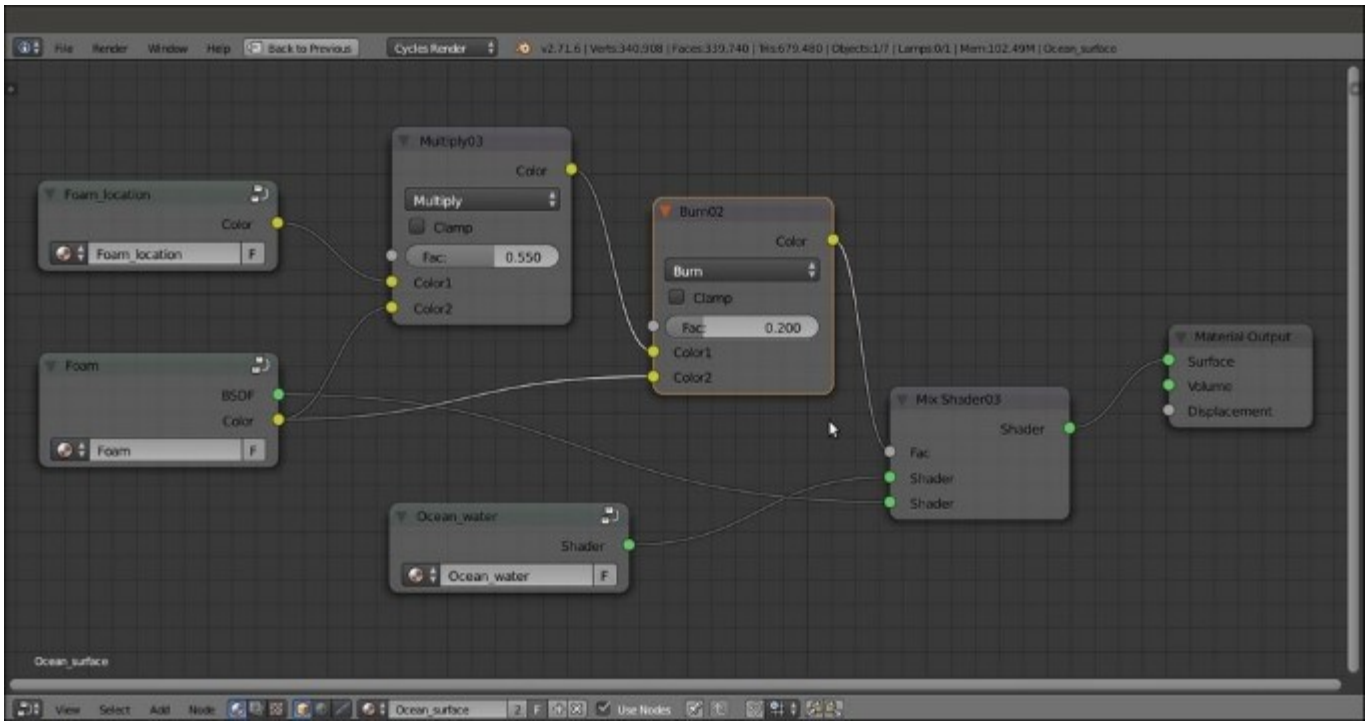
The stencil network

14. Now select all of these nodes, press *Ctrl* + *G*, and drag the **Burn** node output on the right to connect it to the empty socket of the **Group Output** node. Press *Tab* to close the group. Rename it as `Foam_location`.

Putting everything together

What is left now is just to connect these three groups to build the final shader:

1. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**). Label it as `Mix Shader03` and connect its output to the **Surface** input socket of the **Material Output** node.
2. Connect the **Shader** output of the **Ocean_water** group to the first **Shader** input of the **Mix Shader03**. Then connect the **BSDF** output of the **Foam** group to the second **Shader** input.
3. Add two **MixRGB** nodes (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Blend Type** of the first node to **Multiply** and the **Fac** value to `0.550`. Then label it as `Multiply03`. Set the second node **Blend Type** to **Burn** and the **Fac** to `0.200`. Then label it as `Burn02`.
4. Connect the **Color** output of the **Foam_location** group to the **Color1** input of the **Multiply03** node, and the **Color** output of the **Foam** group to the two **Color2** inputs of both the **Multiply03** and **Burn02** nodes.
5. Connect the **Multiply03** node's output to the **Color1** input of the **Burn02** node. Connect the output of the **Burn02** node to the **Fac** input of the **Mix Shader03** node as shown in the following screenshot:

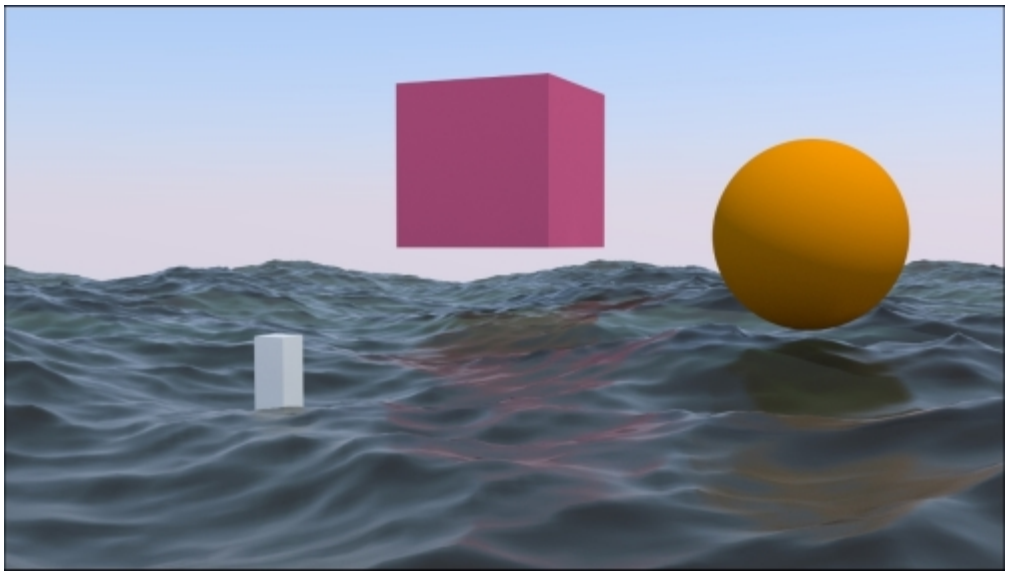


The overall view of the network with the connected node groups

How it works...

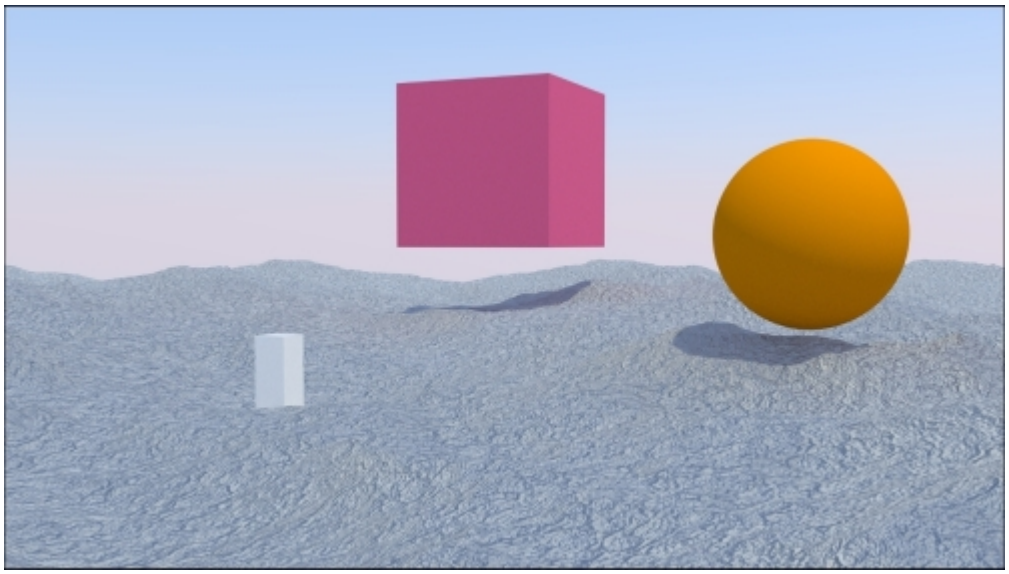
This material, which looks quite complex, is actually easily understandable by splitting the entire process in three stages corresponding to the three group nodes:

- In the first stage, we created the basic ocean water shader by mixing a **Glass** node with a **Transparent BSDF** shader on the ground of the **Facing** value of the **Layer Weight** node and then also with a **Glossy BSDF** shader driven by the index of refraction of water (the **IOR** value of the **Fresnel** node, which is `1.333` for water at 20°C). In other words, the ocean surface nicely reflects the environment but for the faces looking towards the Camera (the **Facing** factor), it is transparent. Very important is the **Bottom_ocean** Plane, which is used to mimic the volume of the water and the underwater perspective and also emitting light to enhance the effect of the sun bouncing from the ocean surface to any floating object. The result of this first stage is shown in the following rendering:



Only the water shader rendered

- In the second stage, we created the material for the foam—a simple, white **Diffuse BSDF** shader. In fact, the peculiarity of the foam shader is mostly in the frothy bumpiness (and in the lacy-shaped outline cut by the procedural textures of the **Foam_location** shader). Have a look at the rendered foam shader:

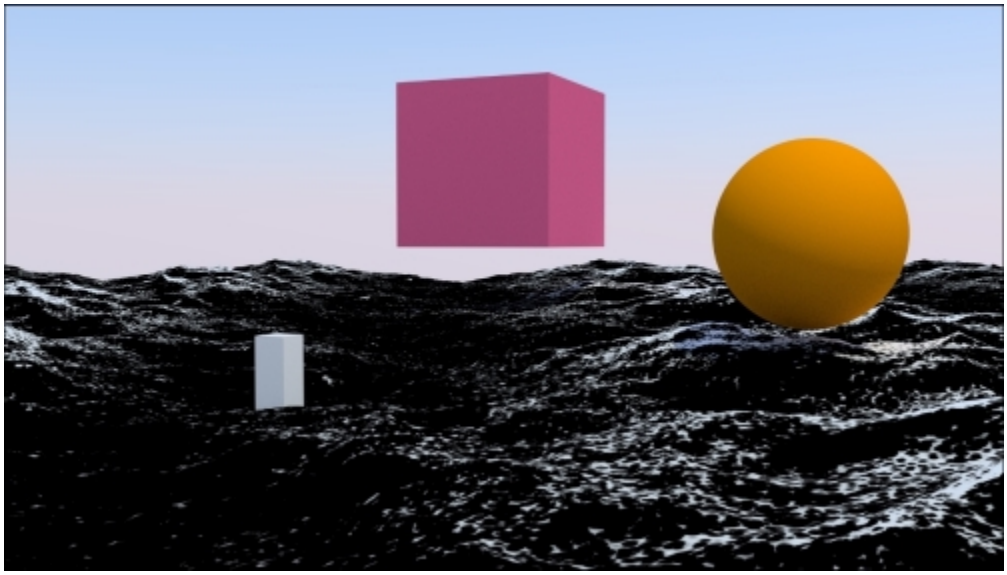


Only the foam shader rendered

- In the third stage, this group of nodes establishes the location of the foam that is mainly formed in the higher parts of waves in the real world, behaving as a gray-scale stencil map. Basically, a gradient texture is mapped on the **Position** (vertices) and multiplied for the **Normal** coordinates of the ocean mesh that, being created by the **Ocean** modifier, is constantly changing. So, only as

the waves rise do they show foam at the top. This effect has been lessened and made a bit random to show some foam scattered around the rest of the surface as well. This works not only for stills but also in animation.

In the following screenshot, you can see the rendering of the resulting black-and-white mask that we used as a stencil for foam location (the image obtained by simply connecting the mask output to an **Emission** shader node to get a quick rendering and preview):



The only stencil material rendered

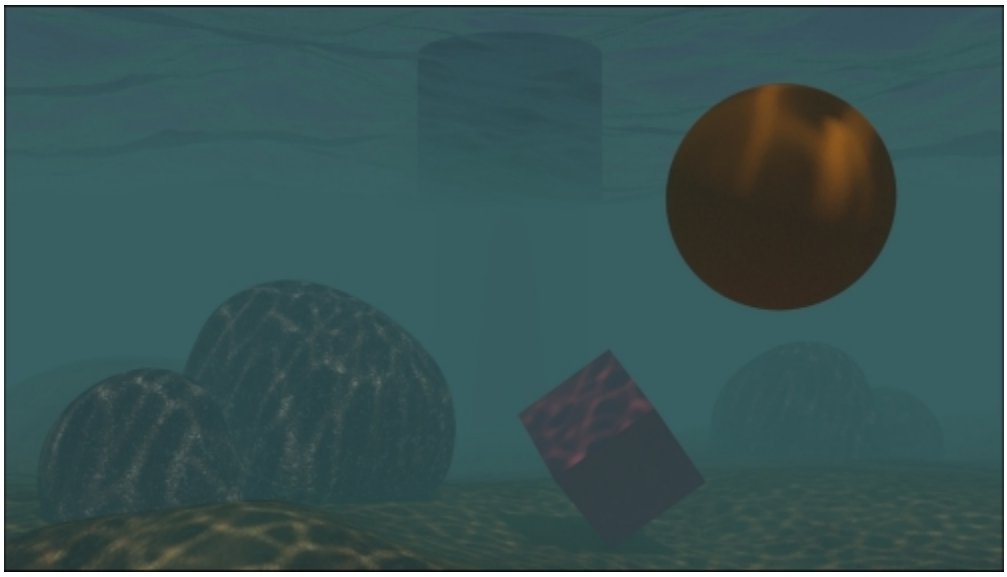
See also

The Blender **Ocean** modifier is able to create its own foam effect, generated as Vertex Colors and baked to a series of images (frames) saved in a directory. These images are then automatically mapped on the surface. They can be used as stencil masks instead of the **Foam_location** group node.

To know more about the **Ocean** modifier, you can take a look at the wiki documentation at http://wiki.blender.org/index.php/Doc:2.6/Manual/Modifiers/Simulate/Ocean.

Creating underwater environment materials

In this recipe, we will create an underwater environment as shown in the following screenshot, looking especially at a fake caustic effect projected by the water's wavy surface and from an atmospheric perspective, obtained by a per material dedicated node group:



The underwater environment in the final rendering

Note

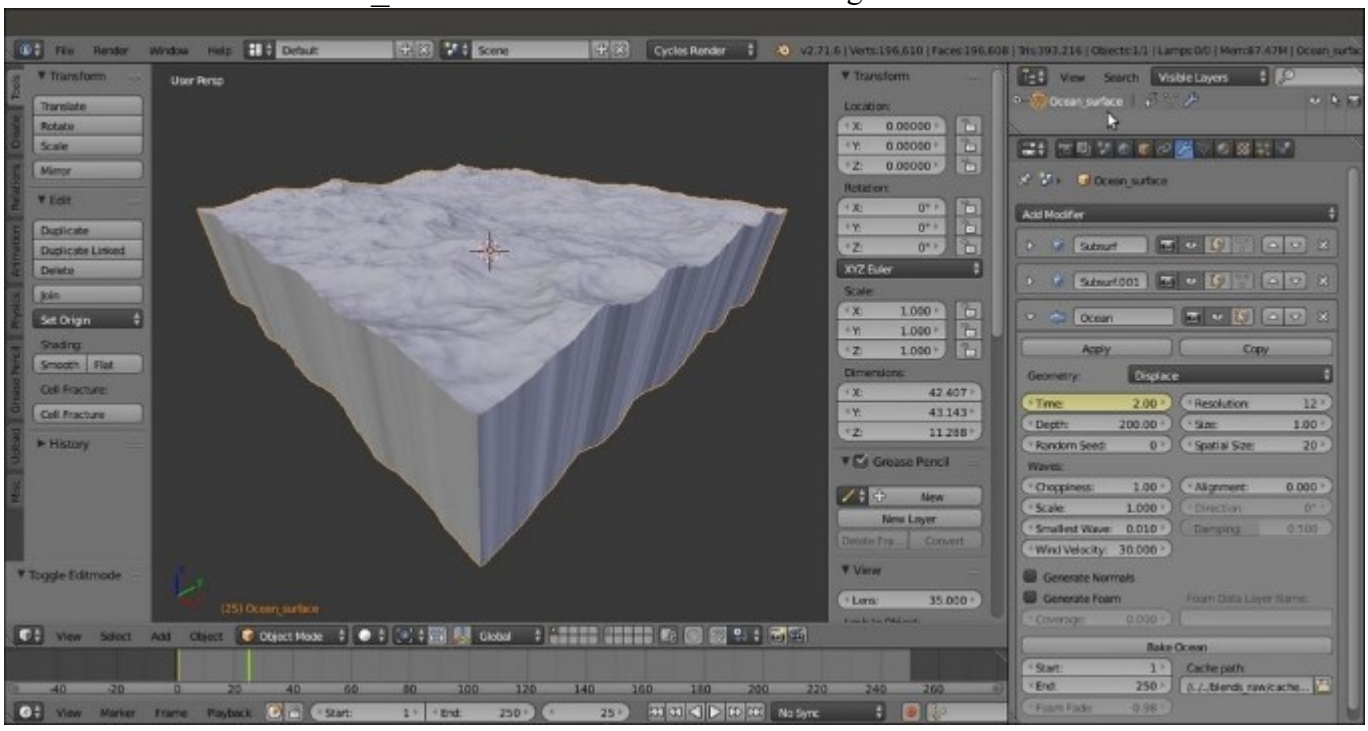
Note that this atmospheric perspective effect is actually a fake and it is not obtained by a volume material. Volumetric shaders will be explained in Chapter 9, *Special Materials*, of this Cookbook.

Getting ready

Let's start by preparing the scene:

1. Start Blender and switch to the Cycles rendering engine. Select the **Cube** and go to **Edit Mode**. Scale it 21 times bigger (Press *A* to select all the vertices, then press *S*, enter digit *21*, and finally press *Enter*). Then scale it on the *z* axis of `0.230` (press *S*, then press *Z*, enter `.23`, and press *Enter*).
2. Go to the **Top** view (press *7* on the numeric keypad) and press *Ctrl* + *R* to add three edge loops along the global *x* axis. Then press *Ctrl* + *R* again to add three edge loops along the *y* global axis. The Cube is now subdivided into 16 equal parts.
3. Select all the faces, go to the **Shading** tab under the **Tool Shelf** panel to the left, and click on the **Flip Direction** button to invert the normals (which must look inwards).
4. Exit **Edit Mode**, switch to the **Objects modifiers** window, and assign a **Subdivision Surface** modifier. Set the type of subdivision to **Simple** and the **Subdivisions** to `4` both for the **View** and the **Render** levels.

5. Assign a second **Subdivision Surface** modifier. Again, set the type of subdivision to **Simple** but the **Subdivisions** to `2` for both the **View** and the **Render** levels.
6. Now assign an **Ocean** modifier. Set the values of **Geometry** to **Displace**, **Spatial Size** to `20`, and **Resolution** to `12`.
7. Go to the **Tool Shelf** again, and in the **Edit** subpanel under the **Tools** tab, click on the **Smooth** button below the **Shading** item.
8. Go to the **Object data** window and click on the **+** icon under the **UV Maps** subpanel to add a set of UV coordinates. There is no need to unwrap the Cube. Check the **Double Sided** item in the **Normals** subpanel at the top.
9. Make sure you are at frame **1**, go to the **Object modifiers** window, and move the mouse over the **Ocean** modifier **Time** slot. Press the *I* key to add a key for the animation. Go to frame **25**, change the **Time** value from `1.00` to `2.00`, and press *I* again to set a second key.
10. In the **Choose Screen** layout button at the top, switch from **Default** to **Animation**. In the **Graph Editor** window, press *T*, and in the **Set Keyframe Interpolation** pop-up menu, select the **Linear** item under **Interpolation**. Then press *Shift* + *E*, and in the **Set Keyframe Extrapolation** pop-up menu, select the **Linear Extrapolation** item to make the ocean animation constant and continuous.
11. Rename the Cube as `Ocean_surface` as shown in the following screenshot:



The Cube with the assigned Ocean modifier

12. Move the Camera to a place below the ocean surface Set the **Location** value of **X** to `16.80000`, **Y** to `-2.64000`, and **Z** to `0.95000`. Then set the **Rotation** value of **X** to `92°`, **Y** to `0°`, and **Z** to `90°`. Next, go to the **Camera** view (press *0* on the numeric keypad).
13. Add a **Cube**, a **UV Sphere,** and whatever other objects you want to add floating under the ocean surface. Assign them very simple and colored **Diffuse BSDF** materials. Add a big **Cylinder** to

the background, close to the far side of the **Ocean_surface** object. Immerse half of it in water (and half will be in the air). Assign a simple **Diffuse BSDF** material to this item too. Smooth the Cylinder and the UV Sphere. If you wish, assign a **Subdivision Surface** modifier to the UV Sphere.

14. Now add a **Plane**. Place it at **Location** values of **X** as `-3.22600`, **Y** as `-2.79600`, and **Z** as `-2.24463`. Enter **Edit Mode** and scale it 30 times bigger (press *A* to select all of the geometry, then press *S*, enter *30* and press *Enter*). Using the **Specials** menu (press *W*) divide the Plane five or six times. Activate the **PET** (**Proportional Editing Tool**), randomly select vertices, and move them up to model the dunes of the ocean bed. Exit **Edit Mode**, smooth it by the **Tools** tab under the **Tool Shelf** panel, and assign a **Subdivision Surface** modifier at level `2`. Disable the modifier visibility in the viewport by clicking on the eye icon. Rename it as `Ocean_bed`.
15. Add a **Cube**. In **Edit Mode**, divide it a couple of times (press *W*, and **Subdivide Smooth**), in **Proportional Editing** mode and by selecting vertices quickly model a big round rock. Replicate it three or four times by rotating and scaling the copies. Place them in a scattered manner on the **Ocean_bed**. Smooth it and assign a **Subdivision Surface** modifier. Disable the modifier visibility in the viewport.
16. Go to the **World** window and click on **Use Nodes**. Then click on the little square with a dot on the right side of the color slot. From the menu, select **Sky Texture**. Click on the **Sky Type** button above the little window and switch to the **Preetham** type.
17. Select the **Lamp**. In the **Object data** window, click on the **Use Nodes** button and set a yellowish color for the light (set the values for **R** to `1.000`, **G** to `0.989`, and **B** to `0.700`). Change it to a **Sun**. Set the **Size** value to `0.010` and the **Strength** value to `2.500`. Then set the **Rotation** values of **X** to `22°`, **Y** to `-7°`, and **Z** to `144°`. You might know that for a Sun Lamp, the location doesn't matter.
18. Go to the **Render** window. Under the **Sampling** subpanel, set the **Clamp Direct** and the **Clamp Indirect** values to `1.00`. Then set the **Samples** to `25` for both **Preview** and **Render**. Under the **Light Paths** subpanel, disable both the **Reflective Caustics** and **Refractive Caustics** items.

As an alternative, just open the `9931OS_05_underwater_start.blend` file and use the prepared scene.

How to do it...

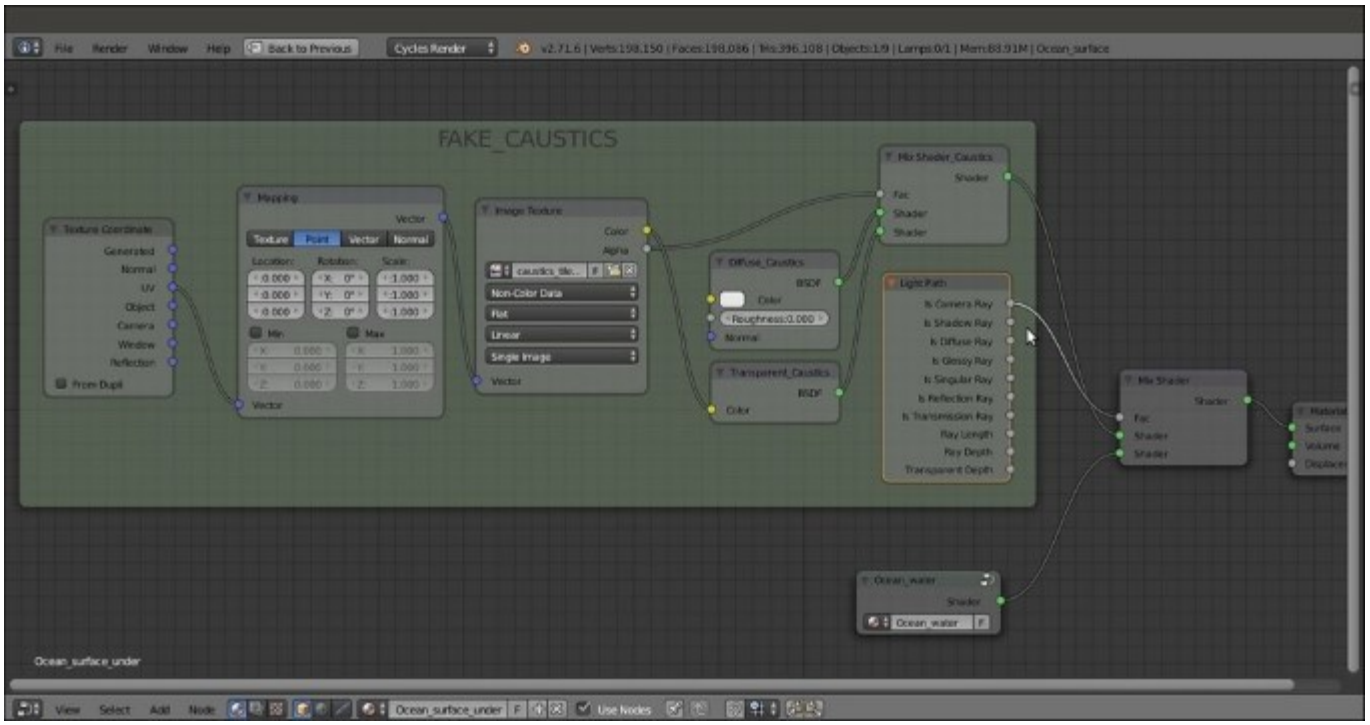
First, let's perform the easy steps by appending the materials that are already made so that we can reuse them:

1. From the `9931OS_03_Rock_procedurals.blend` file, append the `Rock_proc01` material. Select the **Rocks** object and assign the newly appended material.
2. From the `9931OS_03_Ground.blend` file, append the `Ground_01` material. Select the **Ocean_bed** object and assign the material.

Now let's move on to the more complex steps:

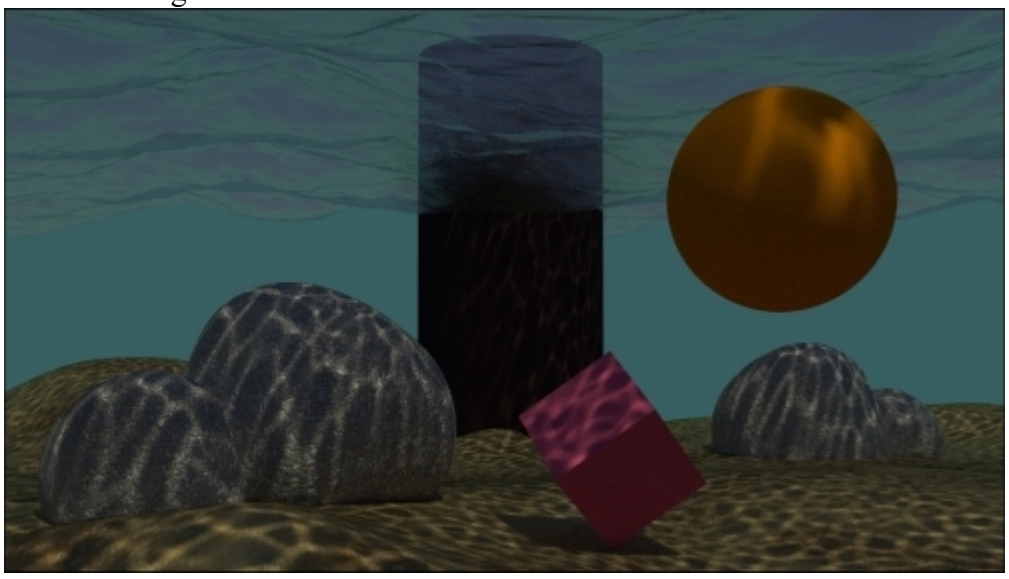
1. From the `9931OS_05_Ocean.blend` file, append the `Ocean_surface` object, material. Select the **Ocean_surface** object and assign the material. Rename it as `Ocean_surface_under`.

2. With the **Ocean_surface** object still selected, enter **Edit Mode**. Go to the **Face** selection mode and select only the upper faces. Then press *Ctrl* + *I* to invert the selection. In the **Material** window under the **Properties** panel, click on the **+** icon on the right (Add a new material slot), rename the new material as `Null`, and click on the **Assign** button. Now the **Ocean_surface** object has two different materials: the transparent water surface and the opaque sides and bottom (a simple white **Diffuse BSDF** material). Exit **Edit Mode**.
3. In the **Material** window, click on the **Ocean_surface_under** material to select it. In the **Node Editor** window, delete the **Foam** and the **Foam_location** node groups. Also delete the two **MixRGB** nodes. Just leave the **Ocean_water** node group connected to the second **Shader** input socket of the **Mix Shader** node, which in turn is connected to the **Material Output** node.
4. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate **Vector** | **Mapping**), and an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**). Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node, and the **Vector** output of this node to the **Vector** input socket of the **Image Texture** node.
5. In the **Image Texture** node, load the `caustics_tileable_low.png` texture and set the **Color Space** to **Non-Color Data**.
6. Add a **Diffuse BSDF**, a **Transparent BSDF**, and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**, and repeat the same for the other two nodes). Label them as `Diffuse_Caustics`, `Transparent_Caustics`, and `Mix Shader_Caustics`.
7. Connect the **Diffuse_Caustics** node's output to the first **Shader** input socket of the **Mix Shader_Caustics** node, and the **Transparent_Caustics** output to the second **Shader** input socket. Then connect the **Color** output of the **Image Texture** node to the **Color** input socket of the **Transparent_Caustics** shader node, and the **Alpha** output of the **Image Texture** node to the **Fac** input of the **Mix Shader_Caustics** node.
8. Now connect the output of the **Mix Shader_Caustics** node to the first (and still empty) **Shader** input socket of the first **Mix Shader** node.
9. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**). Connect the **Is Camera Ray** output to the **Fac** input of the first **Mix Shader** node. Add **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**) and parent these last nodes to it. Then label it as `FAKE_CAUSTICS` as shown in the following screenshot:



The FAKE_CAUSTICS frame mixed with the Ocean_Water node group on the ground of the Is Camera Ray output of the Light Path node

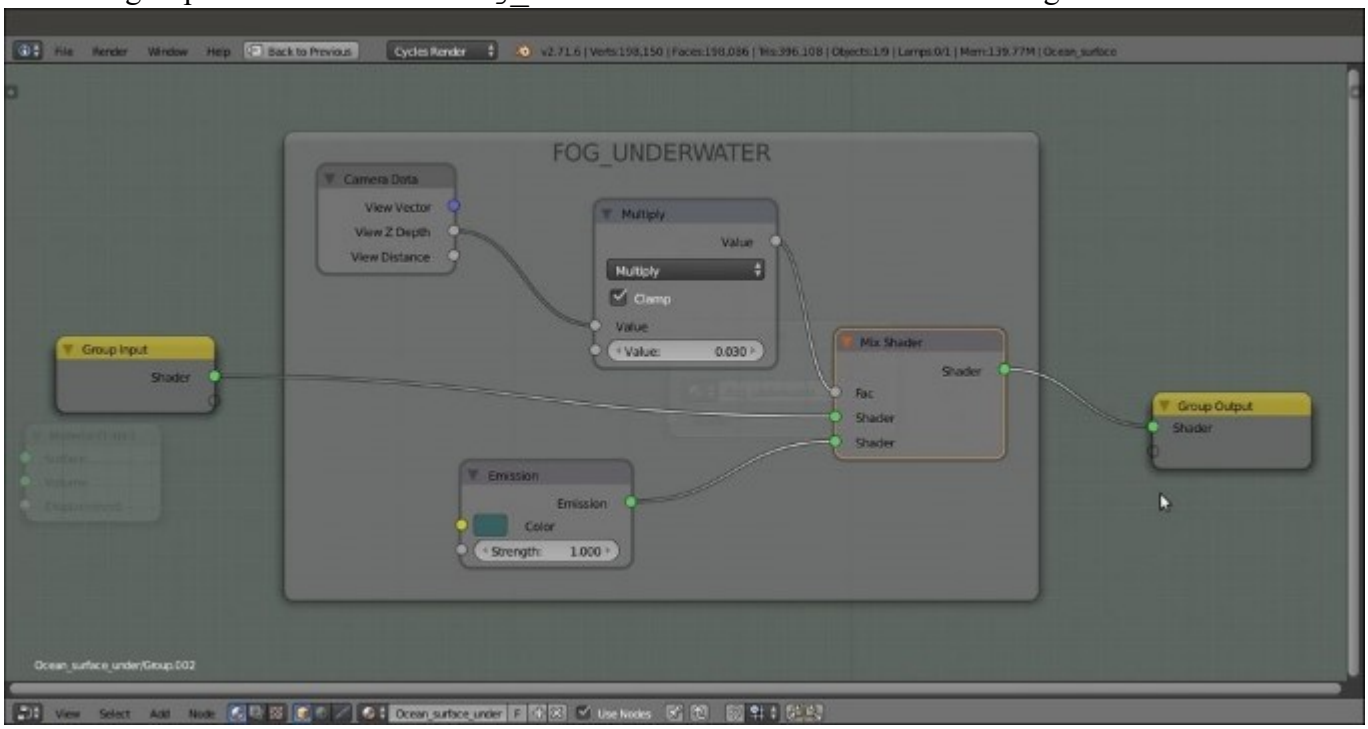
The following screenshot shows where we are so far:



The point where we are so far

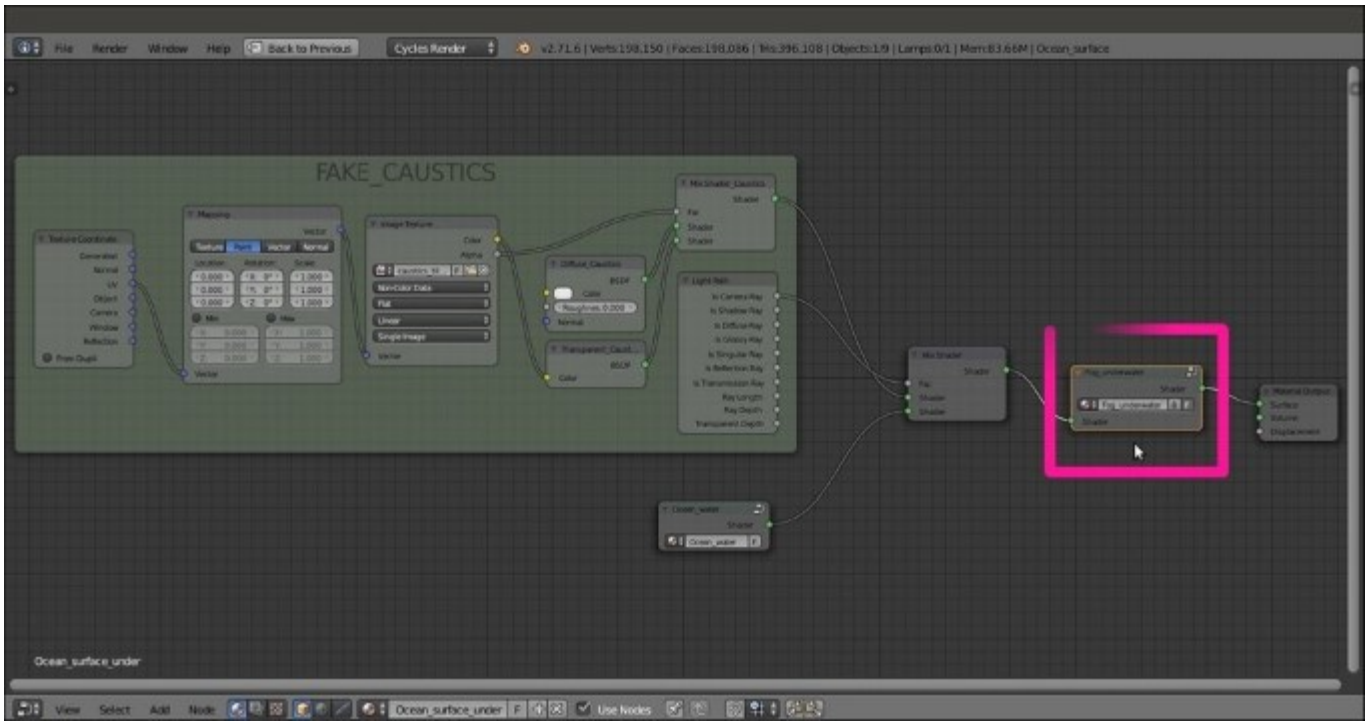
What is missing now is the underwater atmospheric perspective effect. There are several ways to obtain this, for example, by compositing a Mist pass rendered in Blender Internal or by using a volumetric shader. However, we are going to do this with a simple node group added to every one of the different materials.

10. Add a **Camera Data** node (press *Shift* + *A* and navigate to **Input** | **Camera Data**), a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), an **Emission** node (press *Shift* + *A* and navigate to **Shader** | **Emission**), and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**). Label the **Mix Shader** node as `Mix Shader_Fog`.
11. Connect the **View Z Depth** output of the **Camera Data** node to the first **Value** input of the **Math** node. Set the **Math** node's **Operation** to **Multiply** and the second **Value** to `0.030`. Check the **Clamp** option. Connect the **Multiply** node output to the **Fac** input socket of the **Mix Shader_Fog** node.
12. Connect the **Emission** output to the second **Shader** input of the **Mix Shader_Fog** node. Set the **Color** values for **R** to `0.040`, **G** to `0.117`, and **B** to `0.124`.
13. Select all the new nodes and press *Ctrl* + *G* to make a group. Click and drag the first **Shader** input socket of the **Mix Shader_Fog** node into the empty socket of the **Group Input** node on the left and repeat this step by connecting the **Shader** output socket on the right. Press *Tab* to close the group. Then rename it as `Fog_underwater` as shown in the following screenshot:



The FOG_UNDERWATER node group in Edit Mode

14. Add and paste the **Fog_underwater** node (press *Shift* + *A* and navigate to **Group** | **Fog_underwater**) just before the **Material Output** node of every material (in our scene, the **Fog_underwater** node will show eight users if the *fake user* button is selected) as shown in the following screenshot:



The Fog_underwater node group pasted at the end of the shader

How it works...

First of all, why should we choose a Cube for the ocean surface instead of the simpler Plane?

The reason is very simple: in Cycles, the World emits light, and the only way to avoid this is to set the color to pitch black (or by a combination of a **Light Path** node with the **World** materials, but this is another story). In our scene, the World is set to a bright blue sky color, and with a Plane, the underwater objects and the ocean bed would have been lit too much from the sides and the bottom the result look natural. A Cube, on the other hand, envelops all the underwater elements, limiting the lighting to the Sun Lamp passing through the surface, and projecting the image textured caustics. This gives a more natural-looking result.

The image texture we assigned to the water material is used to obtain a textured transparency effect. Right now, the water surface is actually opaque and transparent according to the black and white values of the textures, so as to allow the Sun Lamp light to pass through and project the caustics.

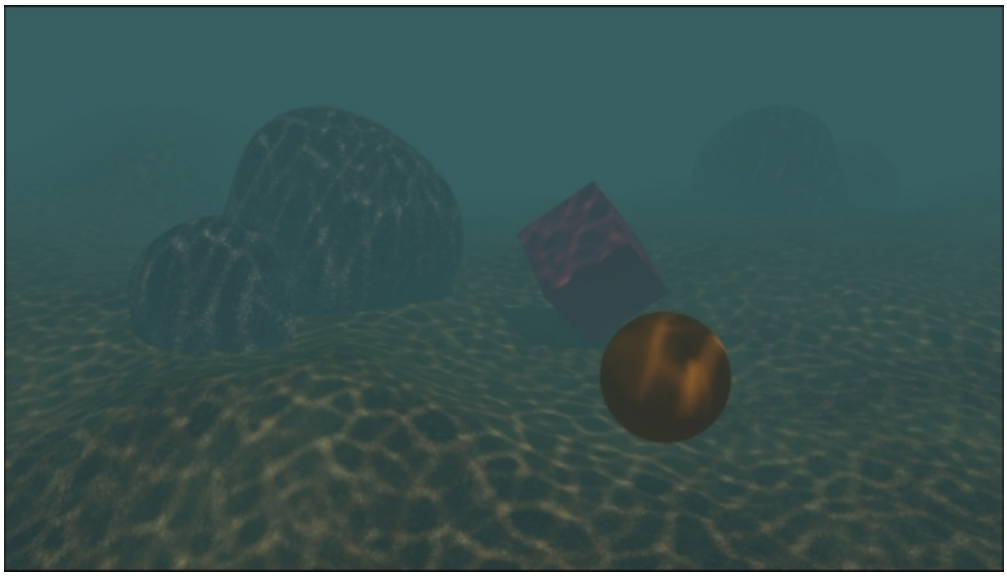
Thanks to the **Is Camera Ray** output of the **Light Path** node, the caustics image texture is not directly renderable on the ocean surface, but it nevertheless has some effect on the other materials. Because the value of **Is Camera Ray** is `1`, the rays starting from the Camera and directly hitting the ocean surface can render only the clean water material plugged into the second input socket of the **Mix Shader** node, while the transmitted caustics (plugged in the first **Shader** socket = 0) get rendered.

The **Fog_underwater** node group is simply an emitter material serving as the background (deep green in this case) and mapped on every underwater material according to the *z* depth of the Camera (it also works with the Camera frame, in the viewport). The density of the fog is set by the **Multiply** node's second **Value**. For the ocean body, a value of `0.030` is good enough.

Note

The Camera *z* axis must not be confused with the global coordinate *z* axis, which is the vertical blue line visible in the 3D view. The Camera *z* axis, on the other hand, is the ideal line connecting the starting point of view to any visible element in the scene.

Note that we didn't expose the values of the nodes in the **Fog_underwater** group. This is because in **Edit Mode**, we can tweak the internal values of just one node to automatically update all the fog group instances assigned to the other materials. Besides, we know that the values exposed on the group interface would overwrite the internal settings and work only for that single node instance.



The final underwater environment rendered from a different point of view

Creating a snowy mountain landscape with procedurals

In this recipe, we will make a snowy mountain landscape by reusing existing shaders—the `Rock_procedural` and the `Snow materials`. We will improve these materials by grouping them and exposing the useful values. Then we will create a new group node that will work as a stencil to depict snow in a more customizable and natural way on the rocks as shown in the following screenshot:

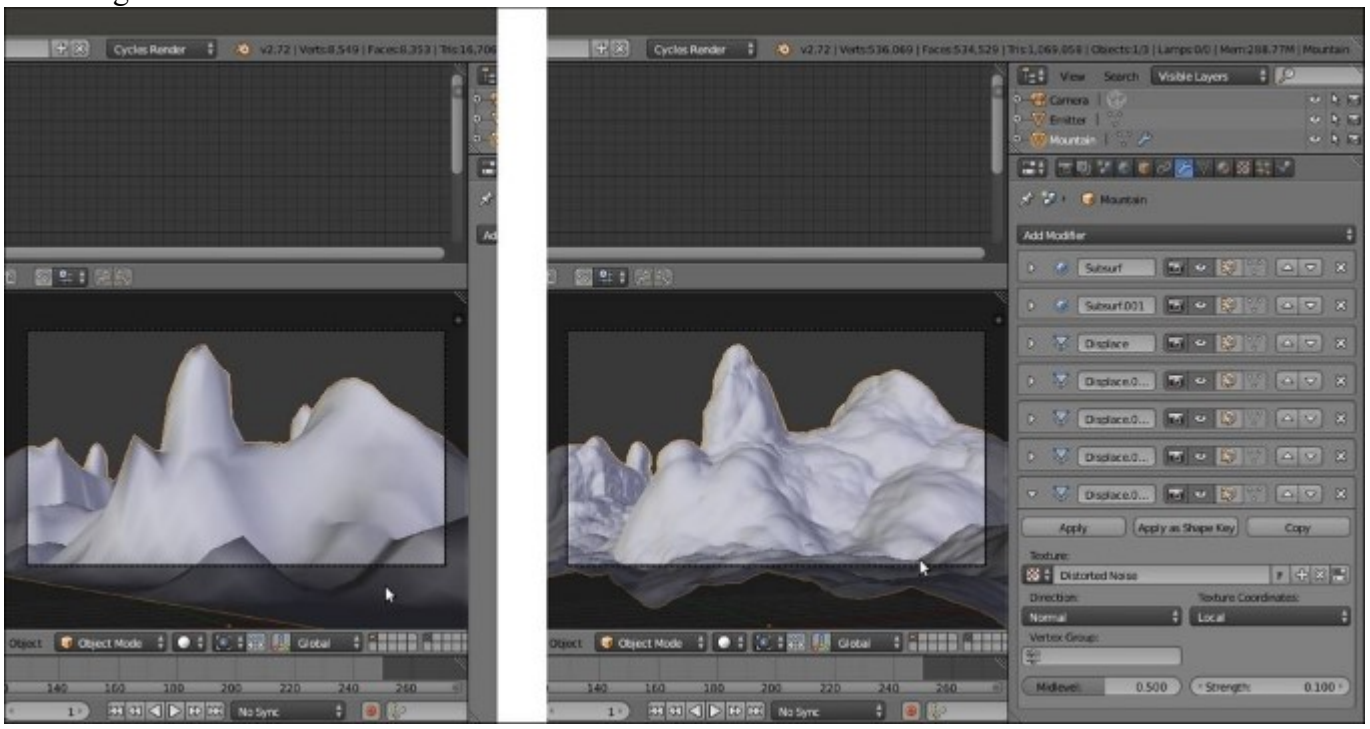
The snowy mountain landscape as it appears in the final rendering

Getting ready

As usual, let's start with the preparation of the scene. In this case, we start with an almost ready blend file:

1. Start Blender and open the `9931OS_05_RockSnow_start.blend` file, where there is a scene with a placed Camera—a simply modeled **Mountain** object and a **Plane** set as **Emitter**.
2. Select the **Mountain** object, go to the **Object modifiers** window, and assign a **Subdivision Surface** modifier. Set the levels to `2` for both **View** and **Render**.
3. Assign a second **Subdivision Surface** modifier. Set the levels to `1` for both **View** and **Render**.
4. Assign a **Displace** modifier. Click on the **Show texture in texture** tab to the extreme right of the **Texture** name slot to go to the **Textures** window. Assign a **Voronoi** procedural texture. Set the **Size** to `1.00`. Go back to the **Displace** modifier and set the **Strength** to `-0.200`.
5. Assign a second **Displace** modifier. In the **Texture** window, assign a new **Voronoi Texture** and set **Distance Metric** to **Manhattan** and **Size** to `0.50`. Back in the **Displace** modifier panel, set the **Strength** to `-0.050`.

6. Assign a third **Displace** modifier, select a **Clouds** texture, and set **Noise** to **Hard** and the **Displace** modifier's **Strength** to `0.040`.
7. Assign a fourth **Displace** modifier. In the **Texture** window, assign a **Musgrave** procedural texture. Set the **Type** to **Hetero Terrain**, **Dimension** to `0.650`, **Lacunarity** to `2.000`, **Octaves** to `0.500`, **Offset** to `0.250`, **Basis** to **Voronoi F1**, and **Size** to `2.00`. Back in the **Displace** modifier panel, set the **Strength** to `0.300`.
8. Assign a fifth **Displace** modifier. In the **Texture** window, assign a **Distorted Noise** texture. Set the **Noise Distortion** to **Voronoi F1**, Basis to **Improved Perlin**, **Distortion** to `2.000`, and **Size** to `3.30`. Back in the **Displace** modifier panel, set the **Strength** to `0.100` as shown in the following screenshot:

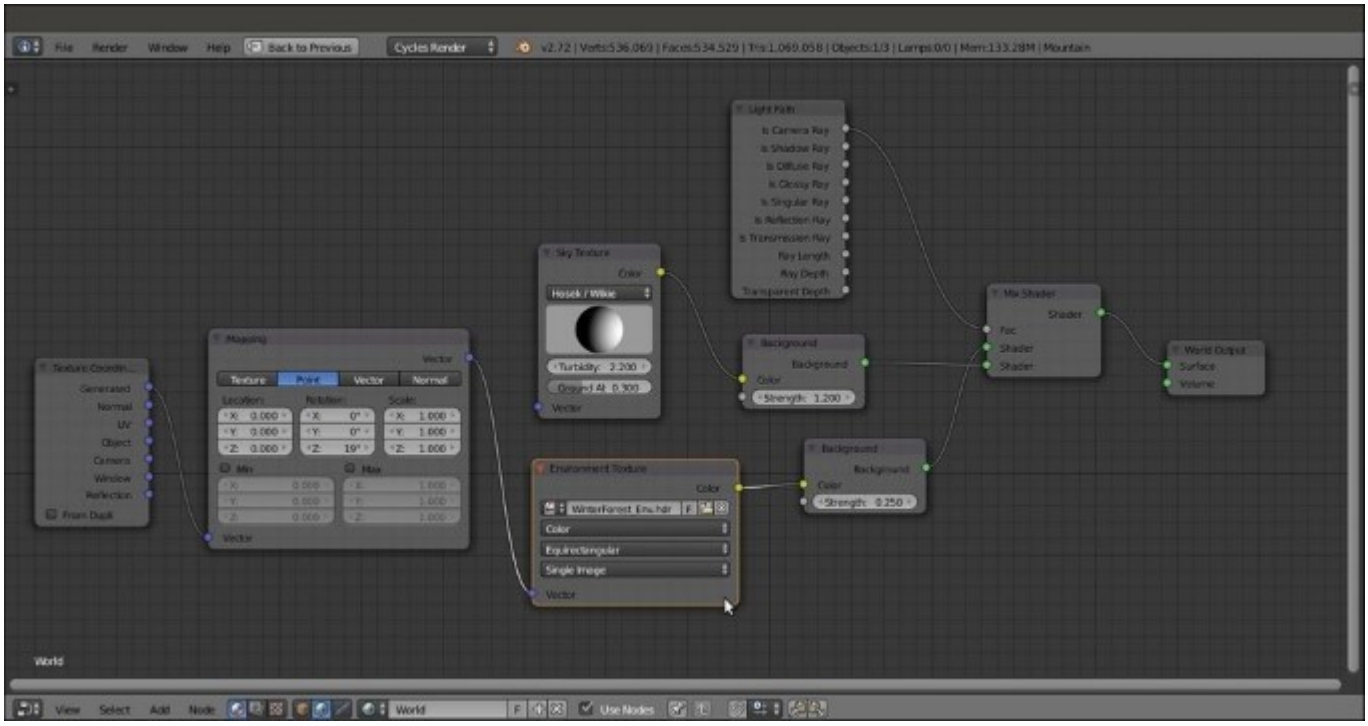


The mountain object obtained using different settings—without and with the several modifiers

9. Now disable the **Display** modifier in viewport button (the eye icon) of each modifier.
10. Go to the **World** window and click on the **Use Nodes** button. Then click on the little square with a dot on the right side of the **Color** slot. From the pop-up menu, select **Sky Texture**. On the **Background** node, set the **Strength** value to `1.200`.
11. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Background** and the **World Output** nodes. Switch the link of the **Background** node with the second input socket.
12. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), an **Environment Texture** node (press *Shift* + *A* and navigate to **Texture** | **Environment Texture**), and a new **Background** node (press *Shift* + *A* and navigate to **Shader** | **Background**).
13. Connect the **Generated** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of this node to the **Vector** input socket of the **Environment**

Texture node. Connect the **Color** output of this node to the **Color** input socket of the second **Background** node.

14. Connect the output of the second **Background** node to the first input socket of the **Mix Shader** node, and set its **Strength** to `0.250`. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**). Connect the **Is Camera Ray** output to the **Fac** input socket of the **Mix Shader** node.
15. Go to the **Environment Texture** node and click on the **Open** button. Browse to the `texture` folder and load the `WinterForest_Env.hdr` image (it's a free, high-dynamic-range image downloaded from the sIBL Archive at http://www.hdrlabs.com/sibl/archive.html, and licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License).
16. Go to the **Mapping** node and set the **Rotation** value of Z to `19°` as shown in the following screenshot:



The World network setting

17. Go to the **Render** window, and under the **Sampling** subpanel, set both the **Clamp Direct** and **Clamp Indirect** values to `1.00`. Set the **Samples** to `10` for **Preview** and `25` for **Render**. Under the **Light Paths** subpanel, disable both the **Reflective Caustics** and **Refractive Caustics** items and set the **Filter Glossy** to `1.00`.

How to do it...

We are going to create the scene and materials by dividing the process into four stages:

- Appending and grouping rock and snow shaders
- Mixing the material groups

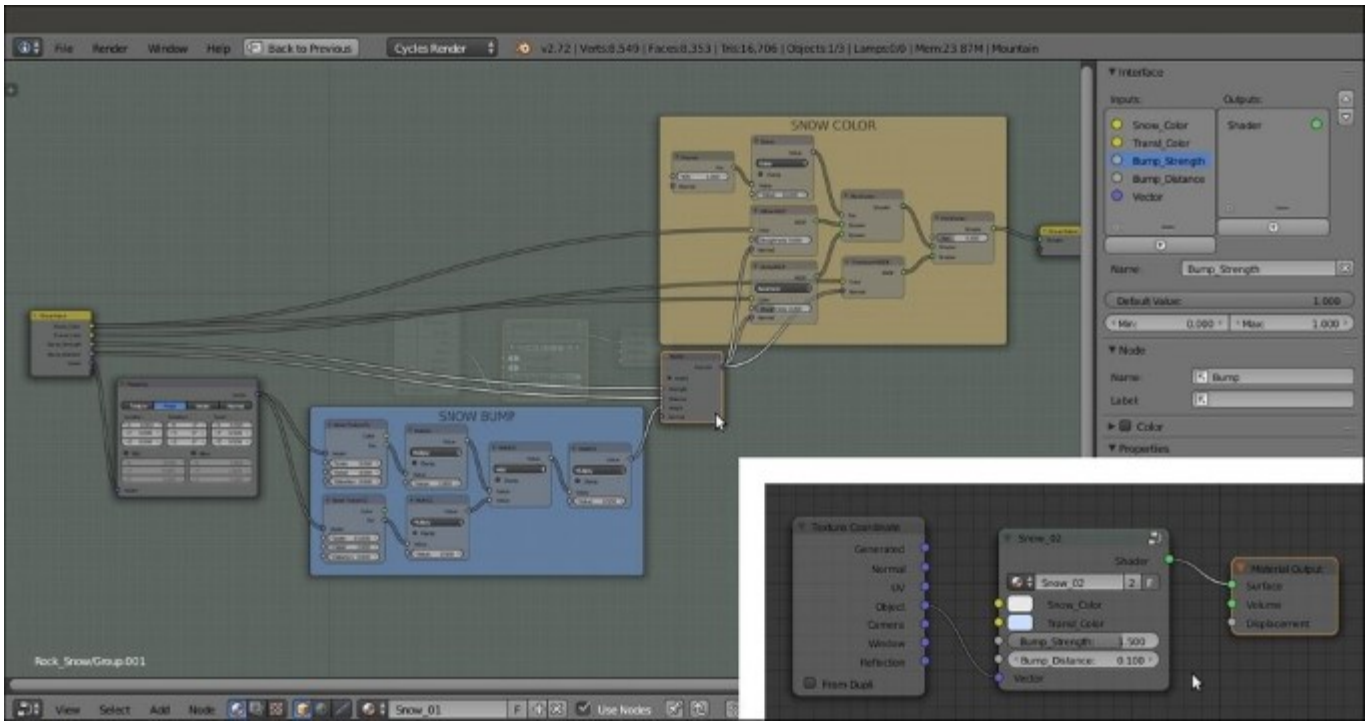
- Creating a stencil shader
- Adding an atmospheric perspective

So, let's start with the first stage.

Appending and grouping the rock and the snow shader

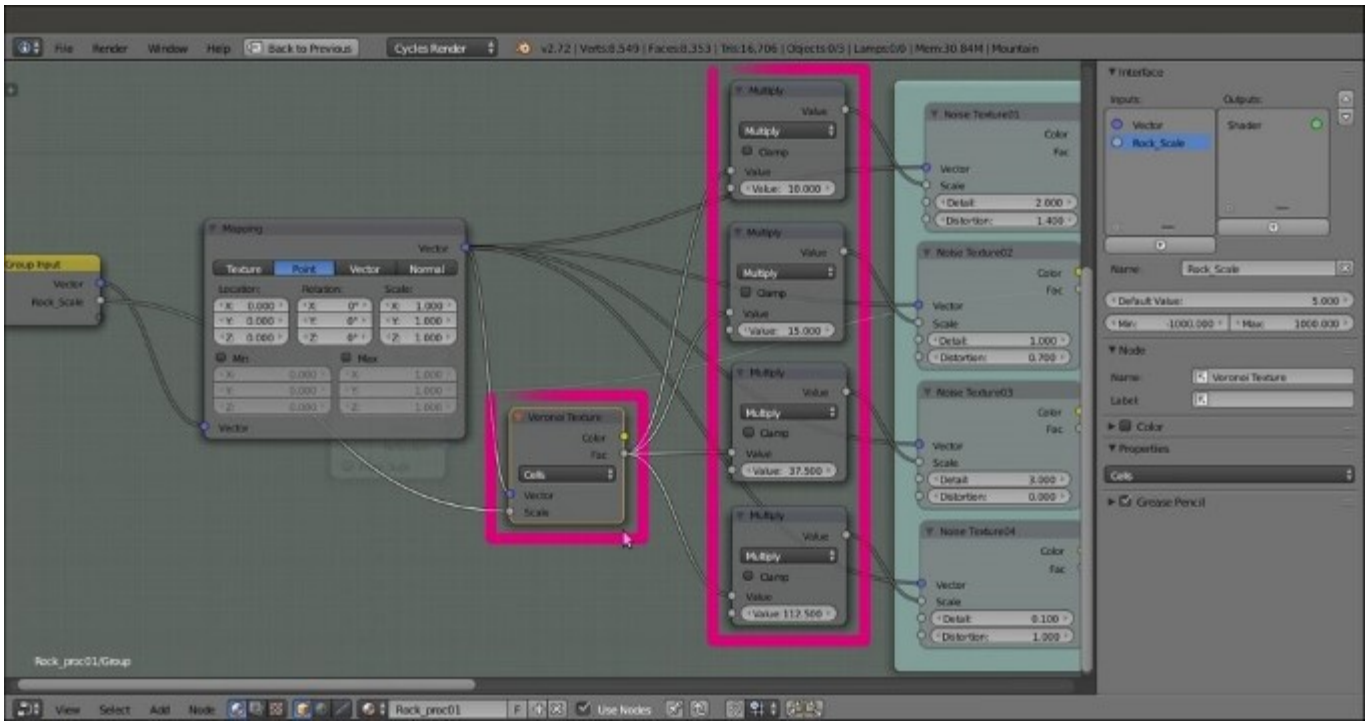
Let's append the required (and previously made) materials, and group them for convenience:

1. From the `9931OS_03_snow.blend` file, append the `Snow_01` material, and for now, assign it to the **Mountain** object.
2. In the **Node Editor** window, select all the nodes except **Texture Coordinates** and **Material Output**, and press *Ctrl* + *G* to group them.
3. Place the **SNOW_COLOR** frame to the right of the **SNOW_BUMP** frame. Select the **Group Output** node, and in the (press *N*) **Properties** panel of the **Node Editor** window, delete the **Value** output.
4. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Value** output of the **Math04** node inside the **SNOW_BUMP** frame to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of the **Bump** node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF** shaders inside the **SNOW_COLOR** frame.
5. Click on the **Strength** socket of the **Bump** node and drag it into the empty socket on the **Group Input** node. Rename the socket (automatically named Strength) as `Bump_Strength`. Repeat this step for the **Distance** socket and rename it as `Bump_Distance`.
6. Drag the **Color** input socket of the **Diffuse BSDF** node into the empty socket of the **Group Input** node. Move the new socket to the top of the list and rename it as `Snow_Color`. Drag the **Color** input of the **Glossy BSDF** shader node and connect it to the same **Snow_Color** socket.
7. Drag the **Color** input socket of the **Translucent BSDF** node into the empty socket of the **Group Input** node. Move the new socket upwards, just below the **Snow_Color** socket, and rename it as `Transl_Color`.
8. Move the **Vector** socket on the **Group Input** node to the bottom of the list, and close the group. Rename it as `Snow_02` and check the *fake user* option. Click on the **Bump_Strength** slider and type `1.500` as shown in the following screenshot:



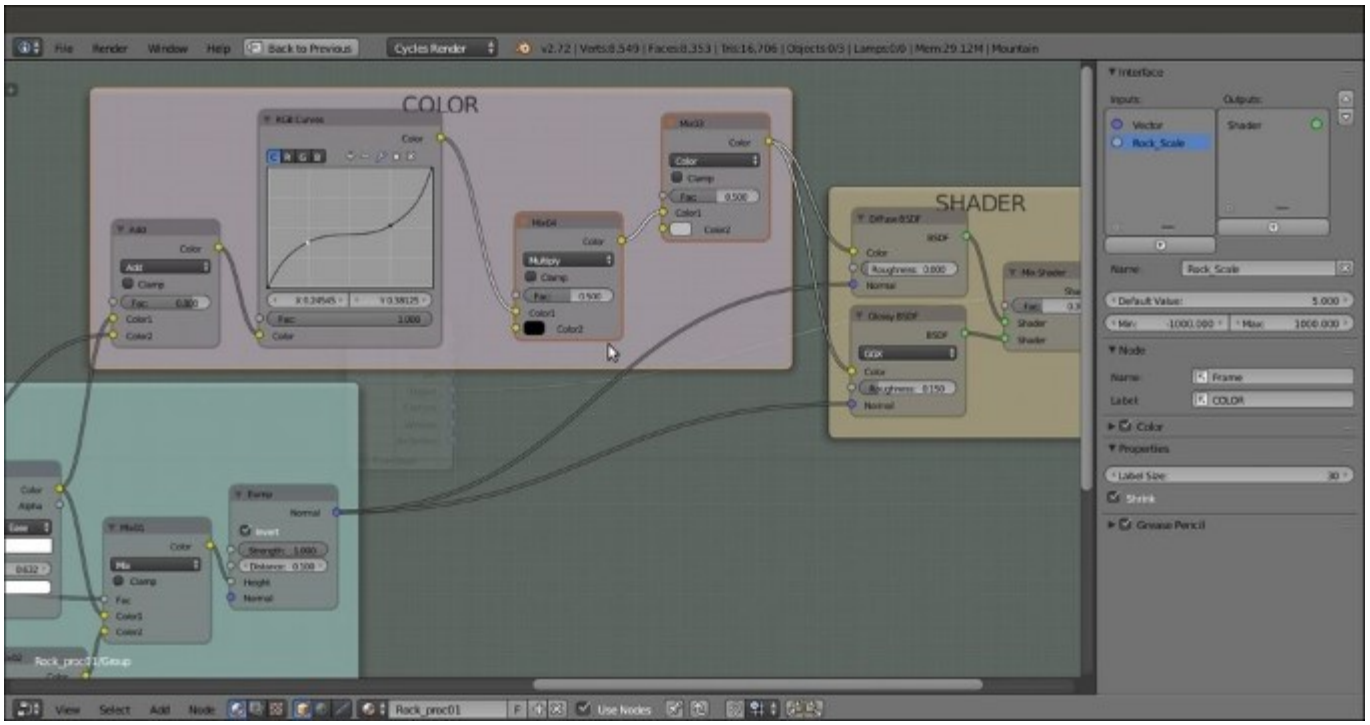
Making a node group of the appended snow material

9. From the `9931OS_03_Rock_procedurals.blend` file, append the `Rock_proc01` material. Go to the **Material** datablock button on the **Node Editor** toolbar and assign it to the **Mountain** object.
10. In the **Node Editor** window, select all the nodes except the **Texture Coordinates** and the **Material Output** nodes. Press *Ctrl* + *G* to group them. Set the **Location** values in the **Mapping** node to `0.000` for all the three axes.
11. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set the **Operation** to **Multiply** and the first **Value** to `1.000`. Press *Shift* + *D* to duplicate it, and do this three times. Connect the **Value** outputs of each of the four **Multiply-Math** nodes to the **Scale** input sockets of the four **Noise Texture** nodes inside the **BUMP** frame.
12. Now, in the second **Value** slot of each **Multiply-Math** node, set `10.000` for **Noise Texture01**, `15.000` for **Noise Texture02**, `37.500` for **Noise Texture03**, and `112.500` for **Noise Texture04**.
13. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**), switch the **Coloring** to **Cells**, and connect its **Fac** output to the first **Value** input socket (the socket with value of `1.000`) of each of the four **Multiply-Math** nodes.
14. Connect the **Voronoi Texture** node's **Vector** input socket to the **Vector** output of the **Mapping** node and drag the link from its **Scale** input socket to the empty socket of the **Group Input** node. Rename the new socket as `Rock_Scale` as shown in the following screenshot:



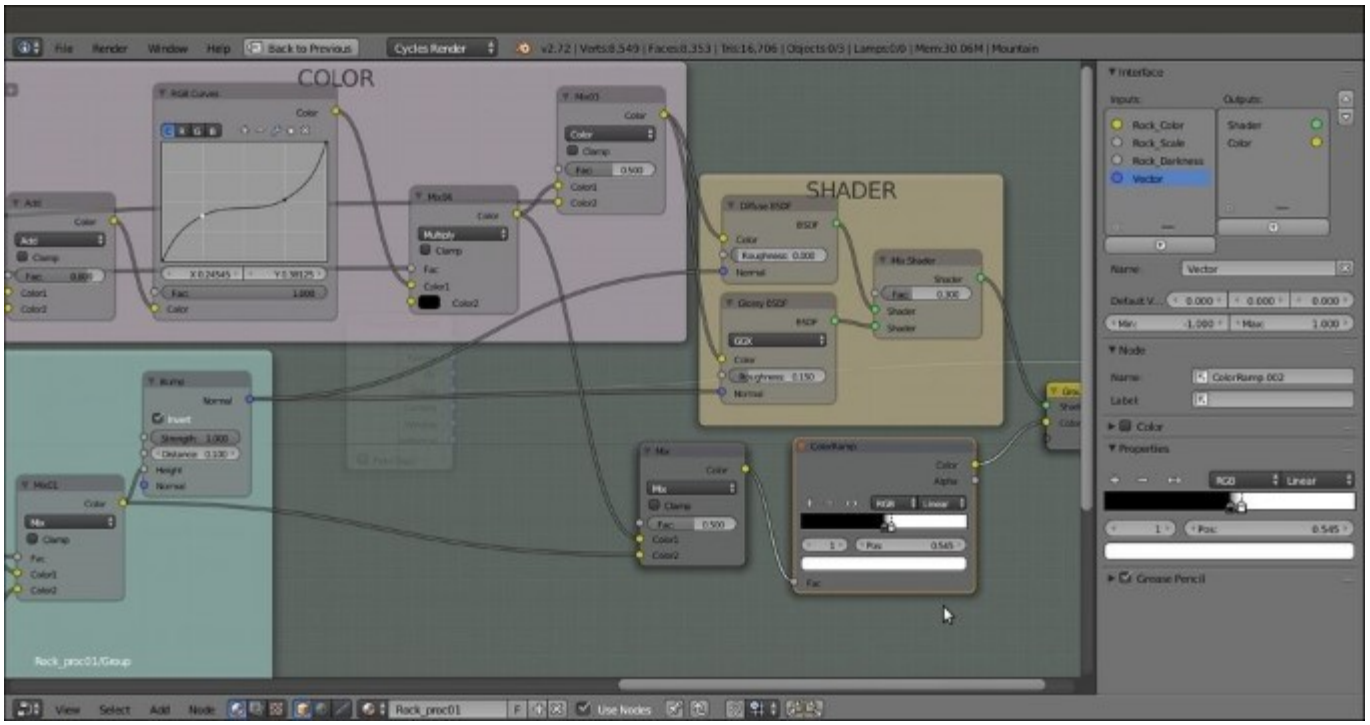
Adding Math nodes to tweak the exposed scale values of the textures for the procedural rock material

15. Now go to the **COLOR** frame and delete the **RGB** node. Then select the **Mix03** node and press *Ctrl* + *X* to delete it, maintaining the connection of the **Darken** node to the **Color2** socket of the **Add** node.
16. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), label it as `Mix03` again, and paste it between the **RGB Curves** node and the **Diffuse BSDF** shader. Set the **Blend Type** to **Color** and connect its output to the **Color** input socket of the **Glossy BSDF** shader.
17. Press *Shift* + *D* to duplicate the **Mix03** node. Label the duplicate as `Mix04` and paste it between the **RGB Curves** and the **Mix03** nodes. Set its **Blend Type** to **Multiply** and the **Color2** to pure black. Then select both the **Mix03** and **Mix04** nodes and parent them to the **COLOR** frame as shown in the following screenshot:



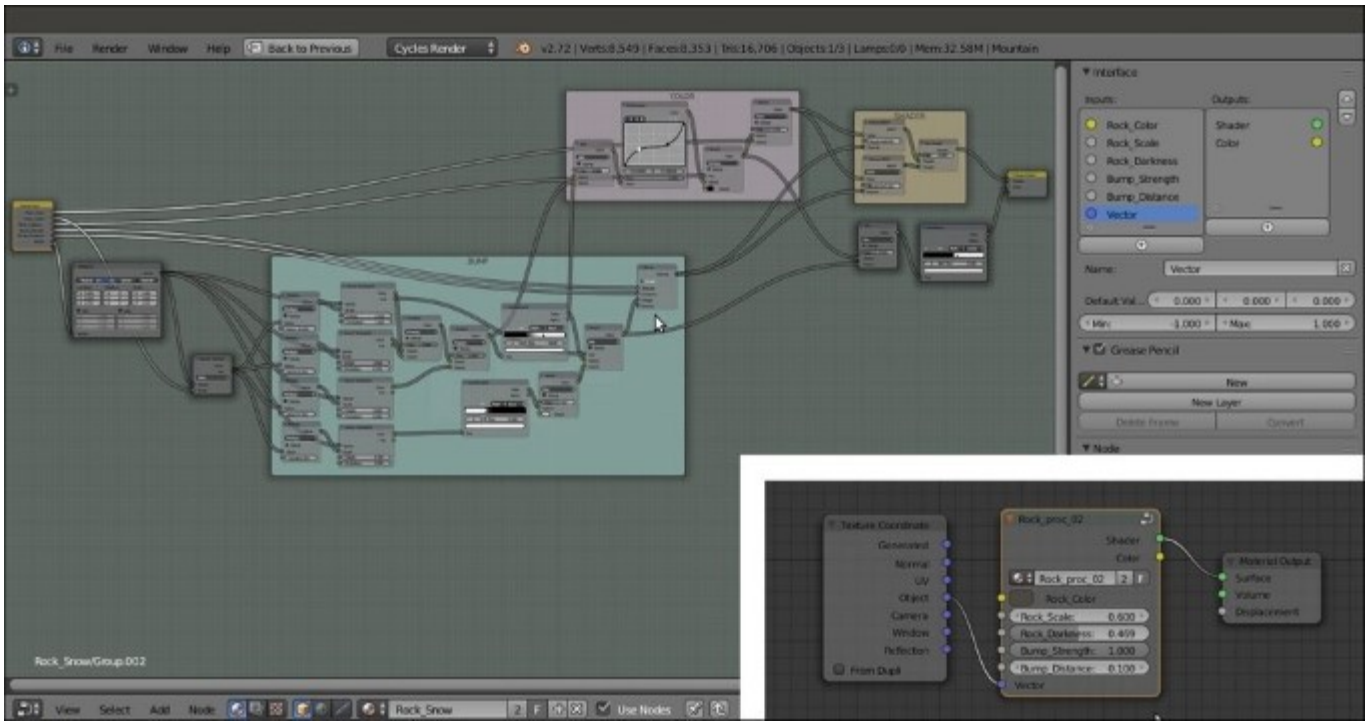
Adding color variations to the rock material

18. Click on the **Color2** input socket of the **Mix03** node. Drag it to the empty socket of the **Group Input** node. Move the new socket to the top of the list and rename it as `Rock_Color`.
19. Repeat the preceding step with the **Fac** socket of the **Mix04** node, and rename the new socket as `Rock_Darkness`. Move the **Vector** socket on the **Group Input** node to the bottom of the list.
20. Add a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Color** output of the **Mix03** node to the **Color1** input socket of the new **MixRGB** node, and the **Color** output of **Mix01** to the **Color2** input socket.
21. Connect the **Color** output of the new **MixRGB** node to the **Fac** input socket of the **ColorRamp** node. Then connect the **Color** output of this node to the empty socket of the **Group Output** node. Go to **ColorRamp** and set the position of the black color stop to `0.500` and the position of the white color stop to `0.545` as shown in the following screenshot:



Creating a Color output in the node group to be used later to detail the stencil effect

22. Drag the **Strength** and the **Distance** sockets of the **Bump** node to the **Group Input** node, and rename them as `Bump_Strength` and `Bump_Distance`, respectively. Move the **Vector** socket to the bottom. Press *Tab* to exit **Edit Mode**.
23. Rename the group as `Rock_proc_02`, and enable the *fake user*. Set the **Rock_Color** values for **R** `0.078`, **G** to `0.067`, and **B** to `0.056`; the **Rock_Scale** to `0.600`; and the **Rock_Darkness** to `0.469` as shown in the following screenshot:

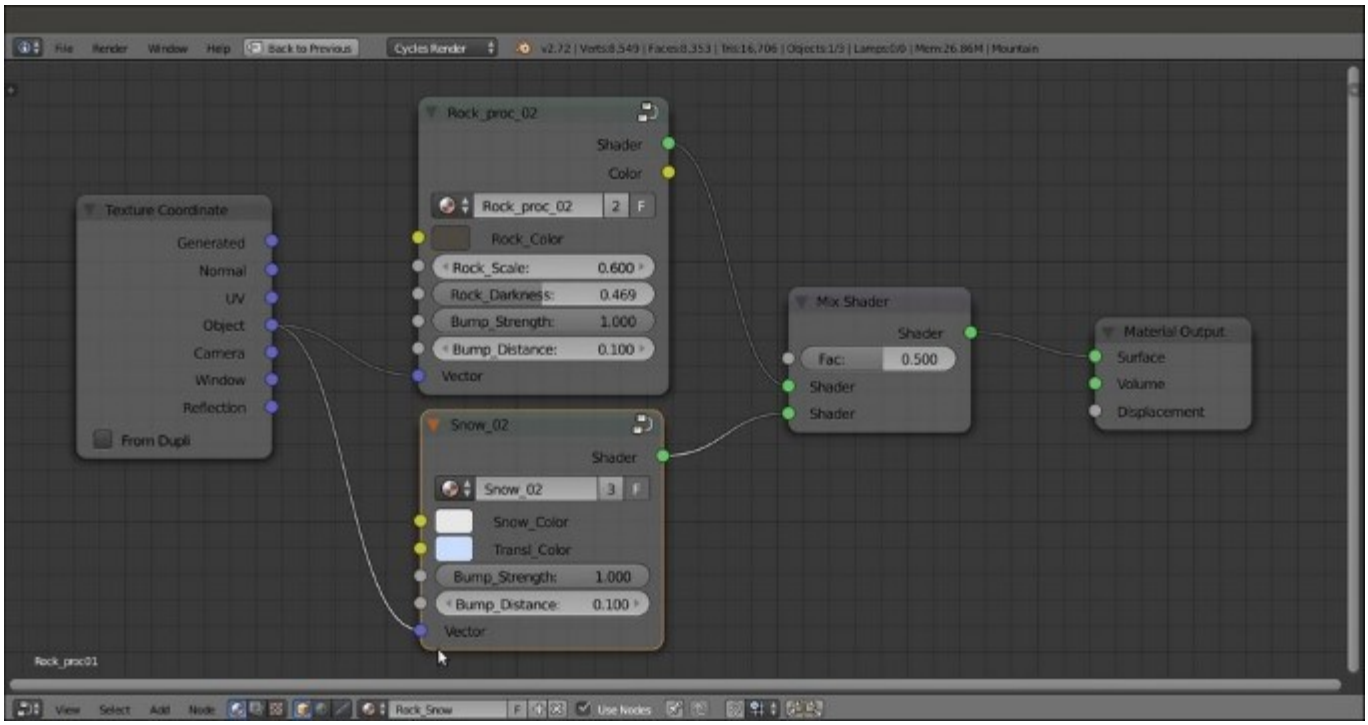


The overall view of the procedural rock node group in Edit Mode

Mixing the material groups

Now we can start to build the real shader by mixing the procedural rock and snow materials:

1. Press *Shift* + *A* with the mouse in the **Node Editor** window and add the **Snow_02** group node (press *Shift* + *A* and navigate to **Group** | **Snow_02**). Then rename the material as `Rock_Snow` in the **Node Editor** toolbar.
2. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Rock_proc_02** group node and the **Material Output** node. Connect the **Shader** output of the **Snow_02** group node to the second **Shader** input socket of the **Mix Shader** node.
3. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Snow_02** group node as shown in the following screenshot:



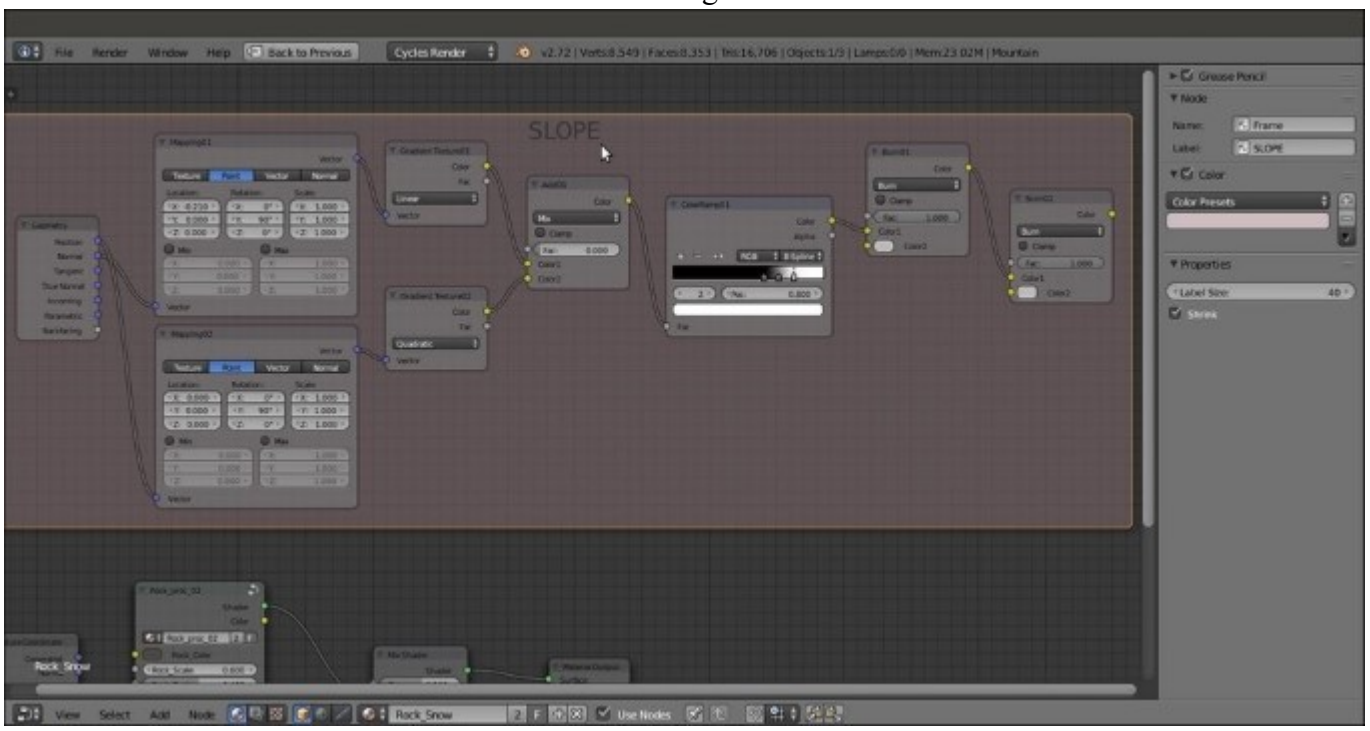
Starting to build the snowy rock mountain material

Creating the stencil shader

At this point, both the materials are assigned to the **Mountain** object, but if you render the preview now, they will appear on the whole mesh surface as a mixture of rock and snow. We must build a separator to establish where the surface will show only the rock and where it will show only the snow:

1. Add a **Geometry** node (press *Shift* + *A* and navigate to **Input** | **Geometry**), two **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**), two **Gradient Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Gradient Texture**), and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**).
2. In the **Properties** panel of the **Node Editor**, label these four nodes as follows: `Mapping01`, `Mapping02`, `Gradient Texture01`, `Gradient Texture02`, and `ColorRamp01`. Connect the **Normal** output of the **Geometry** node to the **Vector** input socket of the **Mapping01** node and the **Position** output to the **Mapping02** node. Then connect the **Mapping01** node to the **Gradient Texture01** node and the **Mapping02** node to the **Gradient Texture02** node.
3. Leave the **Gradient Type** of the **Gradient Texture01** node as **Linear** and set the **Gradient Type** of the **Gradient Texture02** to **Quadratic**. In the **Mapping01** node, set the **Location** value of **X** as `-0.210` and the **Rotation** value of **Y** as `90°`. In the **Mapping02** node, set only the **Rotation** value of **Y** as `90°`.
4. Add three **MixRGB** nodes (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Fac** of the first one to `0.000` and label it as `Add01`. Then connect the **Color** outputs of both the **Gradient Texture** nodes to the **Color1** and to the **Color2** input sockets.

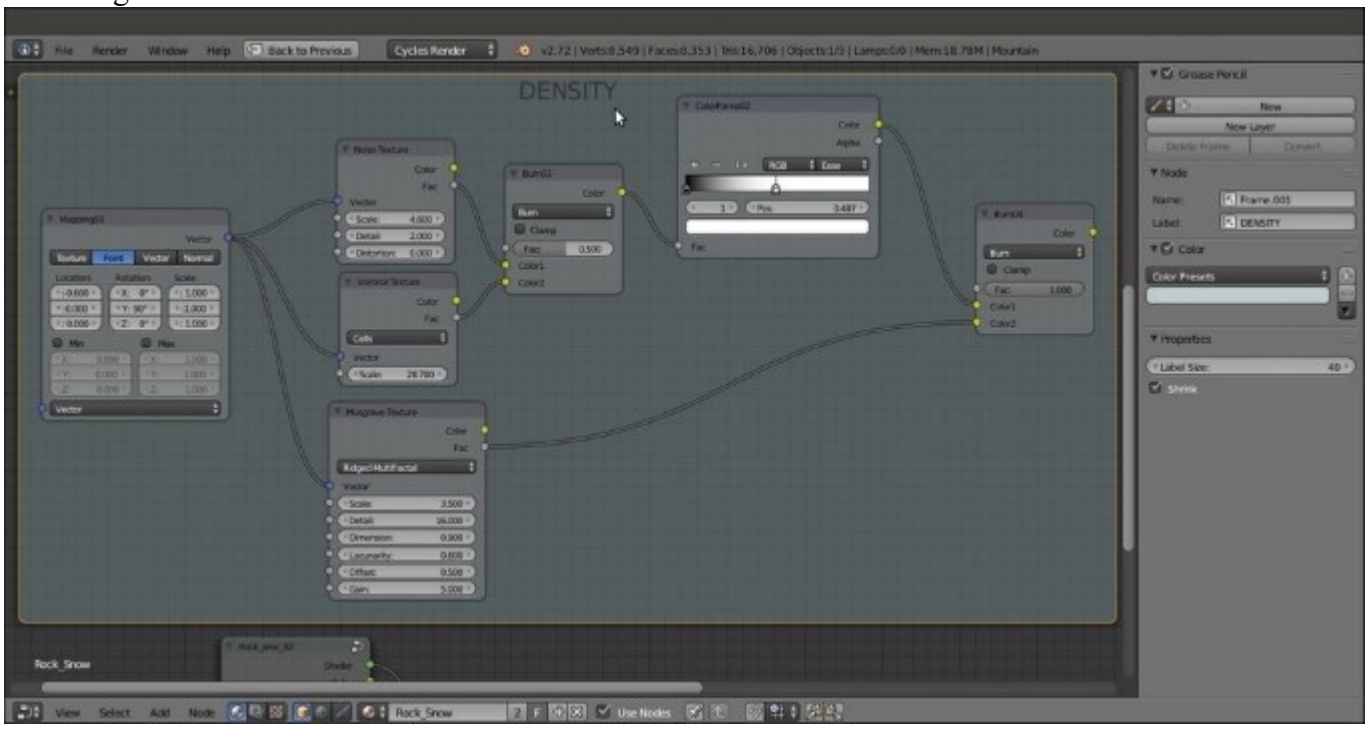
5. Connect the output of the **Add01** node to the **Fac** input socket of the **ColorRamp01** node. Then set its **Interpolation** to **B-Spline** and move the black color stop to `0.600` position. Add a new color stop, set the color to pure black, and move it to the `0.700` position. Then move the position of the white color stop to `0.800`.
6. Label the other two **MixRGB** nodes as **Burn01** and **Burn02**. Connect the **Color** output of the **ColorRamp01** node to the **Color1** input socket of the **Burn01** node, and the **Color** output of this node to the **Color1** input of the **Burn02** node. Set the **Blend Type** for both the nodes to **Burn**.
7. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**) and parent all of these nodes to it. Label the frame as `SLOPE` as shown in the following screenshot:



The SLOPE frame

8. Now press *Shift* + *D* to duplicate one of the **Mapping** nodes (press *Alt* + *P* to unparent it from the frame), and move it to under the **SLOPE** frame. Label it as `Mapping03` and change the **Location** value of **X** to `-0.600`.
9. Add a **Noise Texture**, a **Voronoi Texture**, and a **Musgrave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**, repeat the same for all other nodes) and place them in a column next to the **Mapping03** node. Set the **Noise Texture** node's **Scale** value to `4.600`. Set the **Voronoi Texture** node's **Coloring** to **Cell** and the **Scale** to `28.700`. Set the **Musgrave Texture** node's type to **Ridged Multifractal**, the **Scale** to `3.500`, **Detail** to `16.000`, **Dimension** to `0.900`, **Lacunarity** to `0.600`, **Offset** to `0.500`, and **Gain** to `5.000`.
10. Connect the **Vector** output of the **Mapping03** node to the **Vector** inputs of the three textures. Then add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Burn**, and label it as `Burn03`.

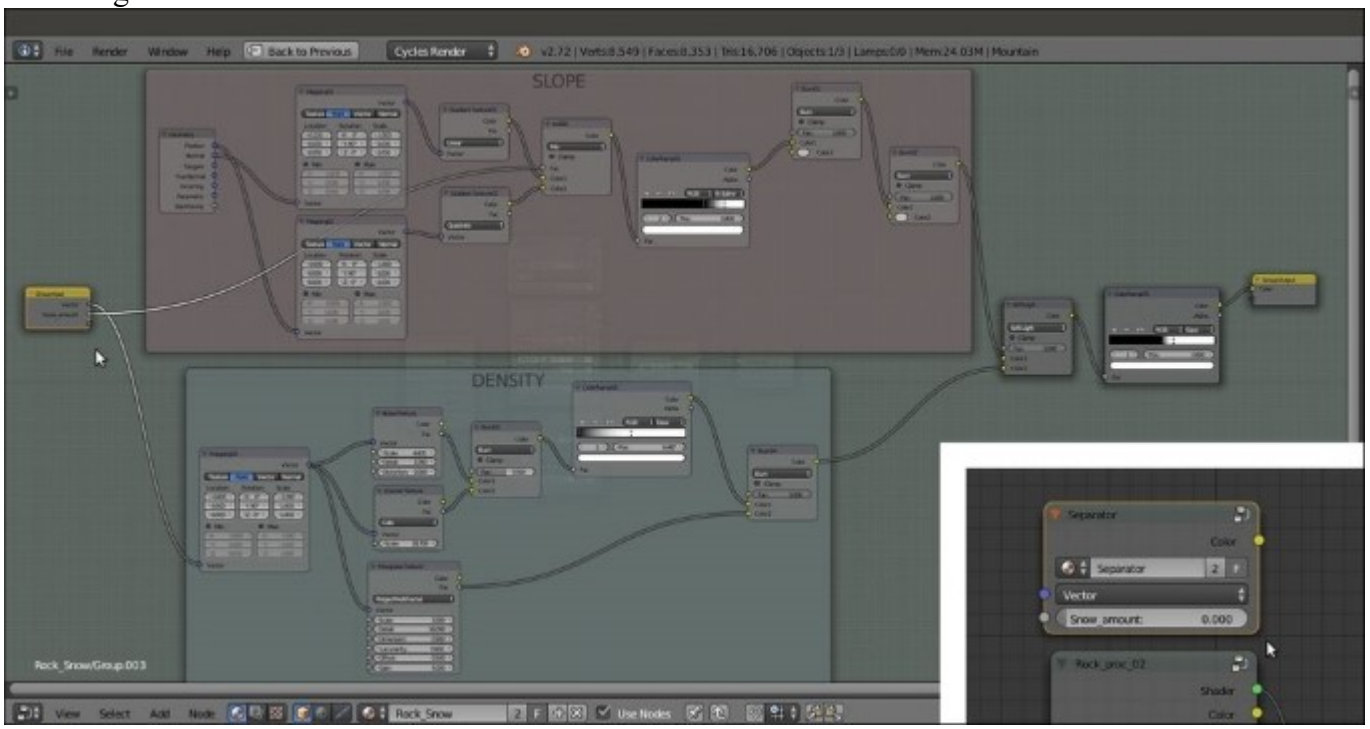
11. Connect the **Fac** outputs of the **Noise Texture** and **Voronoi Texture** nodes to the **Color1** and **Color2** input sockets of the **Burn03** node. Press *Shift* + *D* to duplicate the **Burn03** node, label it as `Burn04`, and set its **Fac** value to `1.000`. Connect the **Color** output of the **Burn03** node to the **Color1** input socket of the **Burn04** node. Then connect the **Fac** output of the **Musgrave Texture** node to its **Color2** input socket.
12. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp02`, and paste it between the **Burn03** and the **Burn04** nodes. Set **Interpolation** to **Ease** and move the white color marker to the `0.487` position.
13. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**), select all of these nodes and then the frame, and press *Ctrl* + *P* to parent them. Label the frame as `DENSITY` as shown in the following screenshot:



The DENSITY frame

14. Box-select the two frames (with all the nodes inside) and press *Ctrl* + *G* to create a group. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Soft Light**, and set the **Fac** value to `1.000`. Connect the **Color** output of the **Burn02** node inside the **SLOPE** frame to the **Color1** input socket. Then connect the **Color** output of the last **Burn04** node inside the **DENSITY** frame to the **Color2** input socket.
15. Drag the **Color** output of the **Soft Light** node into the empty socket of the **Group Output** node to create a new **Color** output on the interface. Then add a new **ColorRamp** (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp04`, and paste it between the **Soft Light** and the **Group Output** nodes. Set **Interpolation** to **Ease**, the black color stop to the `0.500` position, and the white color stop to the `0.600` position.
16. Go to the **SLOPE** frame. Click and drag the **Fac** socket of the **Add01** node to the empty socket of the **Input Group** node. Rename the new input as `Snow_amount`.

17. Go to the **DENSITY** frame and attach the **Vector** input of the **Mapping03** node to the empty socket of the **Input Group** node. Move it up by clicking on the little arrow icon in the **Properties** panel, and press *Tab* to close the group. Rename it as `Separator` as shown in the following screenshot:

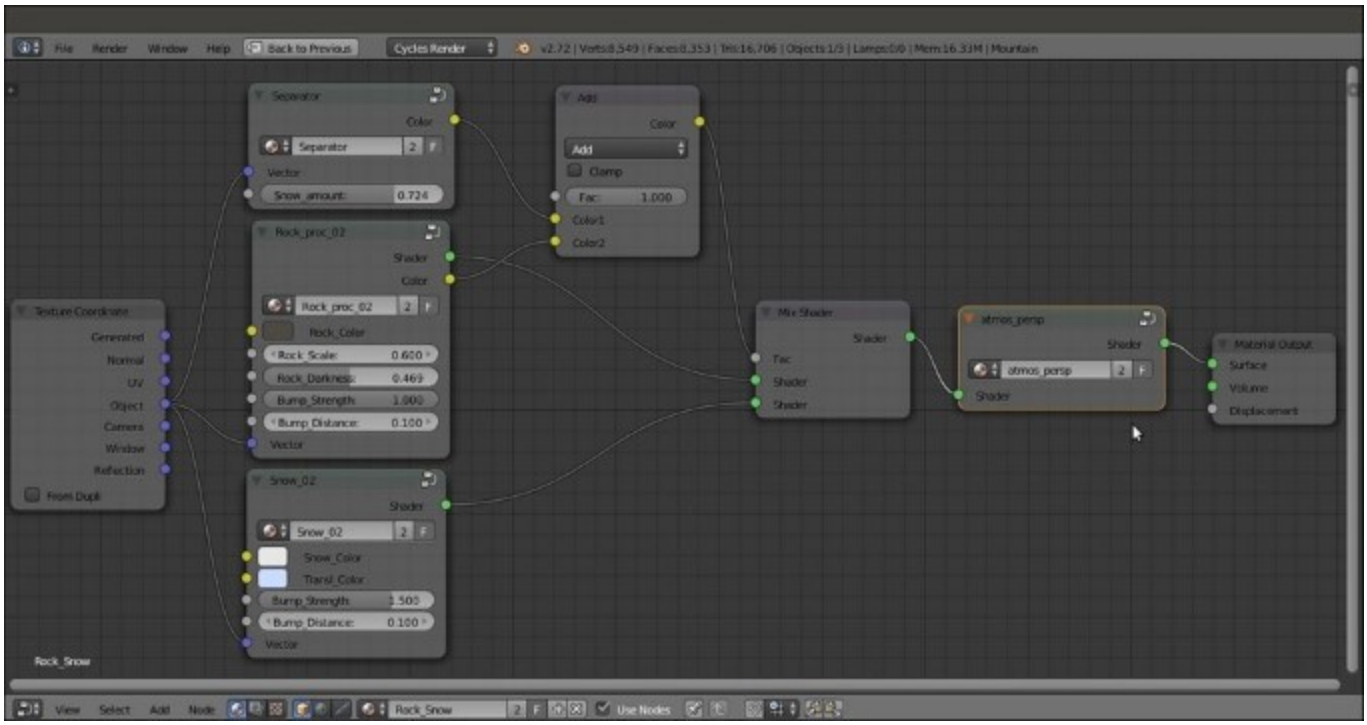


The outputs of the SLOPE and DENSITY frames added inside the Separator node group

18. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Separator** node group. For the rendering of the image at the beginning of this recipe, I've set the **Snow_amount** value to `0.724`.
19. Add one more **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Add**, and connect the **Color** output of the **Separator** group node to the **Color1** input socket. Then connect the **Color** output of the **Rock_proc_02** group node to the **Color2** input socket. Set the **Fac** value to `1.000` and connect the **Color** output to the **Fac** input socket of the **Mix Shader** node.

Adding the atmospheric perspective

The final step we can do to improve our material is to append the **Fog_underwater** node group, from the `9931OS_05_underwater_final.blend` file located in **Nodetree**. Rename this node `Atmos_persp` and paste it just before the **Material Output** node. Then press *Tab* to open the group by entering **Edit Mode**. Set the **Multiply** node value to `0.010` and the color values of the **Emission** shader for **R** to `0.078`, **G** to `0.133`, and **B** to `0.250` as shown in the following screenshot:



The overall network and the atmospheric perspective node group added at the end of the shader. Note the Color output of the Rock material added to the output of the Separator to work as stencil or blending factor.

How it works...

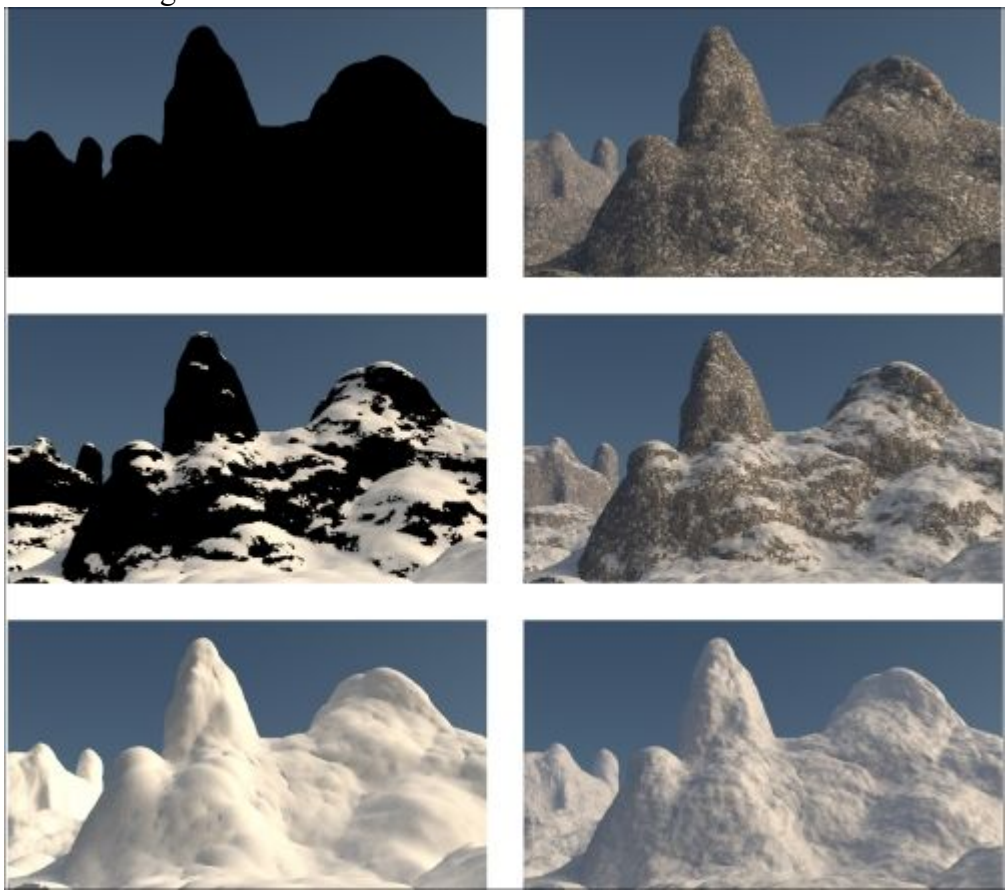
Now let's see how this material actually works, by dividing the creation's process into three parts:

- Firstly, we appended the `Snow` material and made a group, exposing the required properties and changing the way the bump works. In other words, we deleted the output to the **Displacement** input of the **Material Output** node and implemented a per shader bump.

  This doesn't really make a big difference in the final rendering. Just be aware that a bump piped in the **Displacement** socket can react to Ambient Occlusion (which we didn't use in the scene, by the way), but this is not true with the per shader bump.
- Secondly, we appended the `Rock_procedural` material and made a group of it as well. Again, all the necessary values were exposed, and although we kept the material unaltered in this scene, the group could now be easily reused for different kinds of rock in other projects or on different objects.

  We added a **Math** node set to **Multiply** for every texture **Scale** value that needed to be driven by one single exposed input. The first **Value** of the **Math** node, set to the original scale value, gets multiplied by the driven second **Value**, thus increasing or decreasing (for values smaller than `1.000`) the final scale value.

- Thirdly, we built `Separator`, a node group outputting gray-scale values that are connected to the **Fac** input of the **Mix Shader** node, which works as a stencil map, separating the two different materials on the mesh surface accordingly to black and white values. The two gradient textures in the **SLOPE** frame, mapped on the position and the normals of the mesh and then blended together by the **Burn** nodes, make the snow material (the white color value of the stencil map) appear more on the mesh's faces that have more a horizontal trend than a vertical one. Thanks to the **Add01** node, mixing the gradients driven by the exposed input **Snow_amount** and influencing the gradient of the **ColorRamp01** node, it's also possible to set the quantity of snow (the white color in the stencil) on the whole object. The mixed textures in the **DENSITY** frame make the separation line between black and white more frayed and realistic, and so also the **Color** output of the **Separator** group that is added to a **Color** output of the **Rock** shader just before being connected to the **Fac** input of the **Mix Shader**. Have a look at the following screenshot:



The only mask and the Rendered versions of three different values of the Snow_amount slider

In the preceding set of screenshots, you can see the different effects of the `0.000`, `0.700`, and `1.000` values of the **Snow_amount** slider. The black-and-white mask works as a separator between the rock and the snow materials.

Creating a realistic Earth as seen from space

In this recipe, we will create a realistic Earth as shown in the following screenshot, using both image textures from the Web and some procedurals:

The Earth as it appears in the final rendering

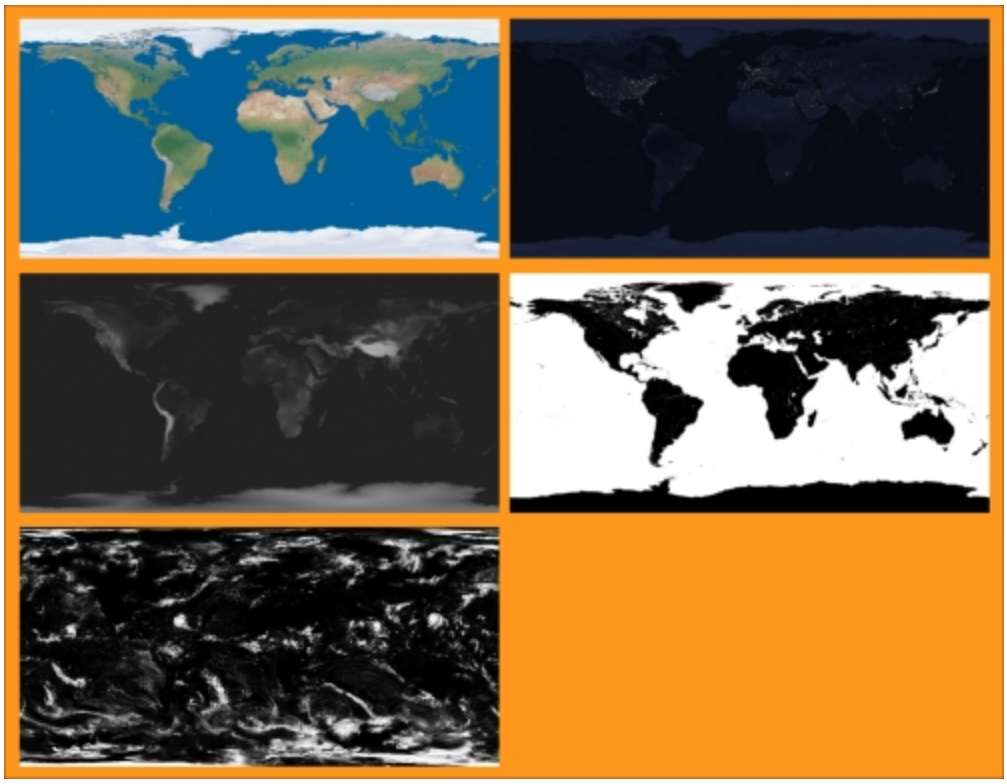
Getting ready

The image textures provided with this Cookbook have generally been heavily down-scaled and are good for demonstration purposes only (in this case, for a very distant Earth render). For better results with this recipe, replace these low-resolution images with high-resolution versions that you can find at these addresses:

- http://www.shadedrelief.com/natural3/pages/textures.html
- http://www.shadedrelief.com/natural3/pages/clouds.html
- http://celestia.h-schmidt.net/earth-vt/
- http://www.celestiamotherlode.net/catalog/earth.php

Before you download anything, always take a look at the license of the images provided by any site you can find to ensure that they are released as freely usable, especially if you are going to use them for commercial work. All the preceding links should be okay, but on the Internet, things can change quite quickly, so double-check!

You will need at least five image maps for this recipe: Earth-color, the color of the land or sea in daylight; Earth-night, the color of the land or sea at night (usually provided with superimposed city lights); Earth-bump, a gray-scale, high map of the continents; Earth-spec, an outline with the continents in black and the water masses perfectly white; and Clouds, a gray-scale map of the clouds as shown in the following screenshot:



The five image textures

Actually, Cycles can handle very big textures pretty well, even 16 K images (that is, images made by 16.000 pixels for the longest side), so you can use them at the best resolution you can find. Be aware that the bigger the resolution of the textures, the longer the rendering times, especially if they are used as bump maps.

Now perform the following steps:

1. Start Blender and switch to the Cycles Render engine.
2. Delete the default **Cube** and add a **UV Sphere** (with the mouse arrow in the 3D view, press *Shift* + *A* and navigate to **Mesh** | **UV Sphere**). In the **Outliner**, rename it as `Earth_Surface`.
3. With the mouse arrow in the **Camera** view, press the *1* key on the numeric keypad to go to the **Front** view. Then press the *5* key to switch to **Orthogonal**. Next, press *Tab* to enter **Edit Mode**, followed by *A* to select all the vertices. Finally, press *U*. In the **UV Mapping** pop-up menu, select **Sphere Projection**. Then exit **Edit Mode**.
4. Make sure you place the 3D Cursor at the center of the **UV Sphere**. Then add an **Empty** (press *Shift* + *A* and navigate to **Empty** | **Arrows**). In the **Object data** window, set its **Size** to `2.00` and rename it as `Empty_terminator`. Go to the **Object Constraints** window and assign a **Damped Track** constraint to the **Empty_terminator**. In the **Target** field, select the **Sun** item (the Lamp), and in the **To** field, click on the **X** button.

5. Reselect the **UV Sphere** and go to the **UV Maps** subpanel under the **Object data** window. Click on the **+** icon button to add a new UV coordinates layer. Rename it as `UVMap_terminator`.
6. Go to the **Object modifiers** window and assign a **Subdivision Surface** modifier first, followed by a **UVProject** modifier. For this modifier, in the **UV Map** field, select the **UVMap_terminator** item. In the **Object to use as projector transform** field, select the **Empty_terminator**.
7. Press *Shift* + *D* and press *Enter* to duplicate the **Earth_Surface** object. In the **Transform** subpanel under the **Properties** panel to the right (press *N* if this is not activated), set the **Scale** value for **X**, **Y**, and **Z** to `1.001`. Rename it as `Earth_Clouds`.
8. Duplicate it again, set the **Scale** value to `1.002`, and rename it as `Earth_Atmosphere`.
9. Add a new **Empty** (press *Shift* + *A* and navigate to **Empty** | **Plain Axes**) and rename it as `Empty_Earth`. In the **Object data** window, set its **Size** to `2.00`. Press *Shift* and select the **Earth_Surface**, **Earth_Clouds**, **Earth_Atmosphere**, and the **Empty_Earth** objects. Press *Ctrl* + *P* and click on **Object** to parent the three UV Spheres to **Empty_Earth**.
10. Select **Empty_Earth**, and in the **Transform** panel, set the **Rotation** values of **X** to `18.387°`, **Y** to `0.925°`, and **Z** to `-4.122°` (you can obviously rotate the **Empty_Earth** as you wish, but this helps provide a nice point of view on the specular effect showing on the oceans).
11. Select the **Camera**, and in the **Transform** panel, set the **Location** values of **X** to `-0.64000`, **Y** to `-4.70000`, and **Z** to `0.12000`. Then set the **Rotation** values of **X** to `89°`, **Y** to `0°`, and **Z** to `-9°`. Go to the **Object data** window and change the **Focal Length** to `60.000` (millimeters). Press the *0* key on the numeric keypad to go to the **Camera** view.
12. Go to the **World** window and change the background **Color** to pure black.
13. Select the **Lamp** and change it to a **Sun**. Set the **Size** to `0.050` and the **Strength** to `10.000`. Set the **Color** values for **R** to `1.000`, **G** to `0.902`, and **B** to `0.679`. In the **Transform** panel, set the **Location** values of **X** to `158.00000`, **Y** to `-27.00000`, and **Z** to `107.00000`. For **Rotation**, set **X** to `1.5°`, **Y** to `56°`, and **Z** to `-8°` (Sun lamps don't need a location, but in this case, we need it to establish a target for a later-to-come day/night terminator trick).
14. Go to the **Render** window. Under the **Sampling** subpanel, set the **Clamp Direct** and **Clamp Indirect** values to `1.00`, the **Preview** samples to `20`, and the **Render** samples to `50`.
15. Go to the **Scene** window. In the **Color Management** subpanel, click on the **Use Curves** item. Set the **Exposure** value to `1.000`. Then click inside the curve window to add a new point, and place it at position **X** as `0.61149` and **Y** as `0.71250`. Then set the value of the **B** channel for the **White Level** between `0.800` and `0.850`.

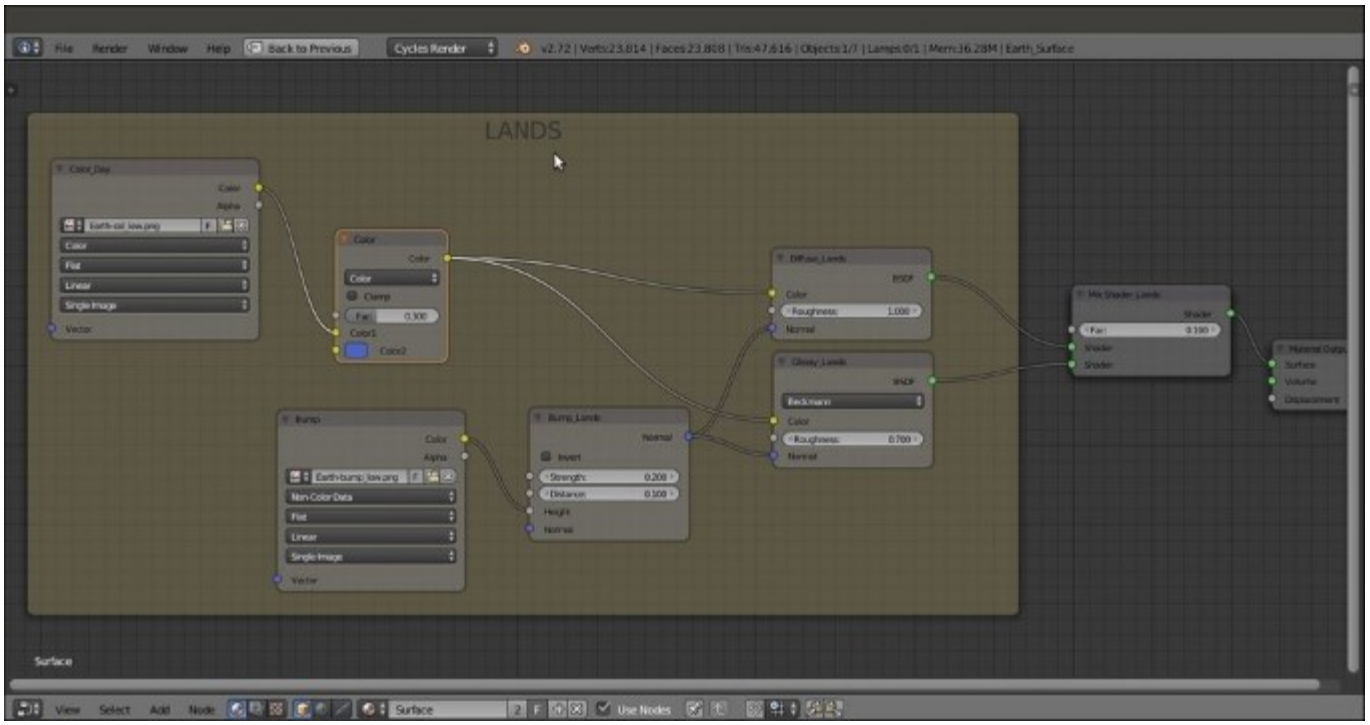
How to do it...

After the creation of the 3D scene and the setting of the lighting, let's go for the materials, starting with the planet's surface.

The planet surface

In the **Outliner** (just temporarily), hide the **Earth_Clouds** and **Earth_Atmosphere** objects by clicking on the little eye icons to the right side of the names. This is to see only the **Earth_Surface** in the viewport, rendered and updated in real time as we work on the material:

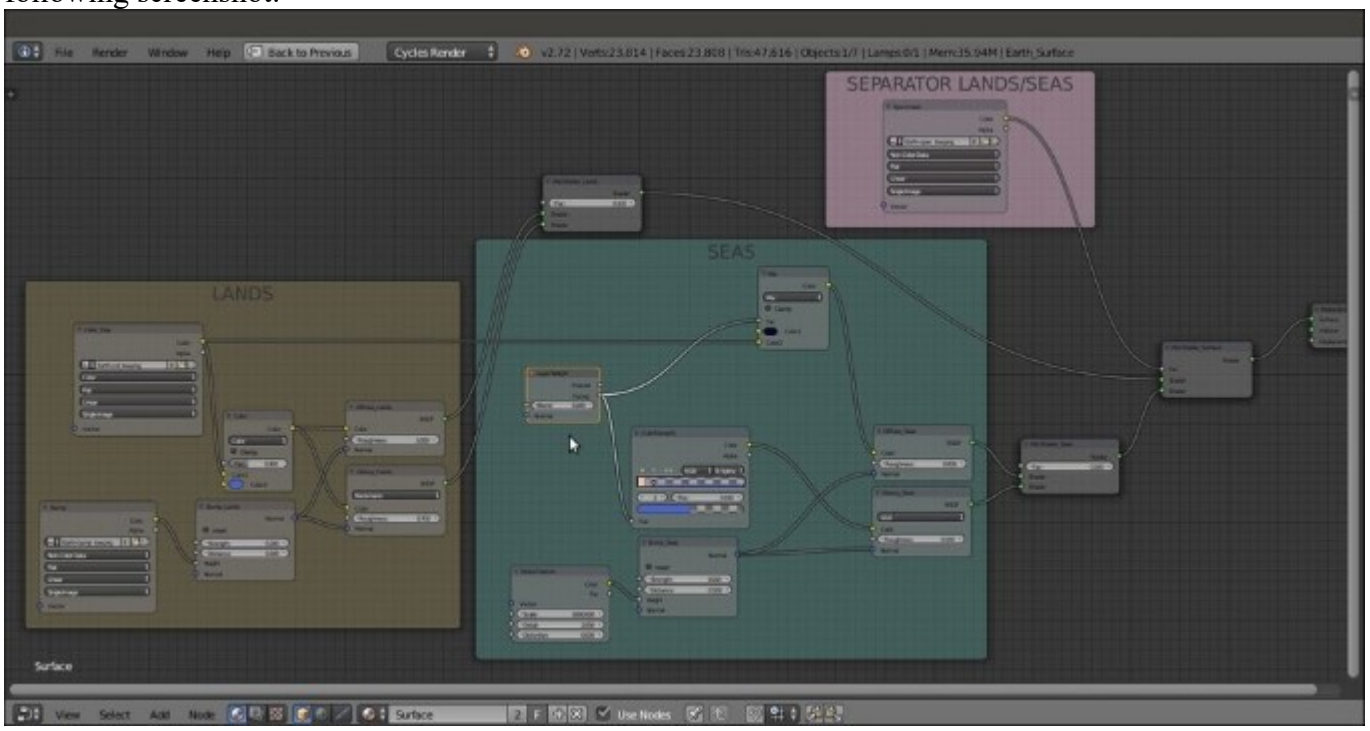
1. Select the **Earth_Surface** object. Click on the **New** button in the **Material** window under the **Properties** panel or in the **Node Editor** toolbar. Rename the material as `Surface`.
2. In the **Material** window under the main **Properties** panel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, load a new **Diffuse BSDF** shader and set its **Roughness** value to `1.000`. In the second **Shader** slot, load a **Glossy BSDF** node. Then set its **Roughness** value to `0.700` and **Distribution** to **Beckmann**. Set the **Fac** value of the **Mix Shader** to `0.100`.
3. Press *N* in the **Node Editor** window to open the **Active Node** panel. Label the shaders as `Diffuse_Lands` and `Glossy_Lands` and the **Mix Shader** as `Mix Shader_Lands`.
4. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and connect its **Color** output to both the **Color** input sockets of the **Diffuse_Lands** and **Glossy_Lands** shaders. Click on the **Open** button on the **Image Texture** node, browse to your `textures` directory, and load the `Earth-col_low.png` image (or a high-resolution version, if available). Label the image node as `Color_Day`.
5. Add a new **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Color** output of this **Image Texture** node to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of the **Bump** node to the **Normal** input sockets of both the **Diffuse_Lands** and the **Glossy_Lands** nodes.
6. Label the second **Image Texture** node as `Bump`. Then click on its **Open** button and load the `Earth-bump_low.png` image. Set the **Color Space** to **Non-Color Data**. Label the **Bump** node as `Bump_Lands` and set the **Strength** value to `0.020`.
7. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and paste it between the **Color_Day** and the **Diffuse_Lands** nodes. Set the **Blend Type** to **Color**, the **Fac** value to `0.300`, and the **Color2** value for **R** to `0.072`, **G** to `0.127`, and **B** to `0.578`.
8. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the two image texture nodes (**Color_Day** and **Bump**), the **Color** node, **Bump_Lands** node, **Diffuse_Lands** and **Glossy_Lands** shaders, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the **Frame** as `LANDS` as shown in the following screenshot:



The LANDS frame

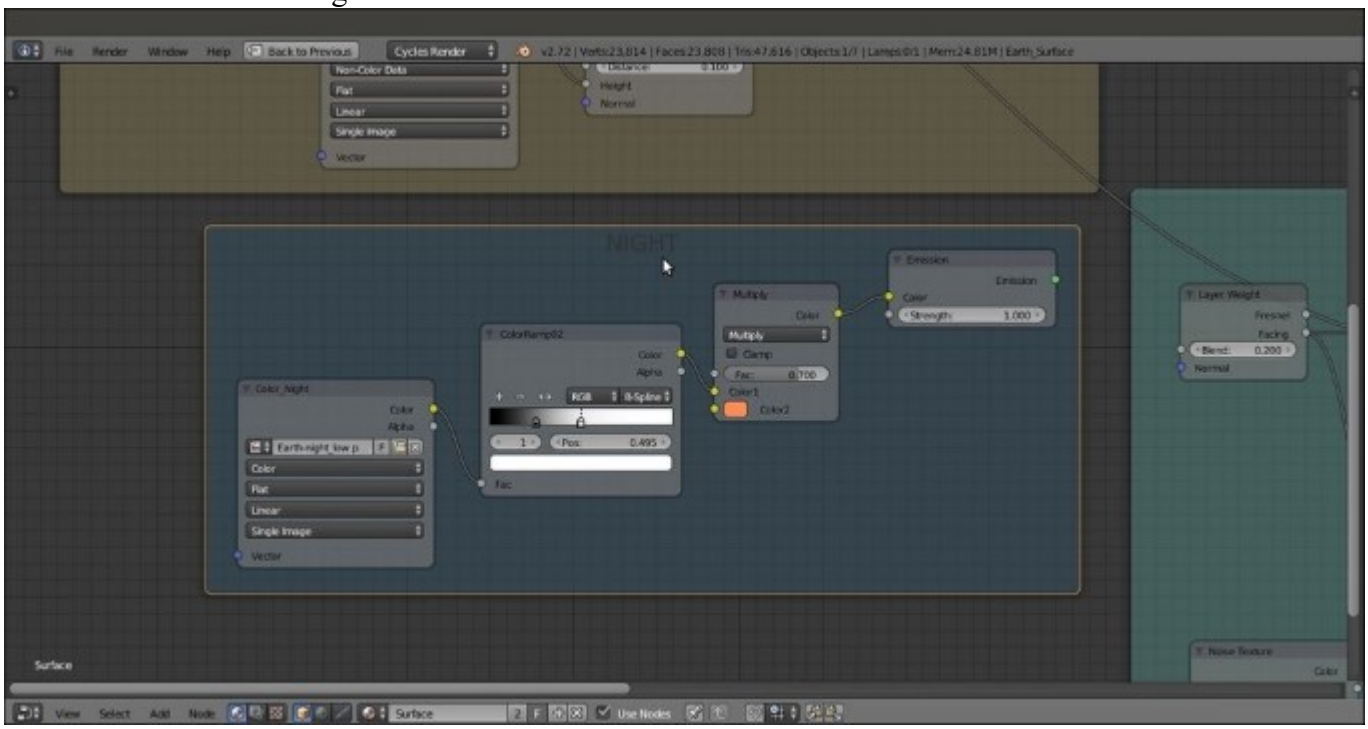
9. Now add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**), a **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**), and a **Glossy BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**).
10. Set the **Noise Texture** node's **Scale** value to `1000.000` and connect its **Color** output to the **Height** input socket of the **Bump** node. Label this node as `Bump_Seas`, set the **Strength** value to `0.015`, and connect its **Normal** output to the **Normal** input sockets of the new **Diffuse BSDF** and **Glossy BSDF** shaders. Label them as `Diffuse_Seas` and `Glossy_Seas` and set the **Glossy BSDF** node's **Roughness** value to `0.150`.
11. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the new nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame as `SEAS`.
12. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader_Seas`, and place it just under the **Mix Shader_Lands** node. Set the **Fac** value to `0.100`. Connect the output of the **Diffuse_Seas** node to the first **Shader** input, and the output of the **Glossy_Seas** node to the second **Shader** input socket.
13. Press *Shift* + *D* to duplicate the **Mix Shader_Seas** node and paste it between the **Mix Shader_Lands** and the **Material Output** nodes. Label it as `Mix Shader_Surface`. Connect the output of the **Mix Shader_Seas** node to its second **Shader** input socket.
14. Add a new **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and rename it as `Spec/mask`. Connect its **Color** output to the **Fac** input socket of the **Mix Shader_Surface** node. Click on the **Open** button to load the `Earth-spec_low.png` image. Set the **Color Space** to **Non-Color Data**.

15. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the **Spec/mask** node and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame as `SEPARATOR LANDS/SEAS`.
16. Now click on the **Color** output of the **Color_Day** image texture inside the **LANDS** frame, and drag it so that it is connected to the **Color** input of the **Diffuse_Seas shader** node inside the **SEAS** frame.
17. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) to the **SEAS** frame (just add it and parent it to the frame). Paste it just before the **Diffuse_Seas** shader. Switch the **Color1** connection to the **Color2** input socket. Then set the **Color1** values for **R** to `0.002`, **G** to `0.002`, and **B** to `0.022`.
18. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) to the **SEAS** frame and label it as `ColorRamp01`. Connect the **Color** output to the **Color** input of the **Glossy_Seas** shader. Set **Interpolation** to **B-Spline**. Change the black color stop (index **0**) to pure white and the white color stop values (index **1**) for **R** to `0.072`, **G** to `0.127`, and **B** to `0.578`, with **Alpha** value set to `0.000`. Move it to `0.150` position. Click on the **+** icon button to add a new color stop. Change its **Color** values for **R** to `0.965`, **G** to `0.462`, **B** to `0.223`, and **Alpha** to `1.000`. Move it to `0.075` position.
19. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) to the **SEAS** frame. Connect the **Facing** output to the **Fac** input socket of the **ColorRamp01** node and the **Fac** input socket of the **MixRGB** node. Set the **Blend** factor to `0.200` as shown in the following screenshot:



The LANDS and the SEAS frames connected and separated by the simple SEPARATOR LANDS/SEAS

20. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), and an **Emission** shader (press *Shift* + *A* and navigate to **Shader** | **Emission**). Label the **Image Texture** node as `Color_Night` and the **ColorRamp** as `ColorRamp02`.
21. Connect the **Color_Night** node's **Color** output to the **Fac** input socket of the **ColorRamp02** node, and the **Color** output of this node to the **Color1** input socket of the **MixRGB** node. Then connect the **MixRGB** node's output to the **Color** input of the **Emission** node.
22. In the **Color_Night** image texture node, load the `Earth-night_low.png` image. Set the **ColorRamp02** node's **Interpolation** to **B-Spline** and move the black color stop to `0.250` position. Then move the white color stop to the `0.495` position. Set the **MixRGB** node's **Blend Type** to **Multiply**, the **Fac** value to `0.700` and the **Color2** values for **R** to `1.000`, **G** to `0.257`, and **B** to `0.090`. Set the **Emission** node's **Strength** value to `1.000`.
23. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select these new nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame as `NIGHT`. This is shown in the the following screenshot:

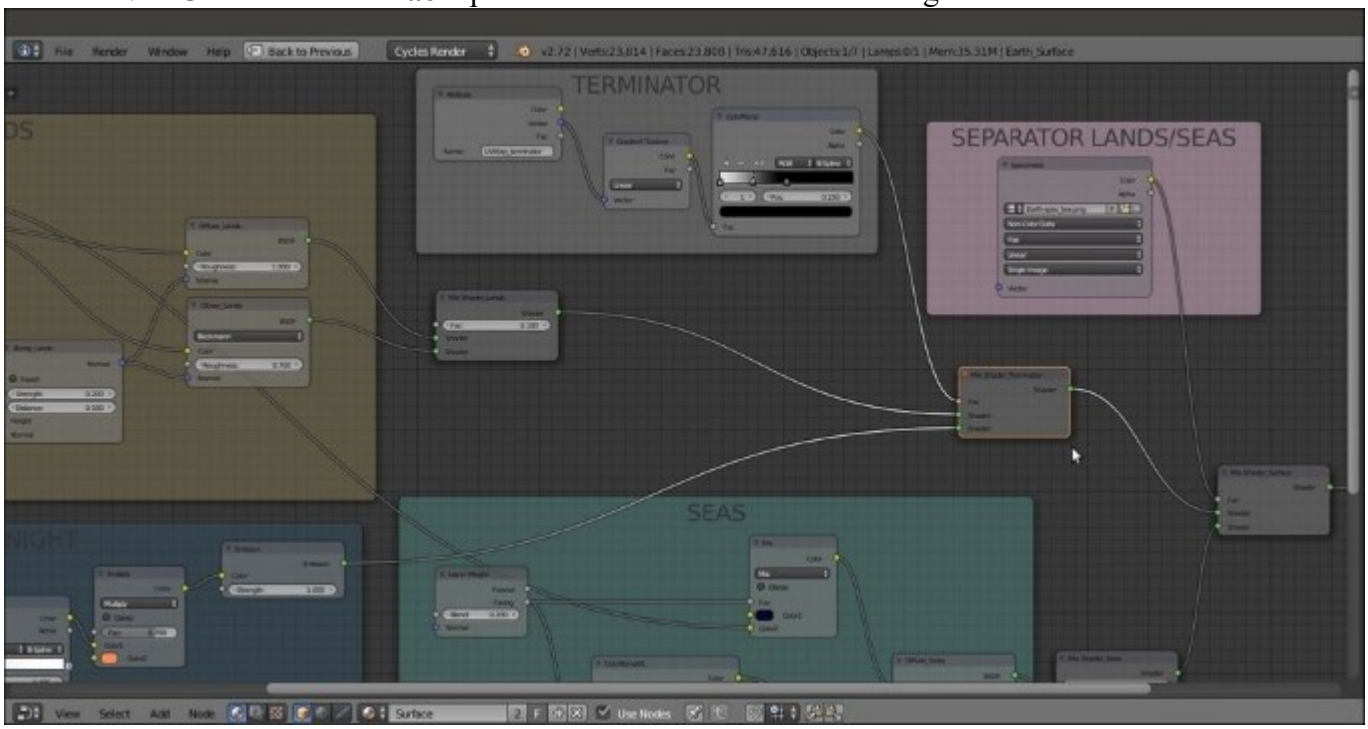


The NIGHT frame

24. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**), a **Gradient Texture** node (press *Shift* + *A* and navigate to **Texture** | **Gradient Texture**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Vector** output socket of the **Attribute** node to the **Vector** input socket of the **Gradient Texture** node, and the **Color** output of this node to the **Fac** input socket of the **ColorRamp** node.
25. In the **Name** slot of the **Attribute** node, write `UVMap_terminator`. Set the **ColorRamp** node's **Interpolation** to **B-Spline**. Then move the black color stop to `0.500` and the white

color stop to the `0.000` position. Click on the **+** icon button to add a new color stop. Set its color to pure black as well.

26. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select these three nodes and then the frame. Press *Ctrl* + *P* to parent them. Rename the frame as `TERMINATOR`.
27. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader_Terminator`, and paste it just between the **Mix Shader_Lands** and the **Mix Shader_Surface** nodes. Connect the output of the **Emission** node inside the **NIGHT** frame to its second **Shader** input socket, and the **Color** output of the **ColorRamp** inside the **TERMINATOR** frame to its **Fac** input socket as shown in the following screenshot:



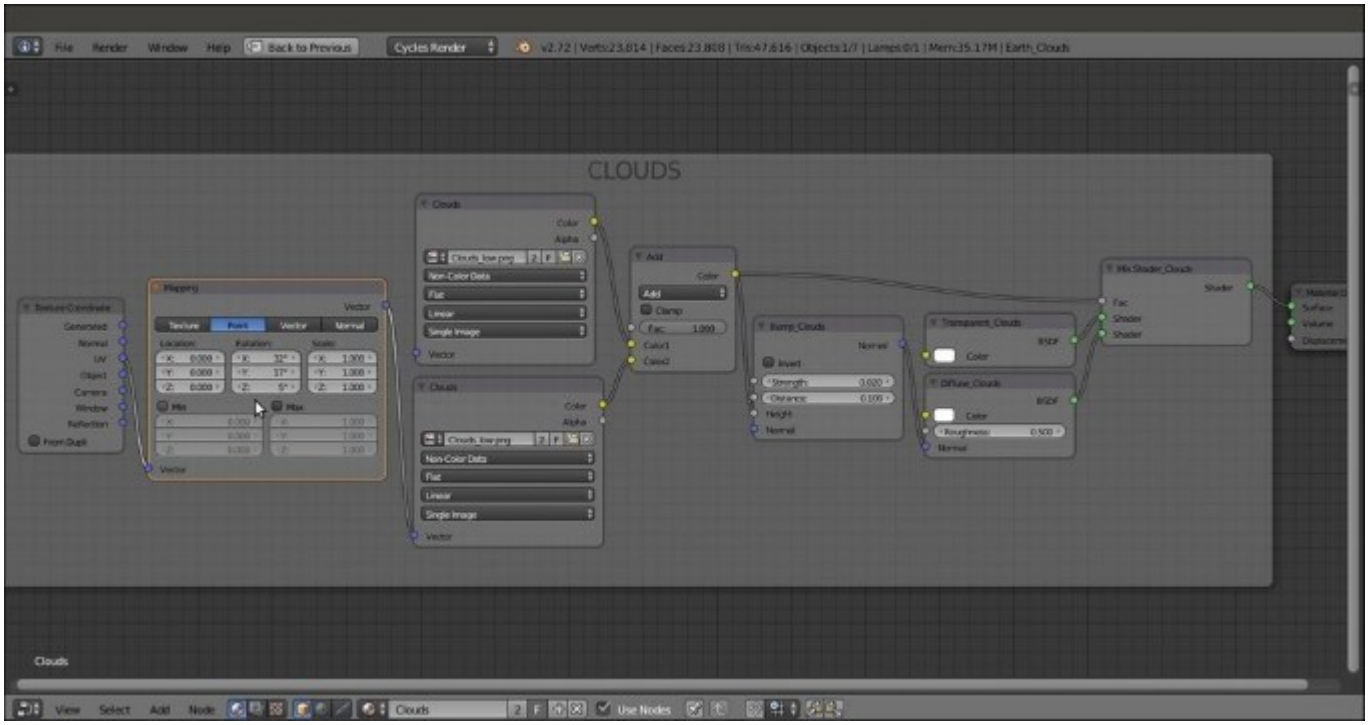
The TERMINATOR frame added to the surface material network

The clouds

As second planet material, let's go with the clouds by performing the following steps:

1. Now go to **Outliner**, unhide the **Earth_Clouds** sphere, and select it. Click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** toolbar. Rename the material as `Clouds`.
2. In the **Material** window under the main **Properties** panel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. Label it as `Mix Shader_Clouds`, and in the first **Shader** slot, load a **Transparent BSDF** shader. In the second **Shader** slot, load a new **Diffuse BSDF** shader. Set the color of both the shaders to pure white.
3. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**).

4. Label the **Image Texture** node as `Clouds` and the **Bump** node as `Bump_Clouds`. Connect the **Bump** node's output to the **Normal** input of the **Diffuse_Clouds** node. Set the **Strength** value to `0.020`.
5. Click on the **Open** button of the **Clouds** image texture node and load the `Clouds_low.png` image. Set the **Color Space** to **Non-Color Data**.
6. Press *Shift* + *D* to duplicate the **Clouds** image node. Then add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **UV** output of the **Texture Coordinate** node to the **Mapping** node, and the output of this node to the **Vector** input socket of the duplicated **Clouds** image texture node.
7. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect the output of both the two **Clouds** image texture nodes to the **Color1** and **Color2** input sockets. Set the **Blend Type** to **Add** and the **Fac** value to `1.000`. Connect the **Color** output to the **Height** input socket of the **Bump_Clouds** node and to the **Fac** input socket of the **Mix Shader_Clouds** node.
8. In the **Mapping** node, set the **Rotation** values of **X** to `32°`, **Y** to `17°`, and **Z** to `5°`.
9. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select the nodes and then the **Frame**. Then press *Ctrl* + *P* to parent them. Rename the frame as `CLOUDS`. This is shown in the screenshot:



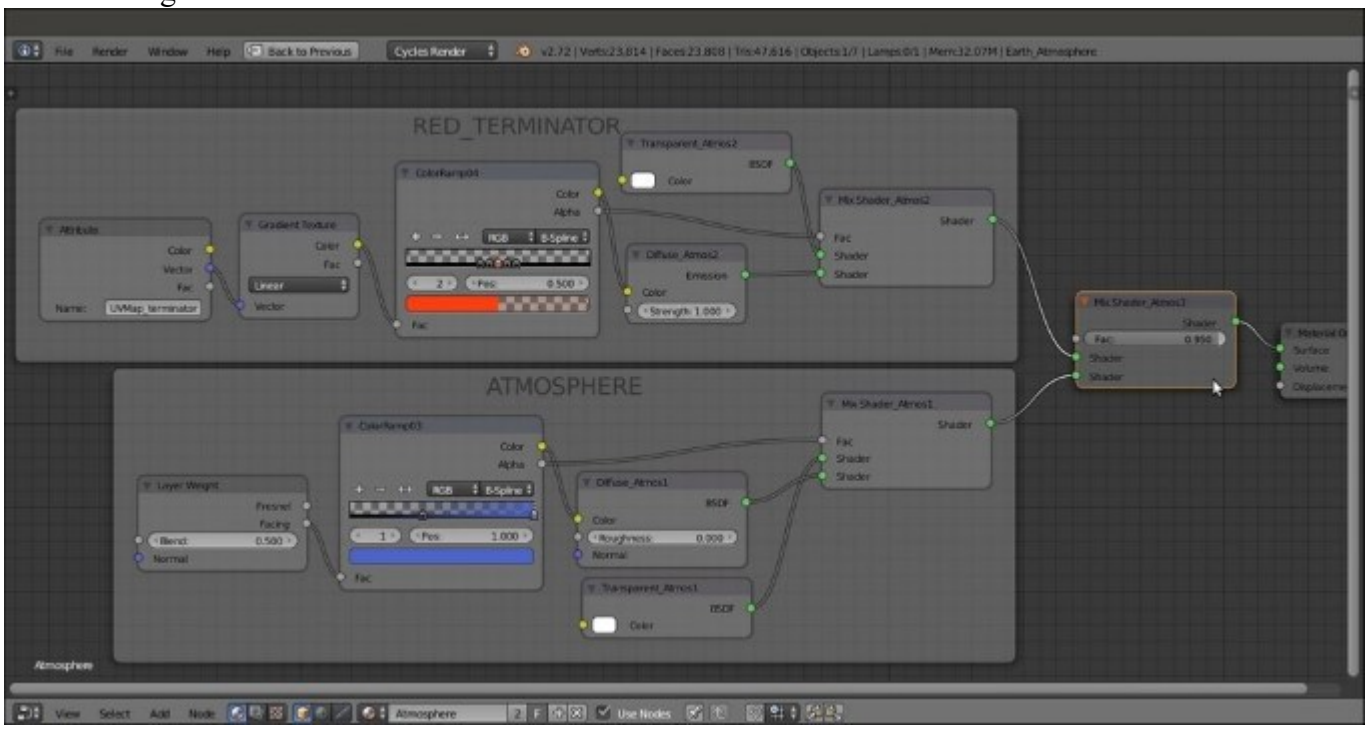
The CLOUDS material

The atmosphere

The third planet material is the atmosphere layer:

1. In **Outliner**, unhide the **Earth_Atmosphere** sphere and select it. Click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** toolbar. Rename the new material as `Atmosphere`.
2. In the **Material** window on the right, under the main **Properties** panel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. Label it as `Mix Shader_Atmos1`, and in the first **Shader** slot, load a **Transparent BSDF** shader (label it `Transparent_Atmos1`). In the second **Shader** slot, load a **Diffuse BSDF** shader (label it `Diffuse_Atmos1`).
3. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Facing** output of the **Layer Weight** node to the **Fac** input of the **ColorRamp** (label it `ColorRamp03`). Set the **ColorRamp03** node's **Interpolation** to **B-Spline**, move the black color stop to the `0.395` position, and set the **Alpha** value to `0.000`. Set the color of the white color stop (index **1**) for **R** to `0.072`, **G** to `0.127`, and **B** to `0.578`.
4. Connect the **Color** output of the **ColorRamp03** node to the **Color** input socket of the **Diffuse_Atmos1** node, and the **Alpha** output to the **Fac** input of the **Mix Shader_Atmos1** node. Set the **Layer Weight** node's blend factor to `0.500`.
5. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select these nodes and then the **Frame**. Then press *Ctrl* + *P* to parent them. Rename the frame as `ATMOSPHERE`.
6. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**), a **Gradient Texture** node (press *Shift* + *A* and navigate to **Texture** | **Gradient Texture**), and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Vector** output socket of the **Attribute** node to the **Vector** input socket of the **Gradient Texture** node, and then the **Color** output of this node to the **Fac** input socket of the **ColorRamp** (label it `ColorRamp04`).
7. In the **Name** slot of the **Attribute** node, type `UVMap_terminator`. Set the **ColorRamp04** node's **Interpolation** to **B-Spline**. Then move the black color stop to the `0.400` position and the white color stop to the `0.600` position, but change this stop's color to black as well. Click on the **+** icon button to add a new color stop. Set its color to pure black and move it to the `0.450` position. Click on the **+** icon button again to add a new color stop. Set its color to pure black and move it to the `0.550` position. Set the **Alpha** of all the four black color stops to the `0.000`. Click once more on the **+** icon button to add a new color stop. Set its color values for **R** to `1.000`, **G** to `0.047`, and **B** to `0.005`. Set **Alpha** value to `0.100` and move it to the `0.500` position.
8. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), a **Transparent BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Transparent BSDF**), and an **Emission** node (press *Shift* + *A* and navigate to **Shader** | **Emission**). Label them as `Mix Shader_Atmos2`, `Transparent_Atmos2`, and `Diffuse_Atmos2`.
9. Connect the **Transparent_Atmos2** node's output to the first **Shader** input socket of the **Mix Shader_Atmos2** and the **Diffuse_Atmos2** output to the second **Shader** input. Then connect the **Color** output of the **ColorRamp04** node to the **Color** input socket of the **Diffuse_Atmos2** node and the **Alpha** output to the **Fac** input socket of the **Mix Shader_Atmos2** node.
10. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select these nodes and then the **Frame**. Then press *Ctrl* + *P* to parent them. Rename the frame as `RED_TERMINATOR`.
11. Add a final **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader_Atmos3`, and set the **Fac** value to `0.950`. Connect the output of the **RED_TERMINATOR** frame to the first **Shader** input socket and the output of the

ATMOSPHERE frame to the second **Shader** input socket. Then connect the output of the **Mix Shader_Atmos3** node to the **Surface** input socket of the **Material Output** node as shown in the following screenshot:



The RED_TERMINATOR and the ATMOSPHERE frames

How it works...

The three overlapping UV Spheres technique is quite old, and (at least for what relates to Blender) dates back to almost 2004—more precisely to the *How to make a realistic planet in Blender(2004)* tutorial I wrote at that time for Blender version 2.23/2.30 (http://www.enricovalenza.com/rlpl.html). That tutorial is now outdated, but the technique and basic concepts still work, even in Cycles. Hence, we get the planet surface on the smaller of the spheres, a clouds layer on a slightly bigger sphere, and the enveloping atmospheric Fresnel effect on the biggest sphere.

We divided the material creation process into the three stages, corresponding to the three layers/spheres. First, we built the more complex of all the three shaders that is the Surface material:

- From step 1 to step 8, we built the shader for the continents—simple image textures connected as color factors to a **Diffuse BSDF** and **Glossy BSDF** shaders. From step 9 to step 13, we made the basic shader for the oceans.
- In steps 14 and 15, we split the continents component from the oceans using the `Earth-spec` map, a black-and-white image working as a stencil for the factor input of the **Mix Shader_Surface** node. We also connected the `Earth-color` map to the **SEAS** diffuse shader to bring color back to the oceans.

- From step 16 to step 19, we added a **ColorRamp** node to the **SEAS** frame, driven by a **Facing** fresnel node. This was done to enhance the color of the water's specularity (according to what NASA's satellite photos often show). A deep blue color was mixed with the color image map by a **MixRGB** node. Thanks to the **Facing** fresnel node, the blue color was mapped on the mesh faces perpendicular to the point of view, resulting in darker water masses towards the center of the Earth sphere.
- From step 20 to step 23, we built the night shader. The Earth-night image was clamped (contrasted) by a **ColorRamp** node, and the resulting brightness values were multiplied by a reddish color in the **MixRGB** node. All of this was then assigned to an **Emission** shader. The night side of the Earth surface shows only in the shadow part of the sphere thanks to the **Empty_terminator** trick.
- From step 24 to step 27, we built the day/night terminator stencil.
- Then, from step 28 to step 36, we built the `Clouds` layer on the second sphere. We added more variety to the single `Clouds_low.png` image by superimposing and offsetting (the mapping rotation of) a copy of the same cloud image.
- From step 37 to step 47, we built the `Atmosphere` layer on the third (bigger) sphere, with the Fresnel atmospheric effect and the reddish terminator.

As you have probably noticed, we didn't use any **Texture Coordinate** or **Mapping** nodes to map the image maps. This is because the UV Spheres had been unwrapped with **Image Texture** nodes. The existing UV coordinate layer was automatically taken into account by Cycles for the mapping.

For the ocean bump, which was obtained by the **Noise** procedural, the Generated mapping option was automatically used.

Thanks to the Damped Track constraints, which were targeted to the position of the Sun lamp, we could use the **Empty_terminator** object as a UV coordinates projector for the day/night division on the planet surface and for the red colored transition zone (the red terminator) in the `Atmosphere` layer.

Chapter 6. Creating More Complex Man-made Materials

In this chapter, we will cover the following recipes:

- Creating cloth materials with procedurals
- Creating a leather material with procedurals
- Creating a synthetic sponge material with procedurals
- Creating a spaceship hull shader

Introduction

In this chapter, we will see some more complex artificial materials, starting with the relatively simpler materials. Remember that the procedure is basically the same as that for all the materials we have seen so far—the generic shader followed by the color pattern or the bump effect (one or more), depending on the preponderance of the different components.

The only difference is the level of complexity they can reach (for example, look at the *Creating a spaceship hull shade* recipe at the end of this chapter).

Creating cloth materials with procedurals

In this recipe, we will create a generic cloth material, as shown in the following screenshot:

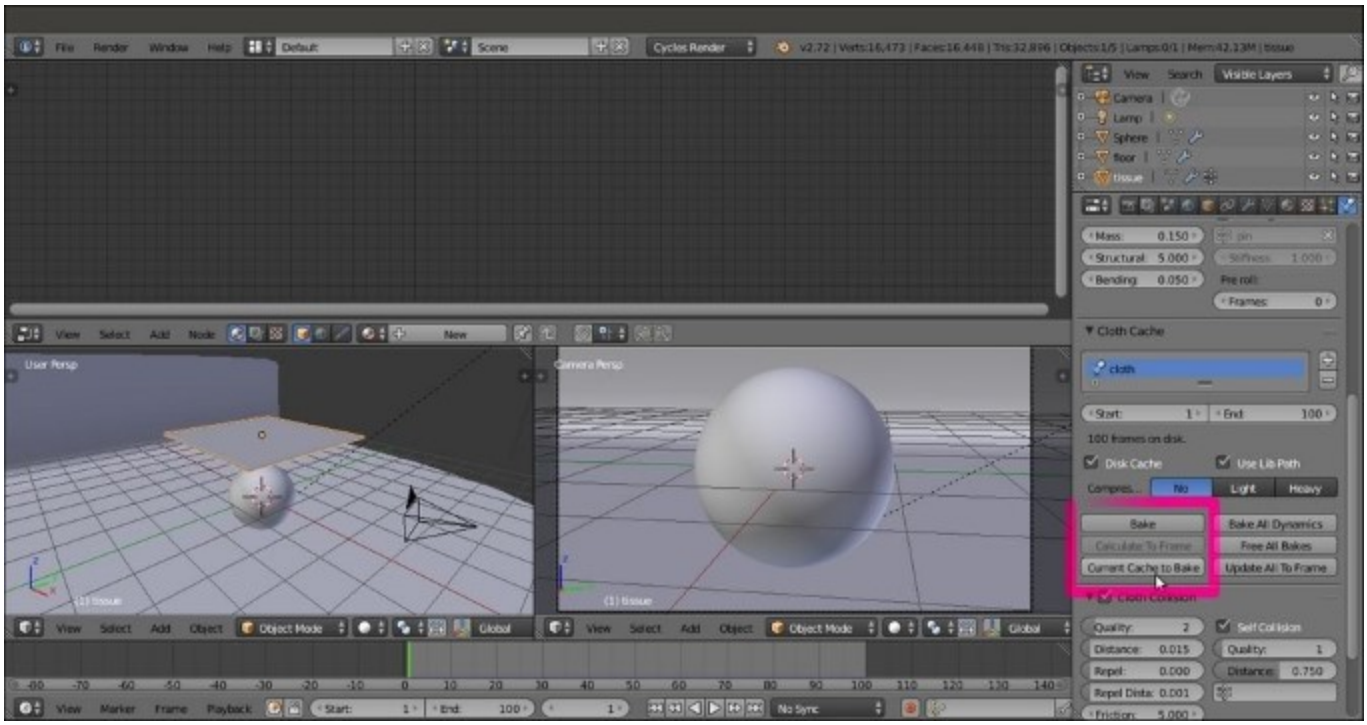
The cloth material as it appears in the final rendering

Getting ready

Before we start creating the material, let's set up the scene by performing the following steps:

1. Start Blender and load the `9931OS_cloth_start.blend` file. In the scene, there is already a cloth simulation.
2. Click on the **Play animation** button in the **Timeline** toolbar (or press *Alt* + *A*) to see the simulation running and being cached in real time. It consists of a Plane (our fabric) draped on a UV Sphere leaning on a bigger Plane (the floor).
3. After the simulation has been totally cached (a total of 100 frames), in the **Physics** window (the last tab to the right) under the **Cloth Cache** tab, press the **Current Cache to Bake** button to save the simulation. The 100-frame simulation is now cached and saved inside a folder (unless differently specified), named as a blend file, and stored on your hard drive in the same directory as the blend file.

From now on, there is no need to perform calculations about the simulation anymore. Blender will read the simulation data from that cache folder, so it will be possible to quickly scroll through the **Timeline** bar and immediately reach any frame inside the cached range.



The cloth simulation scene with the Cloth Cache subpanel to the right

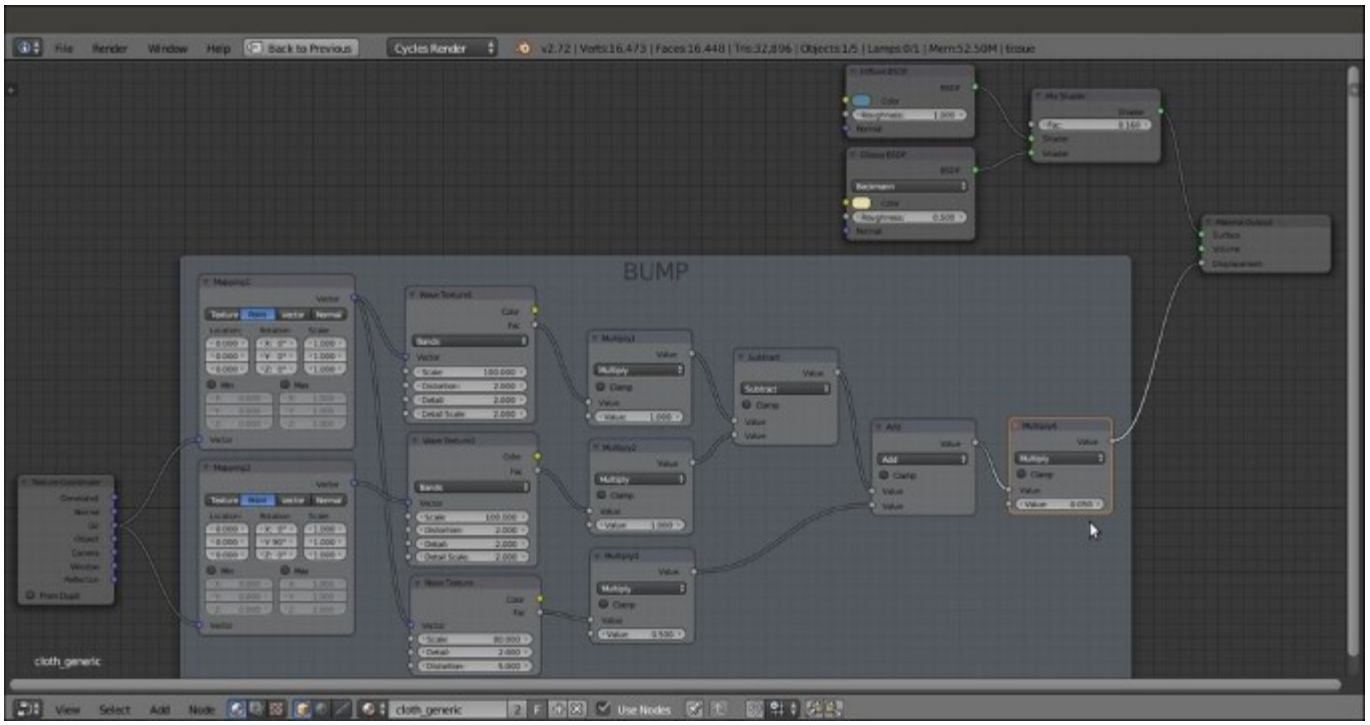
How to do it...

Now we are going to create the material on the fabric Plane, which has been first unwrapped by assigning a basic UV layer (**Object data** | **UV Maps**) and later subdivided by the **Specials** menu. This is done before the cloth simulation by performing the following steps:

1. Go to the **100** frame.
2. Make sure you have the **fabric** Plane selected. Click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** window. Rename the new material `cloth_generic`.
3. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** node, and in the second **Shader** slot, select a **Glossy BSDF** shader node.
4. Set the **Diffuse BSDF** node's **Roughness** value to `1.000`. Set the **Glossy BSDF** node's **Roughness** value to `0.500`. Change the **Glossy BSDF** node's **Color** values of **R** to `0.800`, **G** to `0.730`, and **B** to `0.369`. Set the **Fac** value of the **Mix Shader** node to `0.160`.
5. Add one **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and two **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**). In the **Properties** panel (press *N*) of the **Node Editor** window, label the two **Mapping** nodes as `Mapping1` and `Mapping2`. Then connect the UV output of the **Texture Coordinate** node to the **Vector** input sockets of both the **Mapping** nodes.
6. Now add two **Wave Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**) and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**). Label the

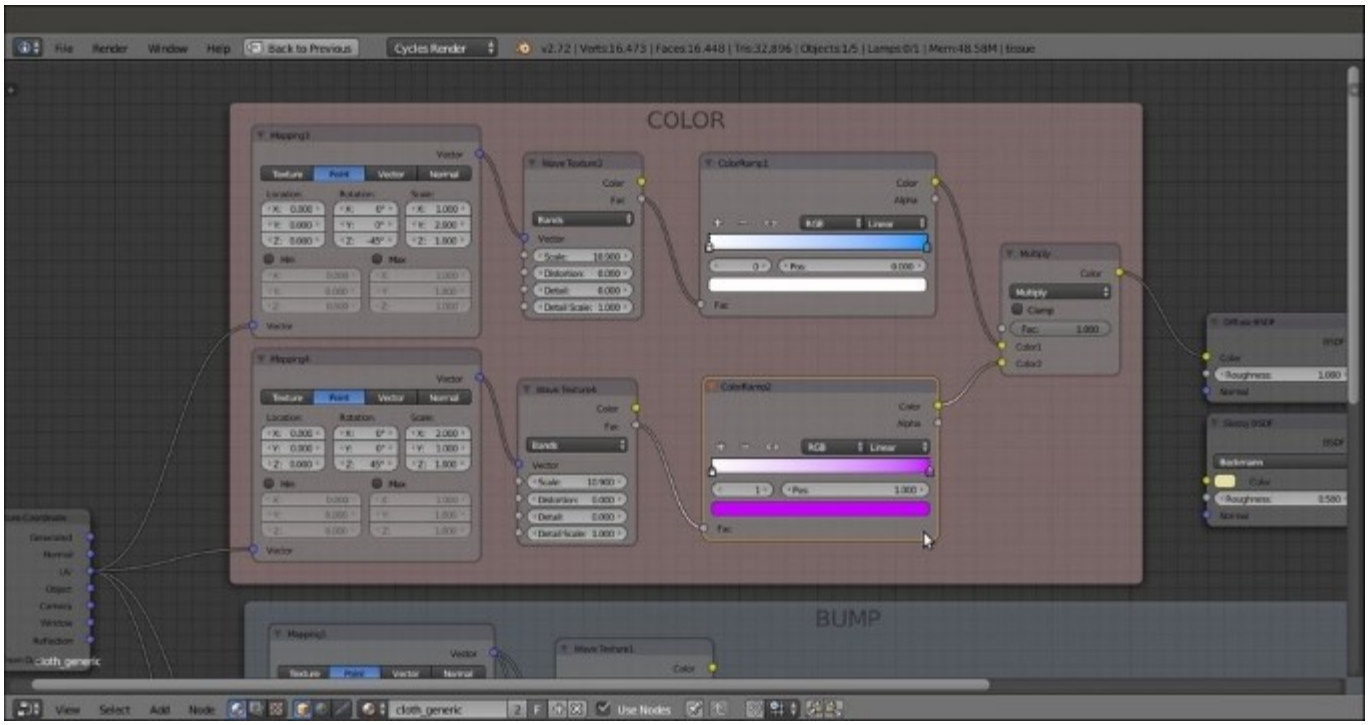
first two nodes as `Wave Texture1` and `Wave Texture2`. Connect the output of the **Mapping1** node to the **Vector** input sockets of the **Wave Texture1** node and the **Noise Texture** node. Connect the output of the **Mapping2** node to the **Vector** input of the **Wave Texture2** node.

7. Add three **Math** nodes (press *Shift* + *A* and navigate to **Converter** | **Math**), one for each texture node. Set their **Operation** mode to **Multiply** and label them as `Multiply1`, `Multiply2` and `Multiply3`. Then connect the **Fac** output of the **Wave Texture1** node to the first **Value** input socket of the **Multiply1** node, the **Wave Texture2** node to the **Multiply2** node, and the **Noise Texture** node to the **Multiply3** node. Set the second **Value** input socket of the **Multiply1** and **Multiply2** nodes to `1.000`, and leave the **Multiply3** node as `0.500`.
8. Go to the **Mapping2** node and set the **Rotation** value of **Y** to `90°`. Go to the **Wave Texture** nodes, and for both of them, set the **Scale** value to `100.000`, **Distortion** to `2.000`, and **Detail Scale** to `2.000`. For the **Noise Texture** node, set the **Scale** value to `80.000` and the **Distortion** value to `5.000`.
9. Add a new **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and set **Operation** to **Subtract**. Connect the output of the **Multiply1** and **Multiply2** nodes to the first and the second **Value** input sockets, respectively.
10. Press *Shift* + *D* to duplicate the last **Math** node. Set the **Operation** mode to **Add**. Connect the output of the **Subtract** node to its first **Value** socket and the output of the **Multiply3** node to the second **Value** input socket.
11. Press *Shift* + *D* to duplicate a **Multiply** node, label it as `Multiply4`, and connect the output of the **Add** node to the first **Value** input socket. Set the second **Value** input socket to `0.050`. Connect the last **Multiply** node's output to the **Displacement** input socket of the **Material Output** node.
12. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the two **Mapping** nodes, the three textures, all the **Math** nodes, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `BUMP`, as shown in the following screenshot:



The shader and the network for the fabric bump pattern

13. Add two new **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Label them as `Mapping3` and `Mapping4`. Then add two **Wave Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**). Label them as `Wave Texture3` and `Wave Texture4`. As before, connect the **UV** output of the **Texture Coordinate** node to both the **Mapping3** and **Mapping4** nodes, and the output sockets of the latter to the new **Wave Texture** nodes.
14. Go to the **Mapping3** node. Set the **Rotation** value of **Z** to `-45°` and the **Scale** value of **Y** to `2.000`. In the **Mapping4** node, set the **Rotation** value of **Z** to `45°` and the **Scale** value of **X** to `2.000`. Go to the two **Wave Texture** nodes to set the **Scale** value to `10.900` and the **Detail** value to `0.000`.
15. Add two **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and label them as `ColorRamp1` and `ColorRamp2`. Connect the **Fac** output of the **Wave Texture3** to the **Fac** input socket of the **ColorRamp1** node, and the **Fac** output of the **Wave Texture4** node to the **Fac** input socket of the **ColorRamp2** node.
16. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Connect the **Color** output of the two **ColorRamp** nodes to the **Color1** and **Color2** input sockets of the **MixRGB** node. Set **Blend Type** to **Multiply** and the **Fac** value to `1.000`.
17. Connect the **Color** output of the **Multiply-MixRGB** node to the **Color** input socket of the **Diffuse BSDF** shader node.
18. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the new nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the **Frame** `COLOR`, as shown in the following screenshot:



The COLOR frame

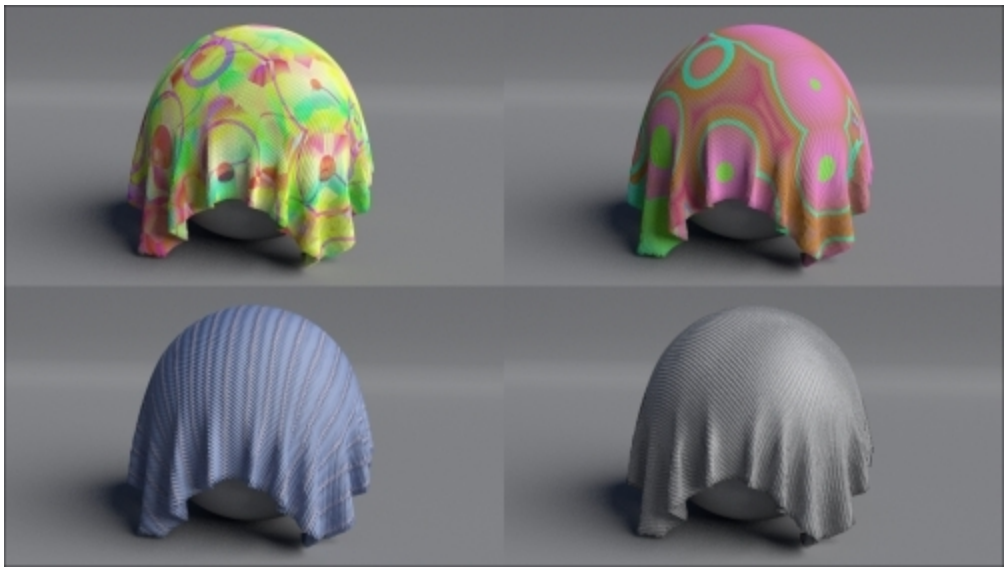
19. At this point, we can change the colors inside the two **ColorRamp** gradients to obtain colored patterns. In my example, I set the **ColorRamp1** colors ranging from pure white going to a light blue, and violet for the **ColorRamp2** node.

How it works...

- From steps 1 to 3, we just made the basic shader by mixing a rough **Diffues BSDF** shader (**Roughness** to `1.000`) with a light glossy shader (the low **Fac** value of the **Mix Shader** node shows a lot more of the diffuse component than the glossy component)
- From steps 4 to 10, we built the bump texture of the fabric by mixing two **Wave Texture** nodes in different orientations and by adding a bit of noise
- From steps 11 to 18, we built a simple cross-color pattern

There's more...

A lot of variations can be obtained by setting different values for the bump and using different texture nodes and combinations for the color pattern, as shown in the following screenshot:



Different color patterns of the cloth material

Tip

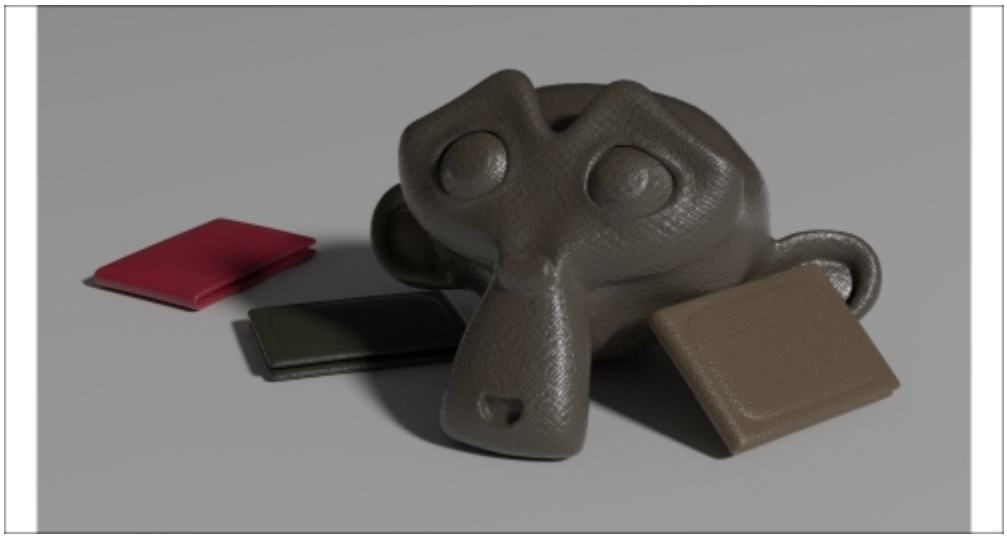
All of these examples are included in the `9931OS_06_cloth.blend` file provided with this module.

See also

To learn more about Blender cloth simulation, you can take a look at the online Blender documentation at http://wiki.blender.org/index.php/Doc:2.6/Manual/Physics/Cloth.

Creating a leather material with procedurals

In this recipe, we will create a leather-like material, as shown in the following screenshot:



The leather-like material assigned to Suzanne and three wallet-like, simple objects

Start Blender and load the `9931OS_06_start.blend` file, where there is already an unwrapped Suzanne mesh.

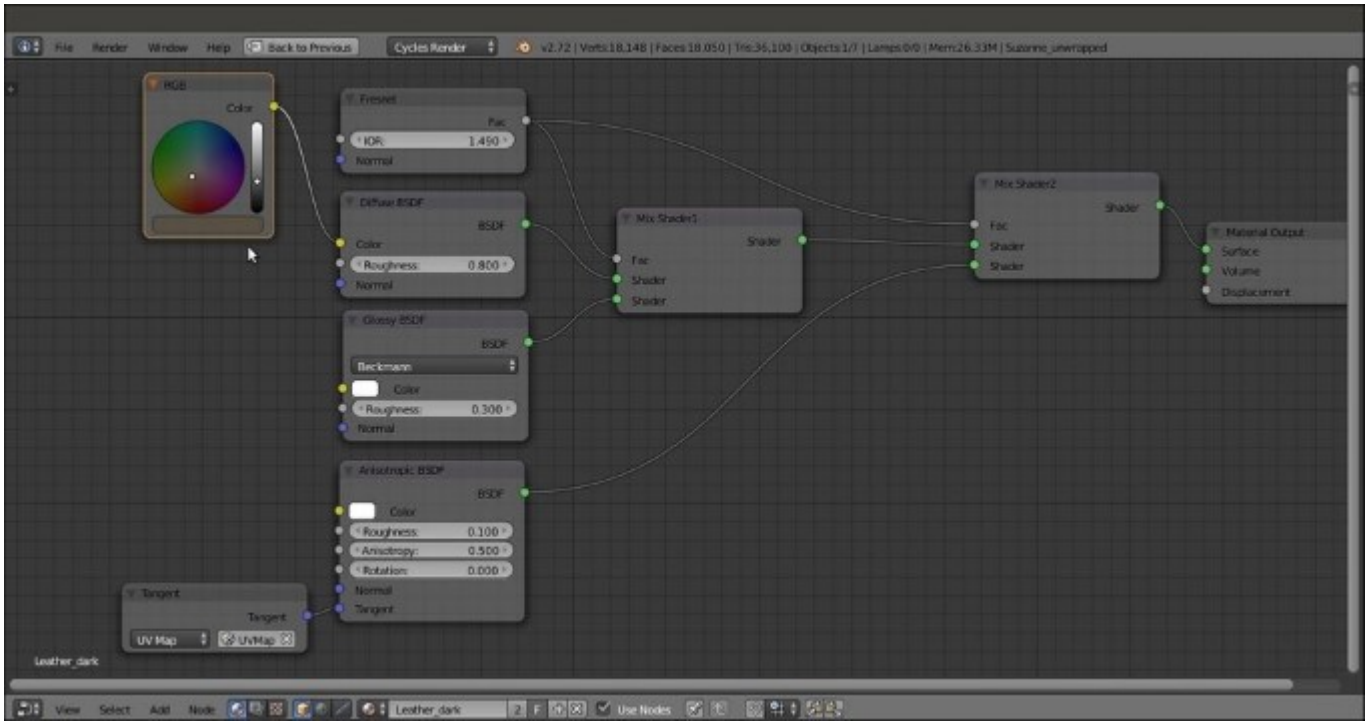
How to do it...

Now we are going to create the material by performing the following steps:

1. Click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** toolbar. Rename the new material `Leather_dark`.
2. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node (label it as `Mix Shader2`). In its first **Shader** slot, select a **Mix Shader** node again (label it as `Mix Shader1`), and in the second slot, load an **Anisotropic BSDF** shader node.
3. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input sockets of both the **Mix Shader** nodes. Set the **IOR** value to `1.490`.
4. Set the **Anisotropic BSDF** node's color to pure white and the **Roughness** value to `0.100`. Add a **Tangent** node (press *Shift* + *A* and navigate to **Input** | **Tangent**). Connect it to the **Tangent** input socket of the **Anisotropic BSDF** shader, and in its **Method to use for the Tangent** slot, select the **UV Map** item. Optionally, click on the blank slot to the right to select the name of the UV layer to be used (this is useful if the mesh has two or more UV layers).
5. Add a **Diffuse BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**) and a **Glossy BSDF** shader node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**). Connect the **Diffuse BSDF** node to the first **Shader** input socket of the **Mix Shader1** node and the **Glossy BSDF** node to the second **Shader** input socket. Set the **Diffuse BSDF** node's

Roughness value to `0.800`. Set the **Glossy BSDF** distribution to **Beckmann**, **Color** to pure white, and its **Roughness** value to `0.300`.

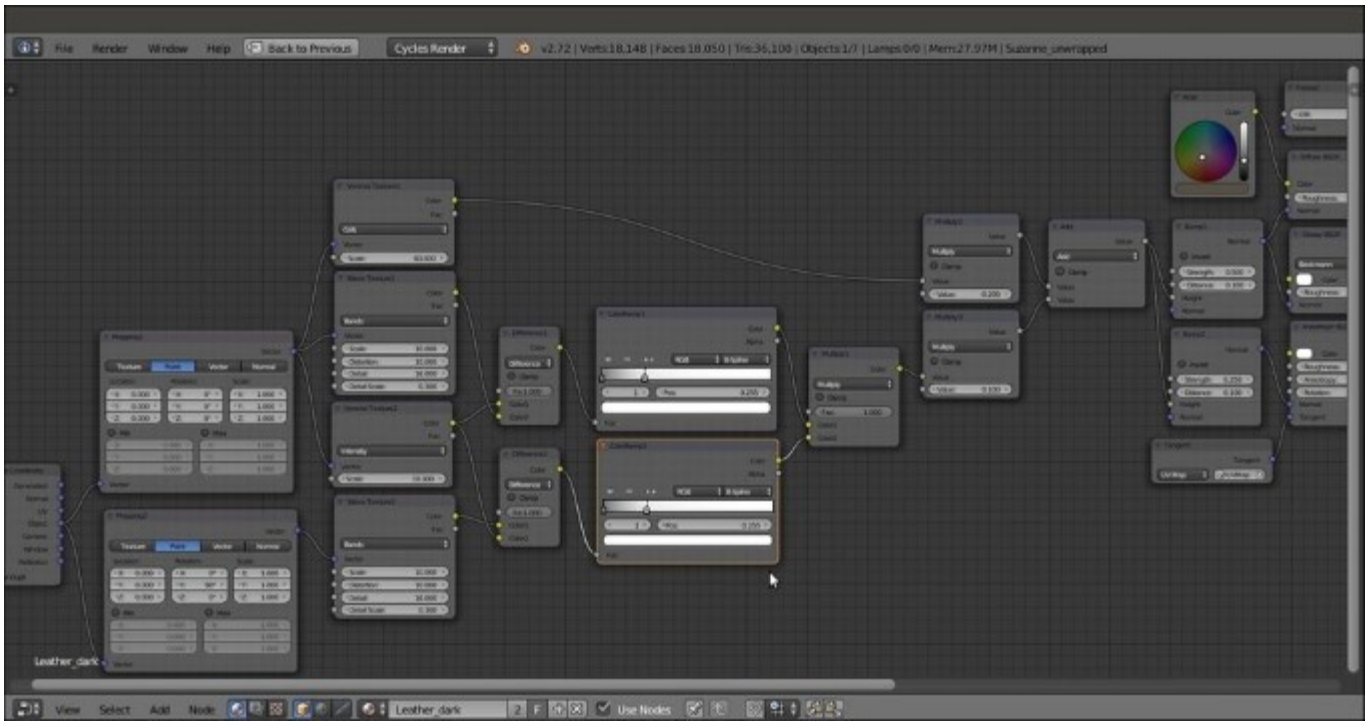
6. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and connect its **Color** output to the **Color** input socket of the **Diffuse BSDF** node. Change the **Color** values of **R** to `0.100`, **G** to `0.080`, and **B** to `0.058`, as shown in the following screenshot:



The shader part of the material

7. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and two **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input sockets of both the **Mapping** nodes (label them as `Mapping1` and `Mapping2`). Then in the **Mapping2** node, change the **Rotation** value of **Y** value to `90°`.
8. Add two **Voronoi Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and two **Wave Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**). Place them in a column next to the **Mapping** nodes in this order from top to bottom: **Voronoi Texture1**, **Wave Texture1**, **Voronoi Texture2**, and **Wave Texture2**.
9. Set the **Voronoi Texture1** node's **Coloring** to **Cells** and the **Scale** value to `60.000`. Go to the **Wave Texture1** node and set the **Scale** value to `10.000`, the **Distortion** value to `10.000`, **Detail** to `16.000`, and **Detail Scale** to `0.300`. Set the **Scale** value of the **Voronoi Texture2** node to `10.000`, and copy and paste the values from the **Wave Texture1** node to the **Wave Texture2** node.
10. Now connect the **Mapping1** node output to the **Vector** input sockets of the two **Voronoi Texture** nodes and the **Wave Texture1** node. Connect the output of the **Mapping2** node to the **Vector** input socket of the **Wave Texture2** node.

11. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set the **Blend Type** to **Difference** and the **Fac** value to `1.000`. Connect the **Color** output of the **Wave Texture1** node to the **Color2** input socket of the **Difference** node, and the **Color** output of the second **Voronoi Texture2** node to its **Color1** input socket.
12. Label the **Difference** node as **Difference1**. Then press *Shift* + *D* to duplicate it. Label the duplicate as `Difference2` and connect the **Color** output of the **Voronoi Texture2** node to its **Color2** input socket. Connect the **Color** output of the **Wave Texture2** node to the **Color1** input socket of the **Difference2** node.
13. Press *Shift* + *D* to duplicate the **Difference** node again, change its **Blend Type** to **Multiply** and label it as `Multiply1`. Connect the output of the **Difference1** node to the **Color1** input socket of the **Multiply1** node. Then connect the output of the **Difference2** node to the **Color2** input socket of the **Multiply1** node.
14. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), label it as **Multiply3**, and connect the output of the **Multiply1** node to the first **Value** input socket. Set the second **Value** input socket to `0.100`.
15. Press *Shift* + *D* to duplicate the **Multiply3** node, label it as `Multiply2`, and connect the **Color** output of the **Voronoi Texture1** node to the first **Value** input socket. Set the second **Value** input socket to `-0.200`.
16. Press *Shift* + *D* to duplicate the **Multiply2** node, label it as `Add`, and change **Operation** to **Add** as well. Connect the output of the **Multiply2** and **Multiply3** nodes to the two **Value** input sockets.
17. Add two **Bump** nodes (press *Shift* + *A* and navigate to **Vector** | **Bump**). Label them as `Bump1` and `Bump2`. Then connect the **Bump1** to the **Normal** input sockets of both the **Diffuse BSDF** and **Glossy BSDF** shader nodes. Connect the **Bump2** node to the **Normal** input of the **Anisotropic BSDF** shader. Set the **Strength** value of the **Bump1** node to `0.500` and the **Strength** value of the **Bump2** node to `0.250`. Connect the output of the **Add** node to the **Height** input sockets of both the **Bump** nodes.
18. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Paste it between the first **Difference1** node and the **Multiply1** node. Set **Interpolation** to **B-Spline** and move the white color stop to the `0.255` position. Label it as `ColorRamp1`.
19. Press *Shift* + *D* to duplicate the **ColorRamp1** node, label it as `ColorRamp2`, and paste it between the **Difference2** node and the **Multiply1** node, as shown in the following screenshot:



The completed network with the added bump pattern

How it works...

- From steps 1 to 6, we built the basic shader for the leather material.
- From steps 7 to 19, we built the bump pattern for the leather. We used two different **Bump** nodes with different values for the **Diffuse BSDF** and **Glossy BSDF** nodes and for the **Anisotropic BSDF** shader, to have slightly different light reflections on the surface.

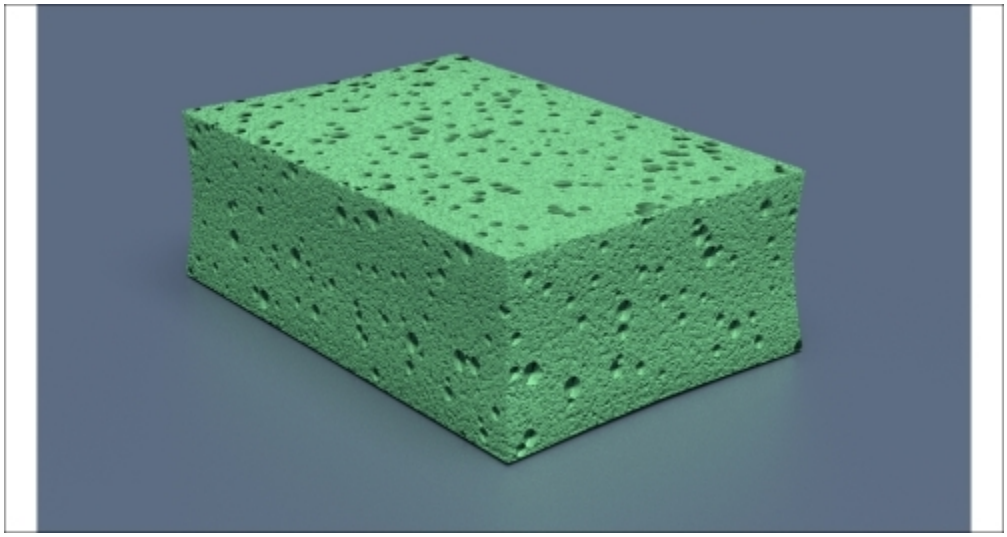
Note

Note that we used the **UV Map** layer information of the mesh for the **Tangent** node to be connected to the **Anisotropic BSDF** shader, and the **Object** mapping node for the bump textures instead.

Actually, we could have used the **UV** mapping output for the texture nodes too, because the mesh had already been unwrapped. However, the scale values for all the nodes in that case would have been double and the flow of the textures on the polygons would have been different (based on the flow of the unwrapped faces in the **UV** window).

Creating a synthetic sponge material with procedurals

In this recipe, we will create a polyurethane sponge material (the type that you usually find in kitchens), as shown in the following screenshot:



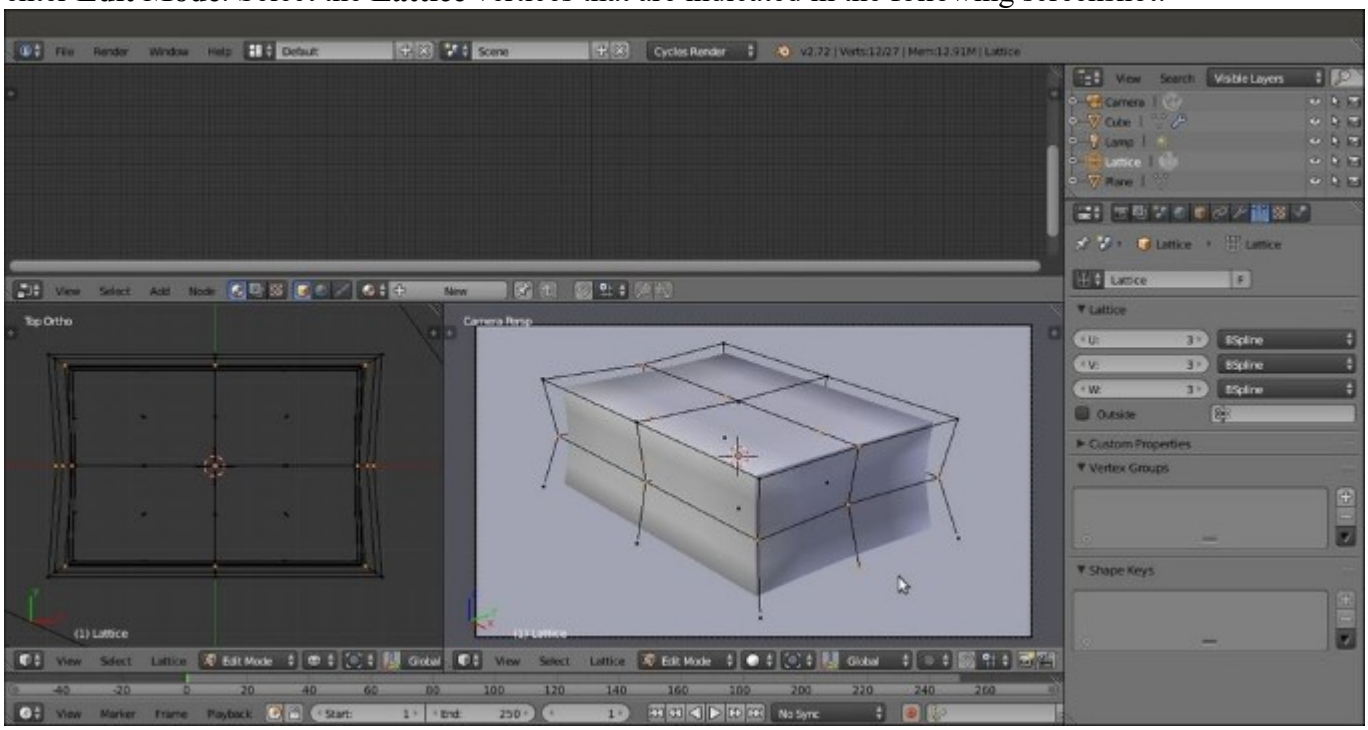
The synthetic sponge material when rendered

Getting ready

Follow these steps to create a synthetic sponge material with procedurals:

1. Start Blender and switch to the **Cycles Render** engine.
2. Select the default Cube, and in the **Transform** subpanel to the right of the 3D viewport (under **Dimensions**), change the values of **X** to `0.350`, **Y** to `0.235`, and **Z** to `0.116`. Press *Ctrl* + *A* to apply the scale.
3. With the mouse arrow in the 3D viewport, add a Plane to the scene (press *Shift* + *A* and navigate to **Mesh** | **Plane**). Exit **Edit Mode**, and in the **Transform** subpanel (the **Dimensions** item), set the values of **X** to `20.000` and **Y** to `20.000`. Press *Ctrl* + *A* to apply the scale. Move the Plane down (press *G*, then press *Z*, enter `-0.05958`, and then press *Enter*) to act as the floor for the sponge.
4. Select the **Lamp** item. In the **Object data** window, click on the **Use Nodes** button and change the type to **Sun**. Set the **Size** to `0.500`, **Color** to pure white, and the **Strength** value to `5.000`. In the **Transform** panel, set the values of **Rotation** value of **X** to `15°`, **Y** to `0°`, and **Z** to `76°`.
5. Select the **Camera** item, and in the **Object data** window under the **Lens** subpanel, change the **Focal Length** value to `60.000`. In the **Transform** subpanel, set the values of the **Location** value of **X** to `0.82385`, **Y** to `-0.64613`, and **Z** to `0.39382`. Change the **Rotation** values of **X** to `68°`, **Y** to `0°`, and **Z** to `51°`.

6. Go to the **World** window and click on the **Use Nodes** button under the **Surface** subpanel. Click on the little square with a dot on the right side of the color slot, and from the menu, select **Sky Texture**. Change **Sky Type** to **Preetham** and set the **Strength** value to `0.100`.
7. Go to the **Render** window, and under the **Sampling** subpanel, set the **Clamp Direct** and **Clamp Indirect** values to `1.00`. Set **Samples** for **Render** and **Preview** to `50`.
8. Split the 3D viewport into two rows. Convert the upper row to a **Node Editor** window. Split the bottom view into two parts and convert the left part to another 3D viewport. With the mouse arrow in the left 3D view, press `0` on the numeric keypad to go to the **Camera** view.
9. Select the **Cube**, and with the mouse arrow in the **Camera** view, press *Shift* + *S*. Navigate to **Cursor to Selected** to place the cursor on the pivot of the Cube (if it's elsewhere). Add **Lattice** to the scene (press *Shift* + *A* and select **Lattice**). Press *Tab* to exit **Edit Mode**. In the **Transform** subpanel, under **Scale**, set the values of **X** to `0.396`, **Y** to `0.264`, and **Z** to `0.129`. Go to the **Object data** window, and in the **Lattice** subpanel set the **U**, **V**, and **W** values to `3`.
10. Reselect the **Cube** and go to the **Object modifiers** window. Assign a **Subdivision Surface** modifier, switch the type of subdivision algorithm from **Catmull-Clark** to **Simple**, and set the **Subdivisions** value to `2` for both **View** and **Render**.
11. Assign a **Bevel** modifier and set the **Width** value to `0.0010`. Assign a **Lattice** modifier, and in the empty **Object** field, select the **Lattice** name. Reselect the **Lattice** object and press *Tab* to enter **Edit Mode**. Select the **Lattice** vertices that are indicated in the following screenshot:



The setup for the sponge scene in Solid viewport shading mode

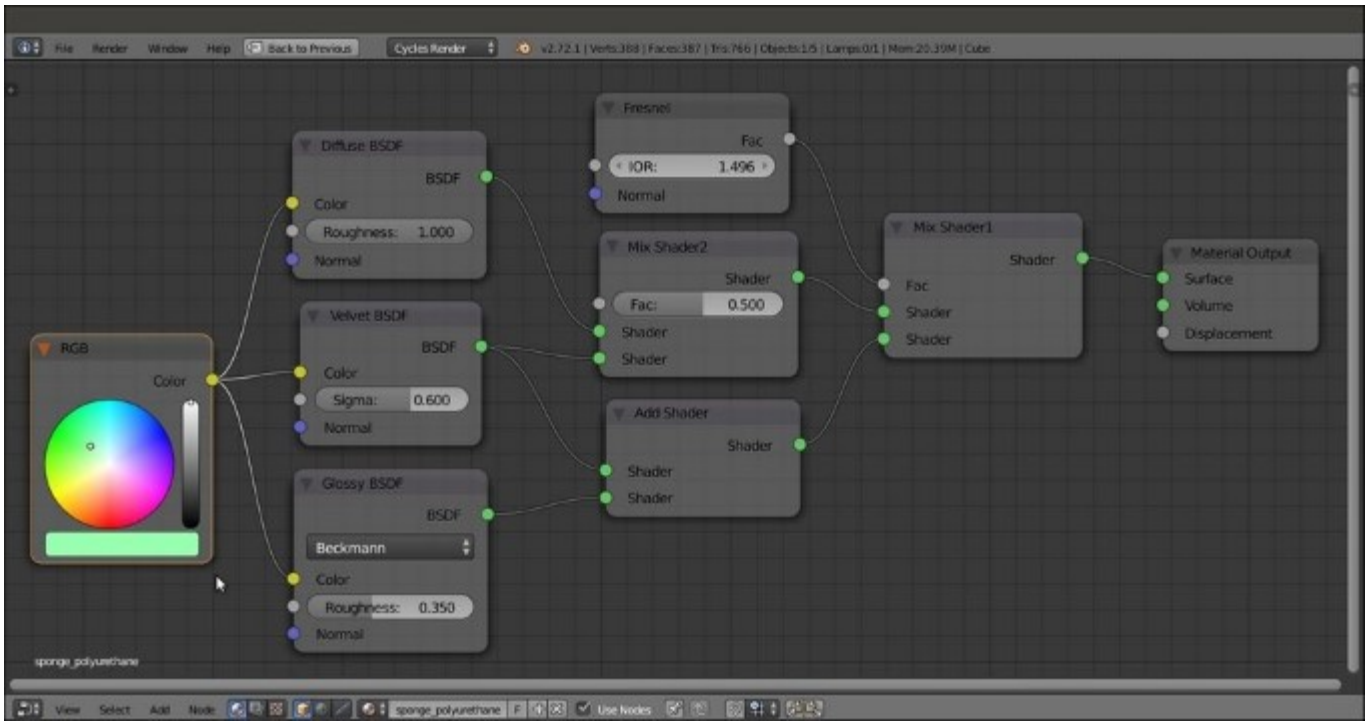
12. Scale the selected vertices on the *x* and *y* axes to slightly smaller values (press *S*, then press *Shift* + *Z*, enter digit *.9*, and press *Enter*). Then scale only the upper vertices to slightly smaller values, and similarly scale other vertices to obtain a shape similar to a kitchen sponge. Then exit **Edit Mode** (press *Tab*).

13. Reselect the **Cube**, navigate to **Tool Shelf** | **Tools** | **Shading**, and select **Smooth**. With the mouse arrow in the 3D view, press *N* and *T* to close the **Transform** and **Tool Shelf** panels. Go to the **Material** window.

How to do it...

Now let's create the material by performing the following steps:

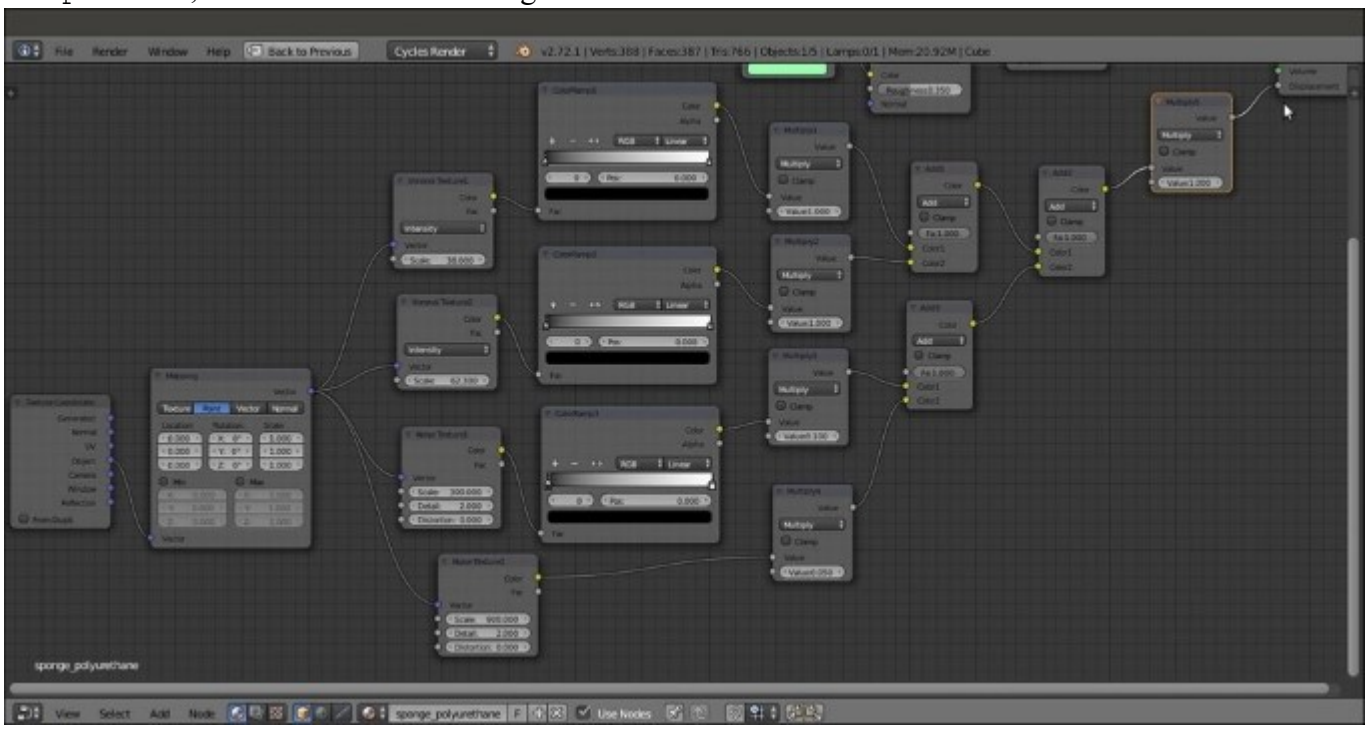
1. First, select the **Plane** and click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** toolbar. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** node, and in the second **Shader** slot, select a **Glossy BSDF** shader node. Set **Distribution** of the **Glossy BSDF** shader to **Beckmann**, the **Fac** value of the **Mix Shader** to `0.400`, and the **Diffuse BSDF** node's **Color** to a shade of blue (in my case, **R** to `0.110`, **G** to `0.147`, and **B** to `0.209`).
2. Now select the **Cube** object and click on **Use Nodes** in the **Material** window under the main **Properties** panel or in the **Node Editor** window's toolbar. Rename the new material `sponge_polyurethane`.
3. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node. In the **Label** slot in the **Node** subpanel under the **Properties** panel of the **Node Editor** window (if this is not present, press the *N* key to make it appear), label it as `Mix Shader1`. Go to the **Material** window, and in the **Mix Shader1** node's first **Shader** slot, select a **Mix Shader** node again. Label it as `Mix Shader2`. In the second **Shader** slot, select an **Add Shader** node.
4. In the first **Shader** slot of the **Mix Shader2** node, select a **Diffuse BSDF** shader node, and in the second slot, a **Velvet BSDF** node. Set the **Diffuse BSDF** node's **Roughness** value to `1.000` and the **Velvet** node's **Sigma** value to `0.600`.
5. Connect the output of the **Velvet** shader to the first **Shader** input of the **Add Shader** node. In its second **Shader** input, load a **Glossy BSDF** shader and set the **Roughness** value to `0.350`.
6. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader1** node. Set the **IOR** value to `1.496`. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and connect its output to the **Color** input sockets of the **Diffuse BSDF**, **Velvet**, and **Glossy BSDF** shader nodes. Set the **RGB** node's **Color** of **R** to `0.319`, **G** to `1.000`, and **B** to `0.435` (any other color is also fine), as shown in the following screenshot:



The basic shader component

7. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), two **Voronoi Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**), and two **Noise Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**). Label the textures as `Voronoi Texture1`, `Voronoi Texture2`, `Noise Texture1`, and `Noise Texture2`.
8. Place the four textures in a row. Then connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the **Vector** output of this node to the **Vector** input sockets of the four texture nodes.
9. Set the **Scale** value of the **Voronoi Texture1** node to `38.000`, the **Voronoi Texture2** node to `62.300`, the **Noise Texture1** node to `300.000`, and the **Noise Texture2** node to `900.000`.
10. Add three **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Label them as `ColorRamp1`, `ColorRamp2`, and `ColorRamp3`. Connect the **Color** output of the **Voronoi Texture1** node to the **Fac** input socket of the **ColorRamp1** node, the **Color** output of the **Voronoi Texture2** node to the **Fac** input socket of the **ColorRamp2** node, and the **Color** output of the **Noise Texture1** node to the **Fac** input socket of the **ColorRamp3** node.
11. Add four **Math** nodes (press *Shift* + *A* and navigate to **Converter** | **Math**). Set **Operation** to **Multiply** and label them as `Multiply1`, `Multiply2`, `Multiply3`, and `Multiply4`. Connect the **Color** output of the three **ColorRamp** nodes to the first **Value** input socket of the first three **Multiply-Math** nodes, and the **Color** output of the **Noise Texture2** node to the first **Value** input socket of the **Multiply4** node. Set the second **Value** input socket of the **Multiply1** and **Multiply2** nodes to `1.000`, the second **Value** input socket of the **Multiply3** node to `0.100`, and the second **Value** input socket of the **Multiply4** node to `0.050`.

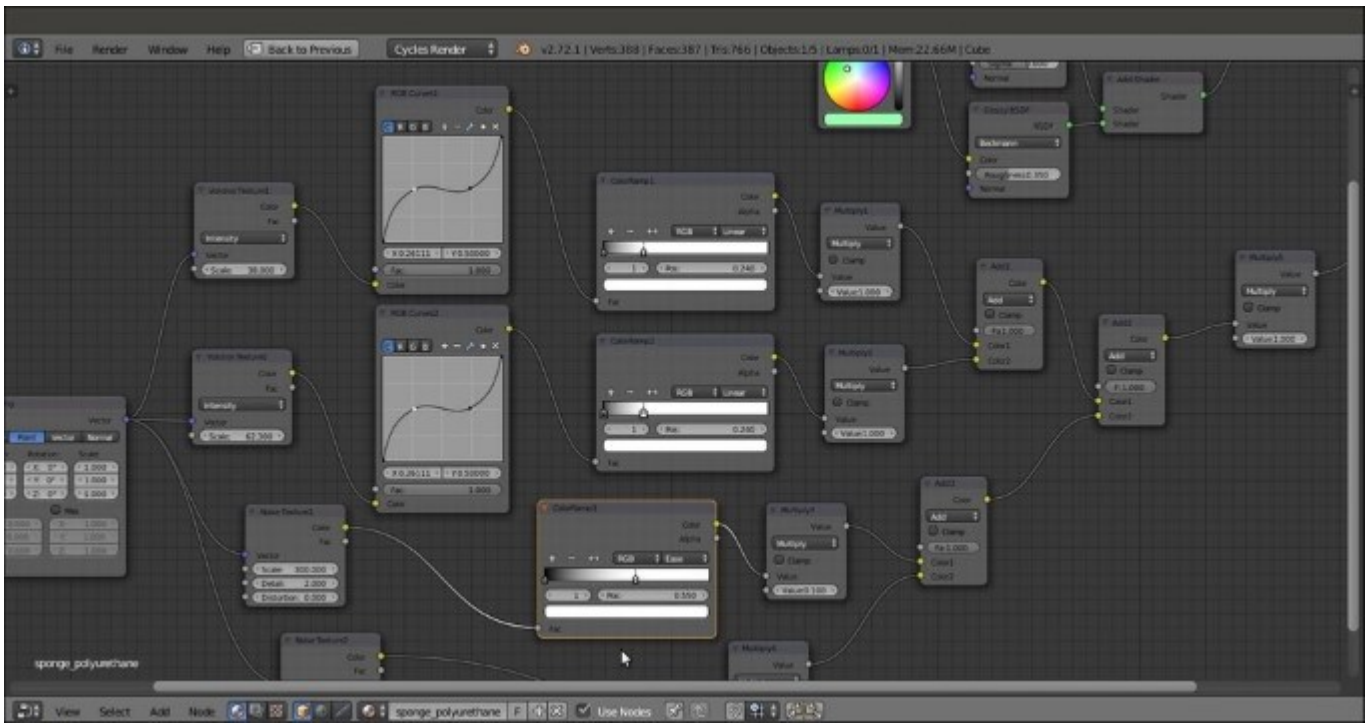
12. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Connect the output of the **Multiply1** node to the **Color1** input socket and the output of the **Multiply2** node to the **Color2** input socket. Change **Blend Type** to **Add** and the **Fac** value to `1.000`. Label the **Add-MixRGB** node as `Add1`.
13. Press *Shift* + *D* to duplicate the **Add1** node, and label it as `Add2`. Connect the output of the **Add1** node to the **Color1** input socket. Then connect the output of the **Multiply3** node to the **Color2** input socket of the **Add2** node.
14. Press *Shift* + *D* to duplicate the **Add2** node, and label it as `Add3`. Paste it between the **Multiply3** and **Add2** nodes, and connect the output of the **Multiply4** node to the **Color2** input socket of the **Add3** node.
15. Add a new **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), set **Operation** to **Multiply**, and label it as `Multiply5`. Connect the output of the **Add2** node to the first **Value** input socket of the **Multiply5** node, and set the second **Value** input socket to `1.000`.
16. Connect the output of the **Multiply5** node to the **Displacement** input socket of the `Material Output` node, as shown in the following screenshot:



The bump pattern

17. Now box-select (press the *B* key and then click and drag the selection to enclose the objects) the **ColorRamp1**, **ColorRamp2**, **Multiply1**, **Multiply2**, **Add1**, **Add2**, and **Multiply5** nodes. Press *G* and move them to the right to make room for new nodes on the left side.
18. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**) and paste it between the **Voronoi Texture1** node and the **ColorRamp1** node. Label it as `RGB Curves1`. Click on the curve to add a control point, and in the coordinate slots below the node window, set the **X** value to `0.26111` and the **Y** value to `0.50000`. Click to add a second control point. Set **X** to `0.73889` and **Y** to `0.51111`.

19. Press *Shift* + *D* to duplicate the **RGB Curves1** node. Paste it between the **Voronoi Texture2** and **ColorRamp2** nodes. Label it as `RGB Curves2`.
20. Go to the **ColorRamp1** node and move the white color stop to the `0.240` position. Then go to the **ColorRamp2** node and repeat this step. Next, go to the **ColorRamp3** node, move the white color stop to the `0.550` position, and set **Interpolation** to **Ease**, as shown in the following screenshot:



Tweaking the bump pattern

How it works...

- From steps 2 to 6, we built the basic shader for the sponge material and the color. As you can see in the **Rendered** camera view, without the bump pattern, there is a visible artifact in the more distant side of the mesh. This is due to the **Smooth** shading we set in step 13 of the *Getting ready* section. Setting the shading to **Flat** again would remove the artifact, but would also show the blocky faces of the deformed sponge mesh. In this case, because of the bump pattern and the fact that the mesh is subdivided, this is not a major issue, and both solutions (smooth but with artifacts or flat but blocky) are fine.
- From steps 7 to 20, we built the sponge bump pattern by mixing **Voronoi Texture** nodes at different sizes, with increased contrast due to the **ColorRamp** nodes. Then we add some noise to avoid a highly smoothed surface.

Creating a spaceship hull shader

In this recipe, we will create a spaceship hull material. We will add random, tiny light windows based on the values of procedural textures, and the spaceship's logo as if it were painted in red on the hull, as shown in the following screenshot:

The final, rendered spaceship hull material assigned to a displaced Torus primitive

Getting ready

To start creating this spaceship hull, we need the spaceship and space first. Follow these steps to build a quick and easy model and set up the scene:

1. Start Blender and switch to the **Cycles Render** engine. Select the default **Cube** and delete it.
2. With the mouse arrow in the 3D view, press *Shift* + *A* and add a **Torus** primitive (press *Shift* + *A* and navigate to **Mesh** | **Torus**). In **Edit Mode**, scale it to at least twice its current size (press *A* to select all the vertices, then type *S*, enter *2*, and press *Enter*).
3. Exit **Edit Mode**, and in **Outliner**, select the **Lamp** object. In the **Object data** window, change it to **Sun**. Then set the **Size** value to `0.050`. Click on the **Use Nodes** button and set the **Strength** value to `10.000`. Change the **Color** value of **RGB** to `0.800`.
4. Go to the **World** window and click on the **Use Nodes** button under the **Surface** subpanel. Click on the little square with a dot to the right of the **Color** slot. From the menu, select **Sky Texture**. Set the **Strength** to `0.100`.
5. Select the **Camera**, and in the **Transform** panel, set the **Location** values of **X** to `6.10677`, **Y** to `-0.91141`, and **Z** to `-2.16840`. Set the **Rotation** values of **X** to `112.778°`, **Y** to `-0.003°`, and **Z** to `81.888°`.
6. Go to the **Render** window, and under the **Sampling** subpanel, set the **Samples** to `25` for **Preview** and `100` for **Render**. Then set the **Clamp Indirect** value to `1.00`, but let the **Clamp**

Direct value remain as `0.00`. Go to the **Film** subpanel and check the **Transparent** item. Then set the output **File Format** to **RGBA**.

7. Under the **Light Paths** subpanel, disable both the **Reflective** and **Refractive Caustics** items. Set **Filter Glossy** value to `1.00`. Under the **Performance** subpanel, set the **Viewport BVH Type** to **Static BVH** (this should speed up the rendering a bit, considering the fact that the model is static and doesn't change shape). Check the **Persistent Images** and **Use Spatial Splits** items.
8. Press *N* with the mouse arrow in the 3D view to close the **Properties** panel. Then press *T* to get rid of the **Tool Shelf** panel. Split the 3D view into two rows. Convert the upper row to a **Node Editor** window.
9. Split the bottom window into two parts. Convert the left part to a **UV/Image Editor** window. Select **Torus** and press *Tab* to go to **Edit Mode**. In the window toolbar, change the selection mode to **Face select**. Select only one face on the mesh (whichever you prefer). Press the *A* key twice to select all the faces, and keep the first face selected as the active face. Then press the *U* key. In the **UV Mapping** pop-up menu, select the **Follow Active Quads** item, and then in the next pop-up menu set **Even** as **Edge Length Mode**. Click on the **OK** button.
10. With the mouse arrow in the **UV/Image Editor** window, press *A* to select all the vertices of the UV islands. Then scale them to one-third of their current size (press *S*, enter digit *.3*, and press *Enter*). Press *Tab* to exit **Edit Mode**, and change **UV/Image Editor** to **3D View**. Convert the right 3D viewport to a **Camera** view by pressing the *0* key on the numeric keypad (with the mouse arrow in the 3D view).
11. Go to the **Object modifiers** window and assign a **Subdivision Surface** modifier to **Torus**. Set the **Subdivisions** level to `4` for both **View** and **Render**. Set the **Camera** view mode to **Rendered** and go to the **Material** window.

How to do it...

Now let's start creating the material. The steps to create a basic hull shader are as follows:

1. Click on **New** in the **Material** window under the main **Properties** panel or in the **Node Editor** toolbar. Rename the new material `spacehull`.
2. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node. Label it as `Mix Shader1`. In the first **Shader** slot, select a new **Mix Shader** node (label it as `Mix Shader2`), and in the second slot, select an **Anisotropic BSDF** node.
3. In the first **Shader** slot of the **Mix Shader2** node, select a **Diffuse BSDF** node, and in the second slot, select a **Glossy BSDF** node. Set the **Distribution** of both the **Glossy BSDF** and **Anisotropic BSDF** nodes to **Ashikmin-Shirley**.
4. In the **Anisotropic BSDF** shader node, set the **Rotation** value to `0.250`. In the **Diffuse BSDF** node, set the **Roughness** value to `0.500`.
5. Add a new `Mix Shader` node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Mix Shader1** and the **Material Output** nodes. Label it as `Mix Shader_Spec_Amount` and connect the output of the **Diffuse BSDF** node to the first **Shader** input socket (so that the link from the **Mix Shader2** node automatically switches to the second socket). Set the **Fac** value to `0.300`.

6. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**). Connect its output to the **Fac** input sockets of the **Mix Shader1** and **Mix Shader2** nodes. Set the **IOR** value to `100.000`.
7. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select all the nodes except the **Material Output** node. Then select the **Frame** and press *Ctrl* + *P* to parent all of them. Label the frame as `SHADER`, as shown in the following screenshot:

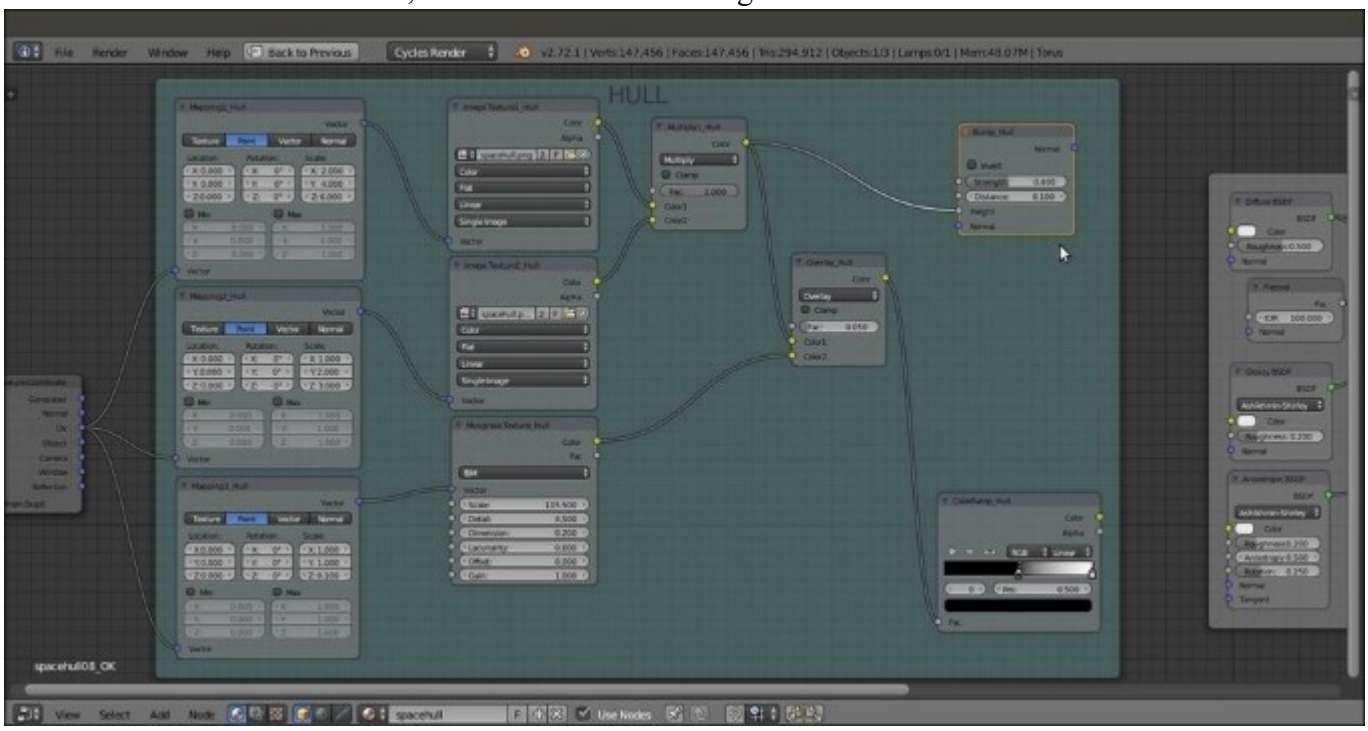
The SHADER frame

The steps to create hull's panels are as follows:

8. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), two **Image Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), and one **Musgrave Texture** node (press *Shift* + *A* and navigate to **Texture** | **Musgrave Texture**).
9. Label the textures as `Image Texture1_Hull`, `Image Texture2_Hull`, and `Musgrave Texture_Hull`. Place them in a column.
10. Press *Shift* + *D* to duplicate the **Mapping** node twice. Label the nodes as `Mapping1_Hull`, `Mapping2_Hull`, and `Mapping3_Hull`. Place them in a column to the left of the texture nodes. Connect the UV output of the **Texture Coordinate** node to the **Vector** input sockets of the three **Mapping** nodes. Connect the **Vector** output of each of the **Mapping** nodes to the **Vector** input socket of each of the texture nodes.
11. Click on the **Open** button in the **Image Texture1_Hull** node to load `image spacehull.png`. Then click on the little arrows to the left of the **Open** button in the **Image Texture2_Hull** node to select the same image texture. Go to the **Musgrave Texture** node and

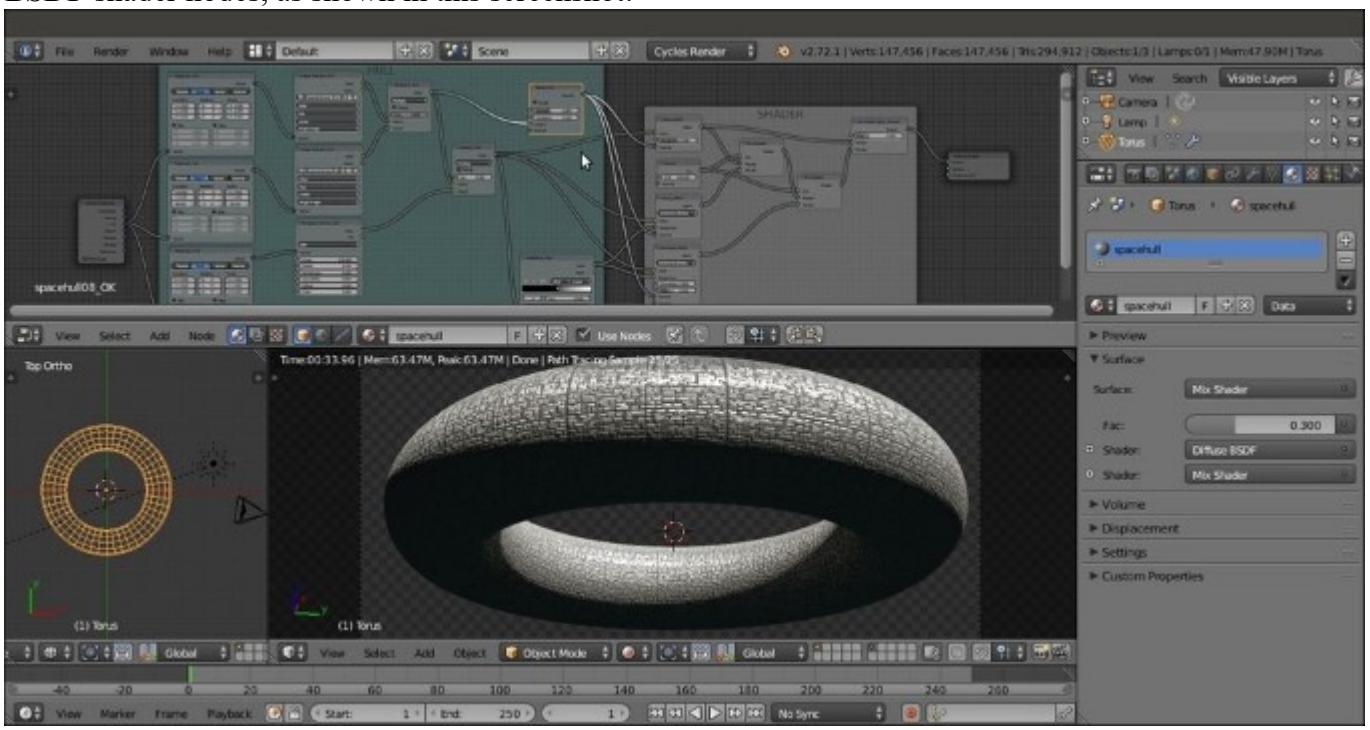
set the **Scale** value to `115.500`, the **Detail** value to `4.500`, the **Dimension** value to `0.200`, and the **Lacunarity** value to `0.600`.

12. Go to the **Mapping1_Hull** node and set the **Scale** value of **X** to `2.000`, **Y** to `4.000`, and **Z** to `6.000`. Then go to the **Mapping2_Hull** node and set the **Scale** value of **Y** to `2.000` and **Z** to `3.000`. Next, go to the **Mapping3_Hull** node and set the **Scale** value of **Z** to `0.100`.
13. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set **Blend Type** to **Multiply** and the **Fac** value to `1.000`. Label it as `Multiply1_Hull`. Then connect the **Color** output of the **Image Texture1_Hull** node to the **Color1** input socket and the **Color** output of the **Image Texture2_Hull** node to the **Color2** input socket.
14. Press *Shift* + *D* to duplicate the **Multiply1_Hull** node, change **Blend Type** to **Overlay**, and label it as `Overlay_Hull`. Set the **Fac** value to `0.050`. Connect the output of the **Multiply1_Hull** node to the **Color1** input socket and the **Color** output of the **Musgrave Texture** node to the **Color2** input socket.
15. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp_Hull`, and connect the output of the **Overlay_Hull** node to its **Fac** input socket. Move the black color stop to the `0.500` position.
16. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**), label it as `Bump_Hull`, and connect the output of the **Multiply1_Hull** node to the **Height** input socket. Set the **Strength** value to `0.400`.
17. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the three **Mapping** nodes, the two **Image Texture** nodes, the **Musgrave Texture** node, the **Bump** node, the **ColorRamp** node, the two **MixRGB** nodes, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `HULL`, as shown in the following screenshot:



The HULL frame

18. Connect the **Normal** output of the **Bump_Hull** node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Anisotropic BSDF** nodes inside the **SHADER** frame. Then connect the output of the **Overlay_Hull** node to the **Color** input sockets of the same **Diffuse BSDF**, **Glossy BSDF**, and **Anisotropic BSDF** nodes. Next, connect the **Color** output of the **ColorRamp_Hull** node to the **Roughness** input sockets of the **Glossy BSDF** and **Anisotropic BSDF** shader nodes, as shown in this screenshot:



The output of the HULL frame connected to the SHADER frame nodes

The steps to create hull's logo are as follows:

19. Add a new **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**) and a new **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**). Label them as `Mapping4_Name` and `Image Texture3_Name`, respectively. Connect the UV output of the **Texture Coordinate** node to the **Mapping4_Name** node, and the output of this node to the **Vector** input socket of the **Image Texture_Name** node.
20. Click on the **Open** button of the **Image Texture** node and load the `spacehull_name.png` image, an image texture of the ARGUS logo with a transparent background (alpha channel).
21. Go to the **HULL** frame and add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Label it as `Mix_Hull_Name` and paste it between the **Overlay_Hull** node and the **Diffuse BSDF** shader node. Then connect its **Color** output to the **Color** input socket of the **Glossy BSDF** and **Anisotropic BSDF** shader nodes.
22. Connect the **Color** output of the **Image Texture_Name** node to the **Color2** input socket of the **Mix_Hull_Name** node. Then connect the **Alpha** output of the **Image Texture_Name** node to the **Fac** input socket of the **Mix_Hull_Name** node.

23. Go to the **Mapping4_Name** node and check both the **Min** and **Max** items. Then set the **Location** value of **X** to `-3.300` and **Y** to `1.000`. Set the **Scale** value of **Y** to `2.500`. (These values depend on the scale and location you want for your logo on the spaceship; just experiment looking at the real-time-rendered preview.)
24. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the **Mapping4_Name** node, the **Image Texture3_Name** node, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as `NAME`, as shown in the following screenshot:

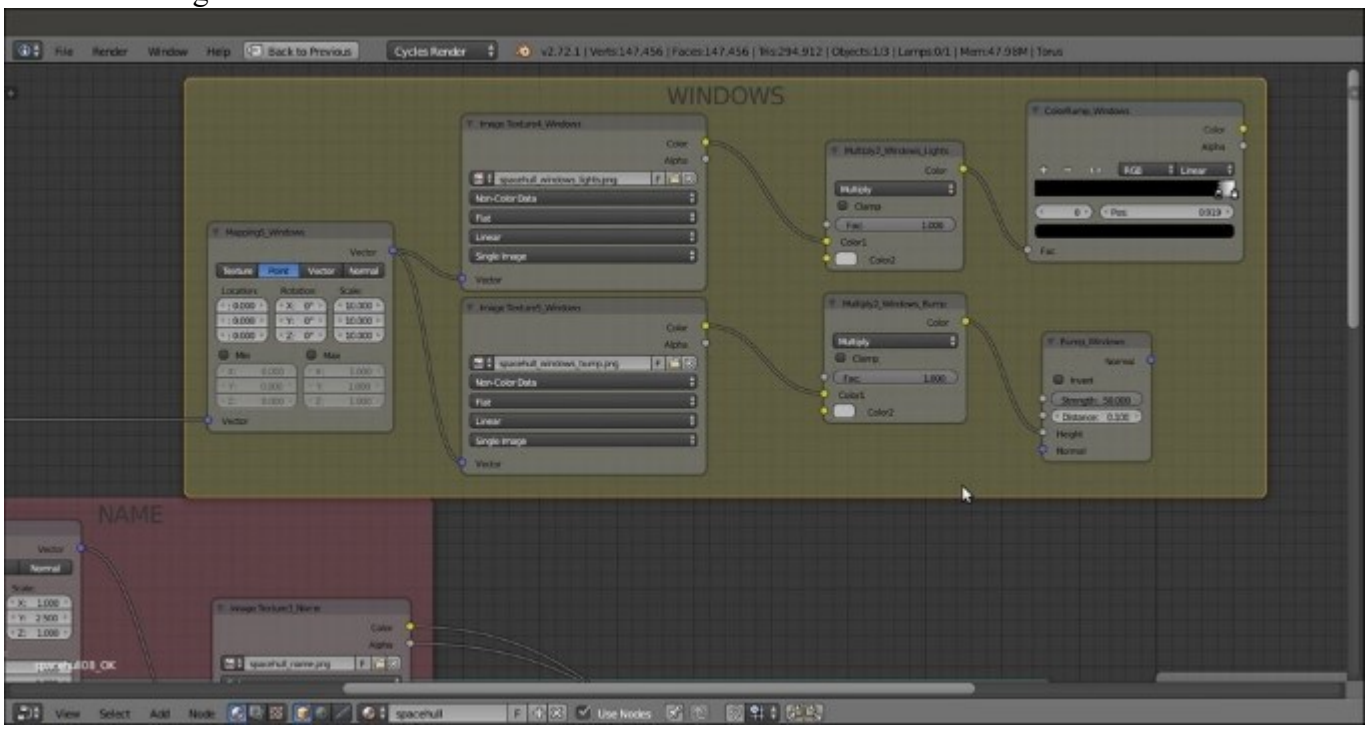
The ARGUS logo on the hull

The steps to create the windows are as follows:

25. Add a new **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**) and two **Image Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Image Texture**). Label them as `Mapping5_Windows`, `Image Texture4_Windows`, and `Image Texture5_Windows`. Connect the **Texture Coordinate** node's **UV** output and the **Mapping** node's output to the **Image Texture** nodes as usual. Then set the **Mapping** node's **Scale** values to `10.000` for the three axes.
26. Click on the **Open** button of the **Image Texture4_Windows** node and load the `spacehull_windows_lights.png` image. Then click on the **Open** button of the **Image Texture5_Windows** node and load the `spacehull_windows_bump.png` image. Set **Color Space** for both the image nodes to **Non-Color Data**.
27. Add two **MixRGB** nodes (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set **Blend Type** to **Multiply** and **Fac** values to `1.000`. for both the nodes Label them as `Multiply2_Windows_Light` and `Multiply2_Windows_Bump`. Connect the output of the **Image Texture4_Windows** node to the **Color1** input socket of the

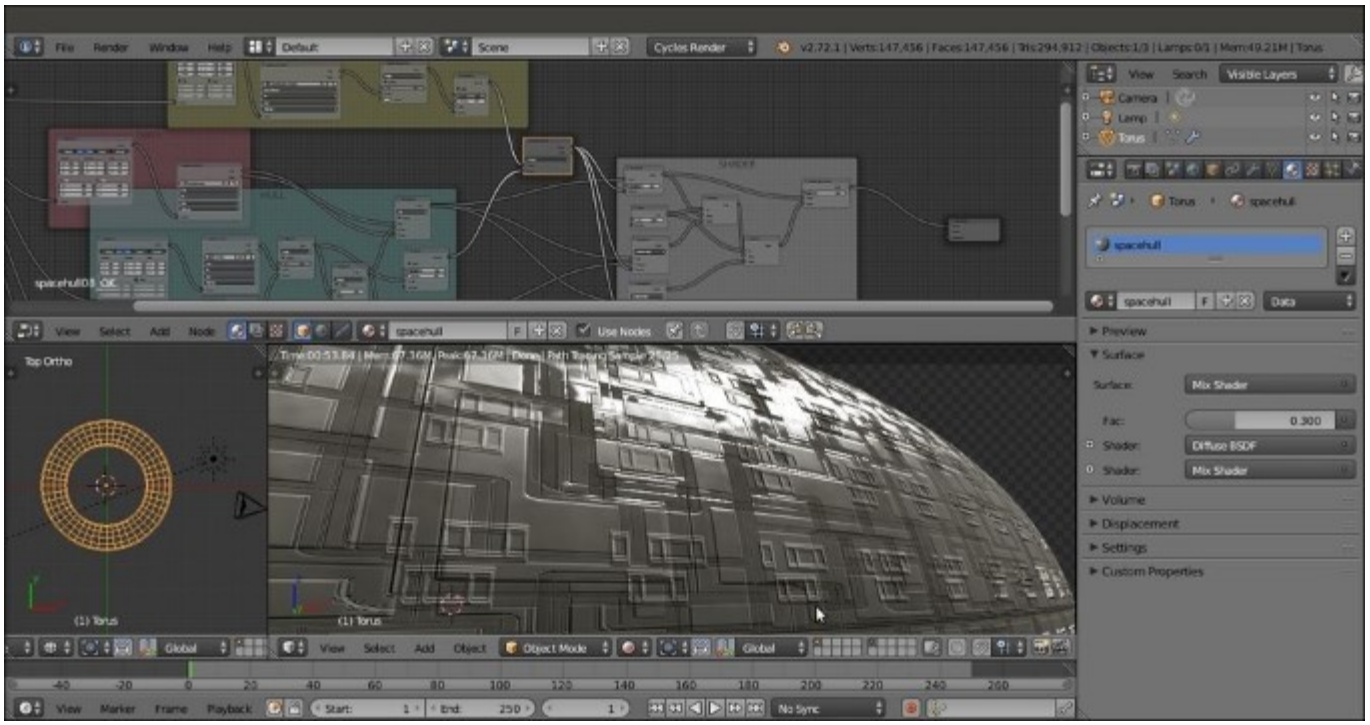
Multiply2_Windows_Light node, and the output of the **Image Texture5_Windows** node to the **Color1** input socket of the **Multiply2_Windows_Bump** node.

28. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp_Windows`, and move the black color stop to the `0.919` position. Connect the output of the **Multiply2_Windows_Light** node to its **Fac** input socket.
29. Add a new **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**), label it as `Bump_Windows`, and connect the output of the **Multiply2_Windows_Bump** node to the **Height** input socket. Set the **Strength** value to `50.000`.
30. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* and select the **Mapping5_Windows** node, the **Image Texture4_Windows** and **Image Texture5_Windows** nodes, the two **MixRGB** nodes, the **ColorRamp_Windows** and the **Bump_Windows** nodes, and then the **Frame**. Press *Ctrl* + *P* to parent them. Label the frame as **WINDOWS**, as shown in the following screenshot:



The WINDOWS frame

31. Add a **Vector Math** node (press *Shift* + *A* and navigate to **Converter** | **Vector Math**) and set **Operation** to **Average**. Connect the **Normal** output of the **Bump_Windows** node inside the **WINDOWS** frame to the first **Vector** input socket, and the **Normal** output of the **Bump_Hull** node inside the **HULL** frame to the second **Vector** input socket. Then connect the **Normal** output of the **Average Bump_Hull** node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Anisotropic BSDF** shader nodes, as shown in this screenshot:

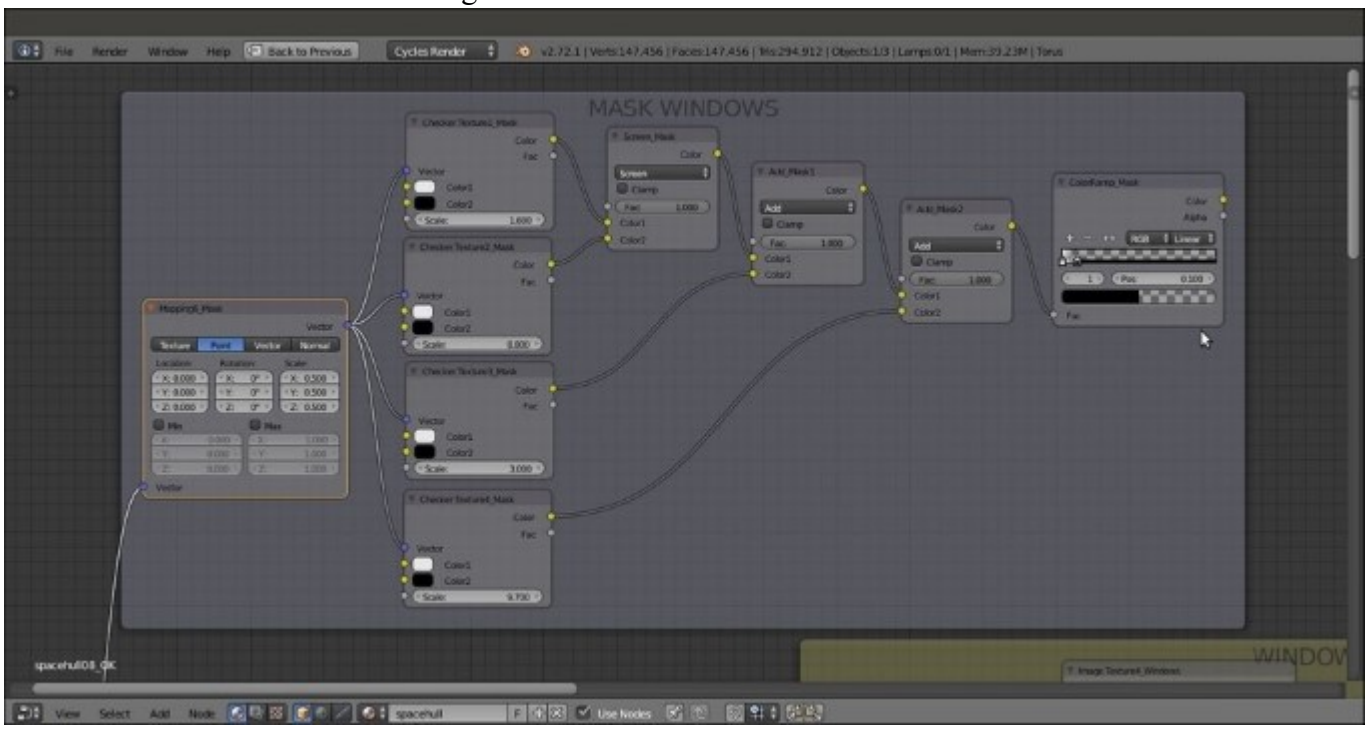


The windows bump visible on the hull

The steps to create the location mask for the windows are as follows:

32. Add one more **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**) and four **Checker Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Checker Texture**). Connect the **Texture Coordinate** node and the nodes as usual. Then label them as `Mapping6_Mask`, `Checker Texture1_Mask`, `Checker Texture2_Mask`, `Checker Texture3_Mask`, and `Checker Texture4_Mask`. In all, the four **Checker Texture** nodes change **Color2** to pure black.
33. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set **Blend Type** to **Screen** and **Fac** value to `1.000`, and label it as `Screen_Mask`. Connect the **Color** output of the **Checker Texture1_Mask** node to the **Color1** input socket of the **Screen_Mask** node, and the **Color** output of the **Checker Texture2_Mask** node to the **Color2** input socket.
34. Press *Shift* + *D* to duplicate the **MixRGB** node, change the duplicate node's **Blend Type** to **Add**, and label it as `Add_Mask1`. Connect the output of the **Screen_Mask** node to the **Color1** input socket. Then connect the **Color** output of the **Checker Texture3_Mask** node to the **Color2** input socket.
35. Press *Shift* + *D* to duplicate the **Add_Mask1** node, and label the duplicate as `Add_Mask2`. Connect the output of the **Add_Mask1** node to the **Color1** input socket. Then connect the **Color** output of the **Checker Texture4_Mask** node to the **Color2** input socket.
36. Add a **ColorRamp** node and label it as `ColorRamp_Mask`. Connect the output of the **Add_Mask2** node to its **Fac** input socket. Then move the black color stop to the `0.100` position and the white color stop to the `0.000` position. Set **Alpha** of the black color stop to `0.000`.

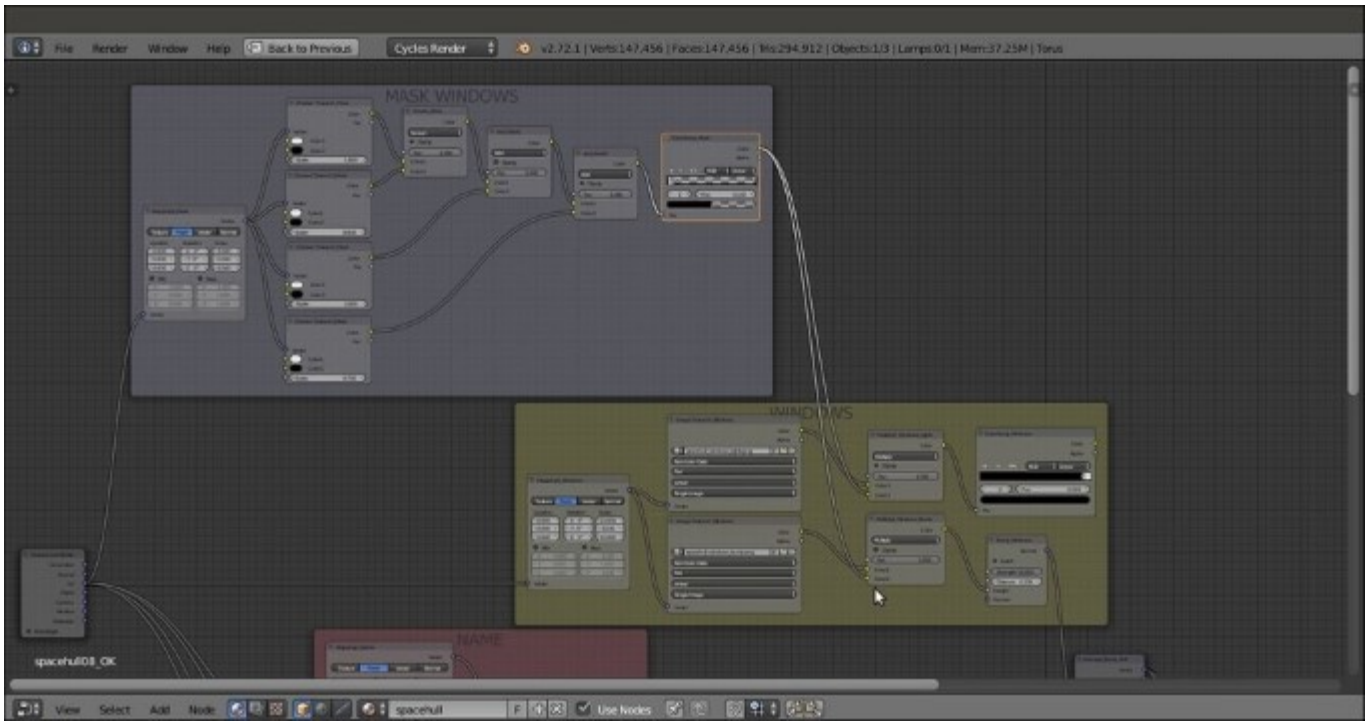
37. Go to the **Checker Texture** nodes. Set the **Scale** value for the **Checker Texture1_Mask** node to `1.600`, the **Checker Texture2_Mask** node to `8.800`, the **Checker Texture3_Mask** node to `3.000`, and the **Checker Texture4_Mask** node to `9.700`. Go to the **Mapping6_Mask** node and set the **Scale** values to `0.500` for all the three axes.
38. Add a **Frame** (press *Shift* + *A* and navigate to **Layout** | **Frame**). Press *Shift* to select the recently added nodes and then the **Frame**. Press *Ctrl* + *P* to parent them. Rename the frame as `MASK WINDOWS` as shown in the following screenshot:



The MASK WINDOWS frame

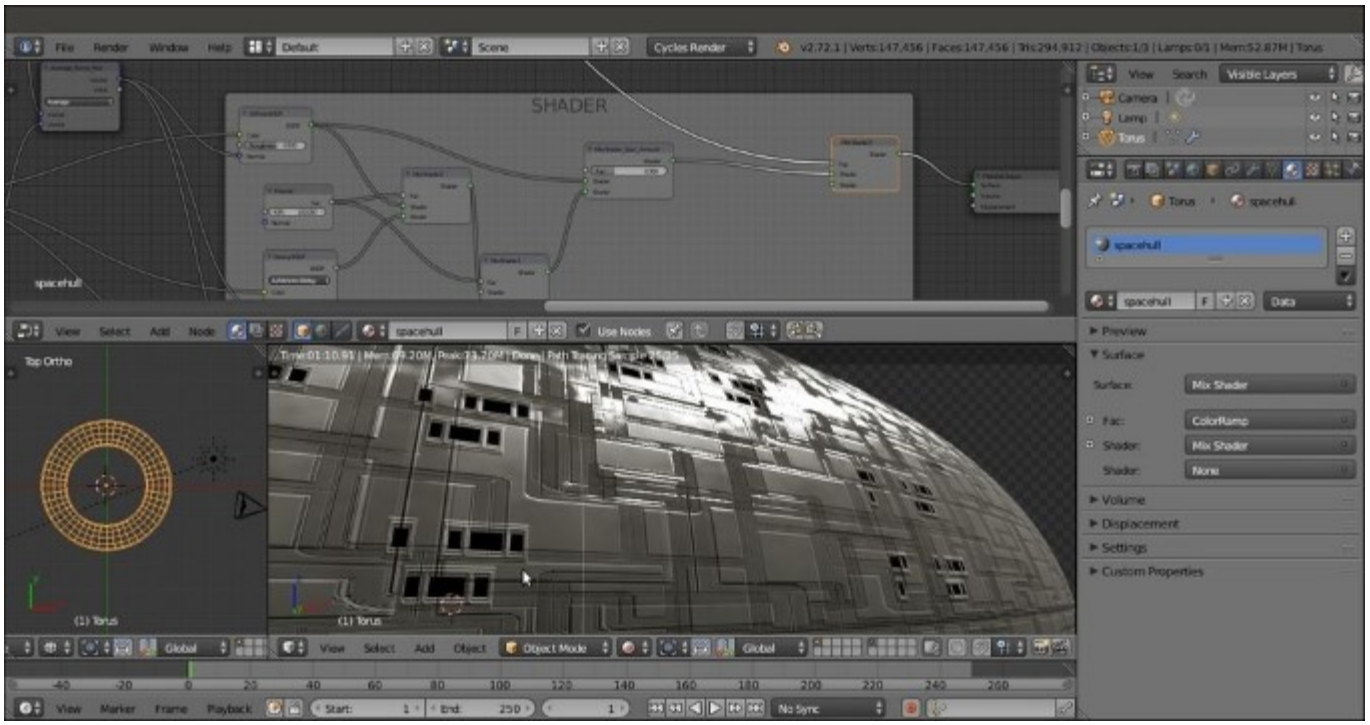
The steps to create the final connections are as follows:

39. Connect the **Color** output of the **ColorRamp_Mask** node to the **Color2** input sockets of both the **Multiply2_Windows_Lights** and **Multiply2_Windows_Bump** nodes, as shown in this screenshot:



The MASK WINDOWS frame output connected to the WINDOWS frame nodes

40. Go to the **SHADER** frame and add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader | Mix Shader**). Label it as `Mix Shader3` and paste it between the **Mix Shader_Spec_Amount** and the **Material Output** nodes.
41. Connect the **Color** output of the **ColorRamp_Windows** node inside the **WINDOWS** frame to the **Fac** input socket of the **Mix Shader3** node, as shown in the following screenshot:



The output of the WINDOWS frame connected to the SHADER frames nodes and the result in the Rendered preview

The steps to create the light emitter for the windows are as follows:

42. Inside the **SHADER** frame, add an **Emission** shader node (press *Shift* + *A* and navigate to **Shader** | **Emission**), a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), and an **Object Info** node (press *Shift* + *A* and navigate to **Input** | **Object Info**). Label the **ColorRamp** node as `ColorRamp_Lights_Colors`, change **Interpolation** to **Constant**, and add six more color stops (eight total). Change the color values alternatively of **R** to `0.800`, **G** to `0.517`, **B** to `0.122`; and **R** to `0.800`, **G** to `0.198`, and **B** to `0.040` (or any other color you prefer).
43. Connect the **Random** output of the **Object Info** node to the **Fac** input socket of the **ColorRamp** node, and the output of this node to the **Color** input socket of the **Emission** node.
44. Connect the **Emission** node's output to the second **Shader** input socket of the **Mix Shader3** node. Then set **Strength** to `3.000`, as shown in the following screenshot:

The windows on the hull getting illuminated by the ColorRamp_Lights_Colors node and an Emission node output connected to the SHADER output

How it works...

- From step 1 to step 7, we built the general shader for the metallic hull, which is similar to the metal node group we saw in Chapter 4, *Creating Man-made Materials in Cycles*. This was achieved by mixing **Diffuse BSDF** and **Glossy BSDF** shaders with an **Anisotropic BSDF** node on a ground with a quite high **IOR** value (`100.000`), and through the usual **Mix Shader** nodes. We added one more **Mix Shader** node (**Mix Shader_Spec_Amount**) to include the possibility of setting more specularity than anisotropy, and vice versa.
- From step 8 to step 18, we built the **HULL** frame group by superimposing two differently scaled versions of the same image. Then they could be used for the color, bump, and specular components. These components were obtained by contrasting the paneling through a **ColorRamp** node and then going straight to the **Glossy BSDF** shader's roughness and the **Anisotropic BSDF** shader, to add a metallic look. The mixture of both **Glossy BSDF** and **Anisotropic BSDF** is made on the ground of a **Fresnel** node set to `100.000`. A very high value like this is needed because the specularity is then mixed again with the **Diffuse BSDF** component to the purpose to obtain a slider to tweak the effect.
- From step 19 to step 24, we added the red hull's logo, ARGUS, using its own alpha channel to overimpose it on the hull surface's panels..
- From step 25 to step 30, we built the **WINDOWS** frame group.
- In step 31, we merged (averaged) the bump effect of the windows with the bump effect of the hull's panels.

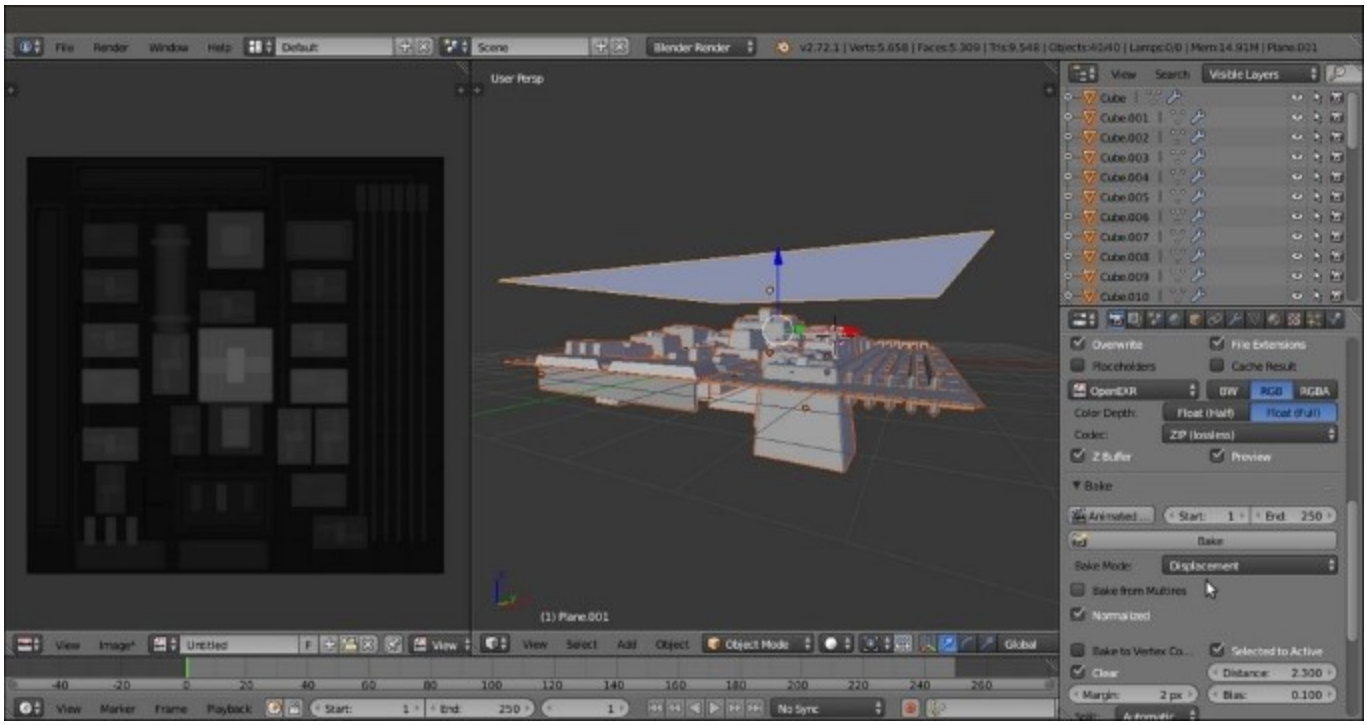
- From step 32 to step 38, we made the masking for the windows to give them a random appearance.
- From step 39 to step 41, we simply connected the various frames' output.
- From step 42 to step 44, we created the light-emitting material for the windows. Note that the **WINDOWS_ MASK** frame group provides the masking for the windows' positions. The **WINDOW** frame group provides the whiteness values for the windows, the bump, and the last nodes added to the **SHADER** frame the light emission based on the output of the previous frame groups.

There's more...

The appearance of the hull can be improved even further using some displacement to add geometric details to the spaceship surface, which (at the moment) is a bit too smooth:

1. Go to the **Object modifiers** window and assign a second **Subdivision Surface** modifier to **Torus**. Set the **Subdivisions** levels to `2` for both **View** and **Render**.
2. Assign a **Displace** modifier. Then click on the **Show texture in texture tab** button on the right side of the **Texture** slot. In the **Textures** window, click on the **New** button. Then replace the default **Clouds** texture with an **Image or Movie** texture.
3. Click on the **Open** button and load the `spacehull_displ.exr` texture.
4. Go back to the **Object modifiers** window and set the displacement's **Strength** value to `0.200`. In the **Texture Coordinates** slot, select **UV**.

This way, the displacement features get mixed with the hull bump panels of the shader, giving a nice result. The `spacehull_displ.exr` texture is a 32-bit float displacement map created and stored in the **Blender Internal** engine. I modeled the Planes and scaled Cubes a simple **greeble** panel, then I baked the displacement on a different and unwrapped Plane, as shown in the following screenshot:

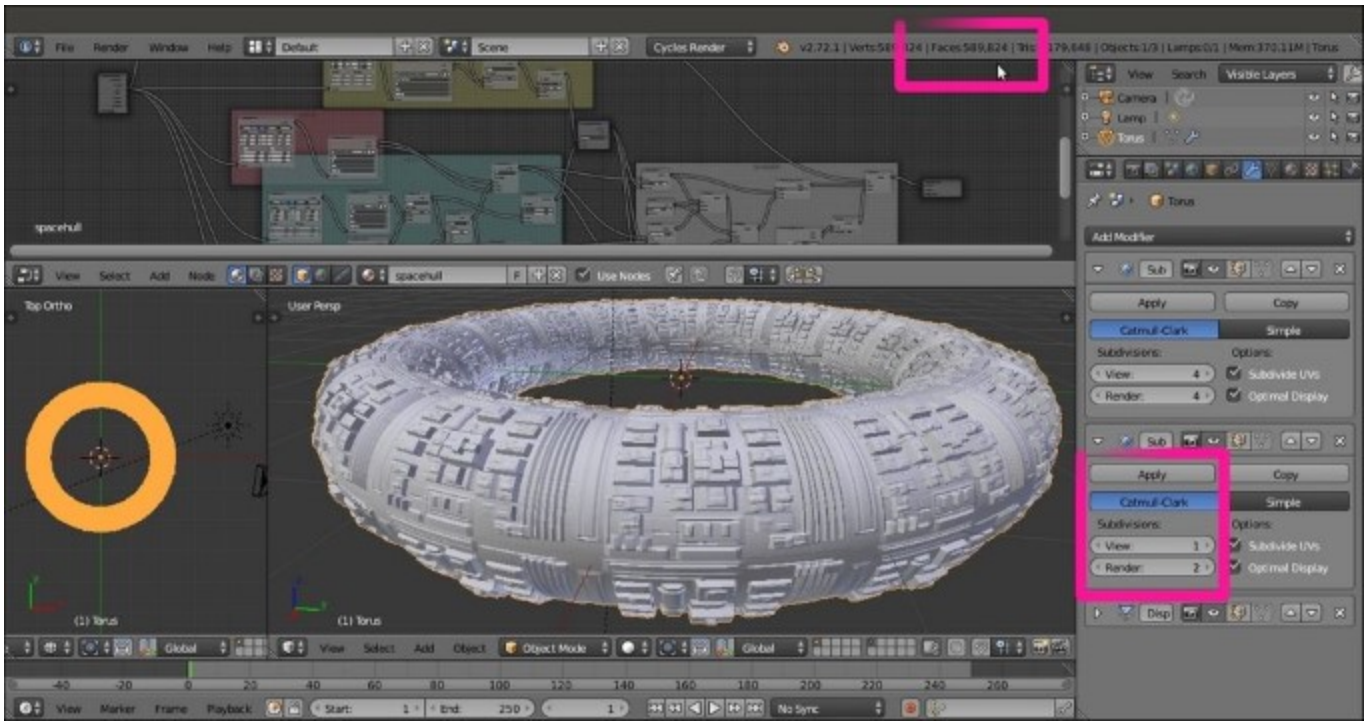


The greeble scene ready for the baking

Tip

If you want to take a look at the baking scene, open the `9931OS_06_greeble.blend` file.

Finally, we can try to set the first **Subdivision Surface** modifier level to `4` and lower the second **Subdivision Surface** modifier's level to `1`. Then, starting from the top one, apply all the modifiers. You will inevitably lose the details but will obtain a much lighter mesh—589,824 faces against the initial 2,359,296—and considering the fact that most of the details come from the texturing, the result looks pretty good (at least from a distance). It also looks good if the shading is set to **Flat** instead of **Smooth**.



The Torus spaceship with the applied modifiers

See also

- The displacement technique on the Blender Artists forum, at http://blenderartists.org/forum/showthread.php?273033-Sculpting-with-UVs-and-displacements.

Chapter 7. Subsurface Scattering in Cycles

In this chapter, we will cover the following recipes:

- Using the Subsurface Scattering shader node
- Simulating Subsurface Scattering in Cycles using the Translucent shader
- Simulating Subsurface Scattering in Cycles using the Vertex Color tool
- Simulating Subsurface Scattering in Cycles using the Ray Length output in the Light Path node
- Creating a fake Subsurface Scattering node group

Introduction

Subsurface Scattering is the effect of light not getting directly reflected by a surface but penetrating it and bouncing internally before getting absorbed or leaving the surface at a nearby point. In short, light is *scattered*.

The RGB channels of a surface color can have different scattering values, depending on the material; for example, for human skin the red component is more scattered (as a rough approximation, you could say that the values for the three channels are blue = 1, green = 2, and red = 4).

In Cycles, a true Subsurface Scattering node has been introduced in Blender 2.67. Since Version 2.72, it also works with the GPU (only in the Experimental feature set).

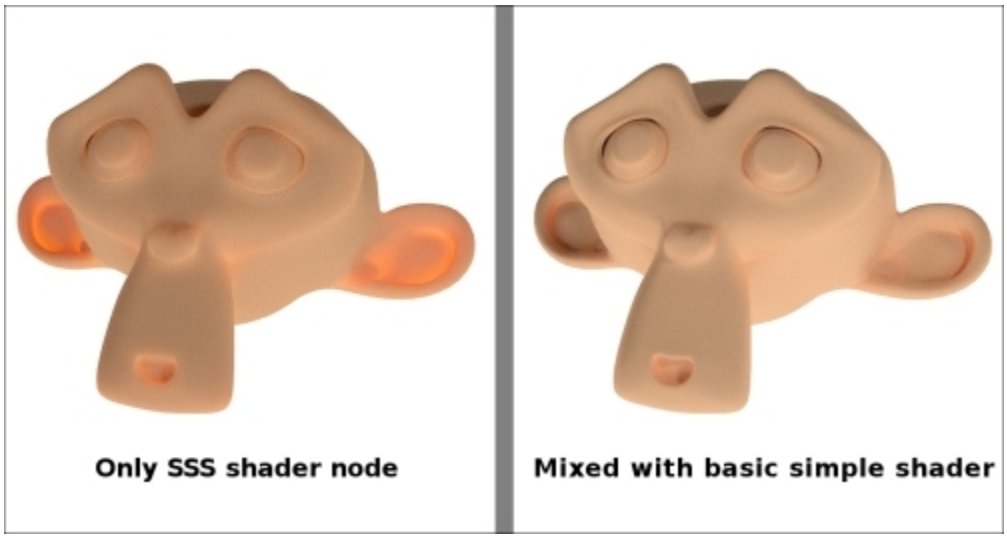
But sadly, it still has the common big Cycles problem—it takes a lot of samples to produce a noise-free rendering. In short, it's slow.

Besides the true node, there are other ways to simulate Subsurface Scattering in Cycles. All the recipes in this chapter faking the SSS effect use the **Translucent** shader node to achieve this effect, and shifting of colors is simulated by giving a main color to the translucent component. Keep in mind that even if the scattering effect in the true SSS node could be basically considered a sort of translucency effect, these tricks are not comparable to the real Subsurface Scattering effect. They are just ways to give the impression that light is being scattered through a material surface.

Also, depending on the recipe, you'll see that the effects of Subsurface Scattering can be quite different, and the more suitable method should be used according to the type of material you are going to create. The differences in these recipes are basically in the way translucency mixing is driven by different types of input.

Using the Subsurface Scattering shader node

Let's first see how the true **Subsurface Scattering** node works in Cycles, and an example is given in the following screenshot:



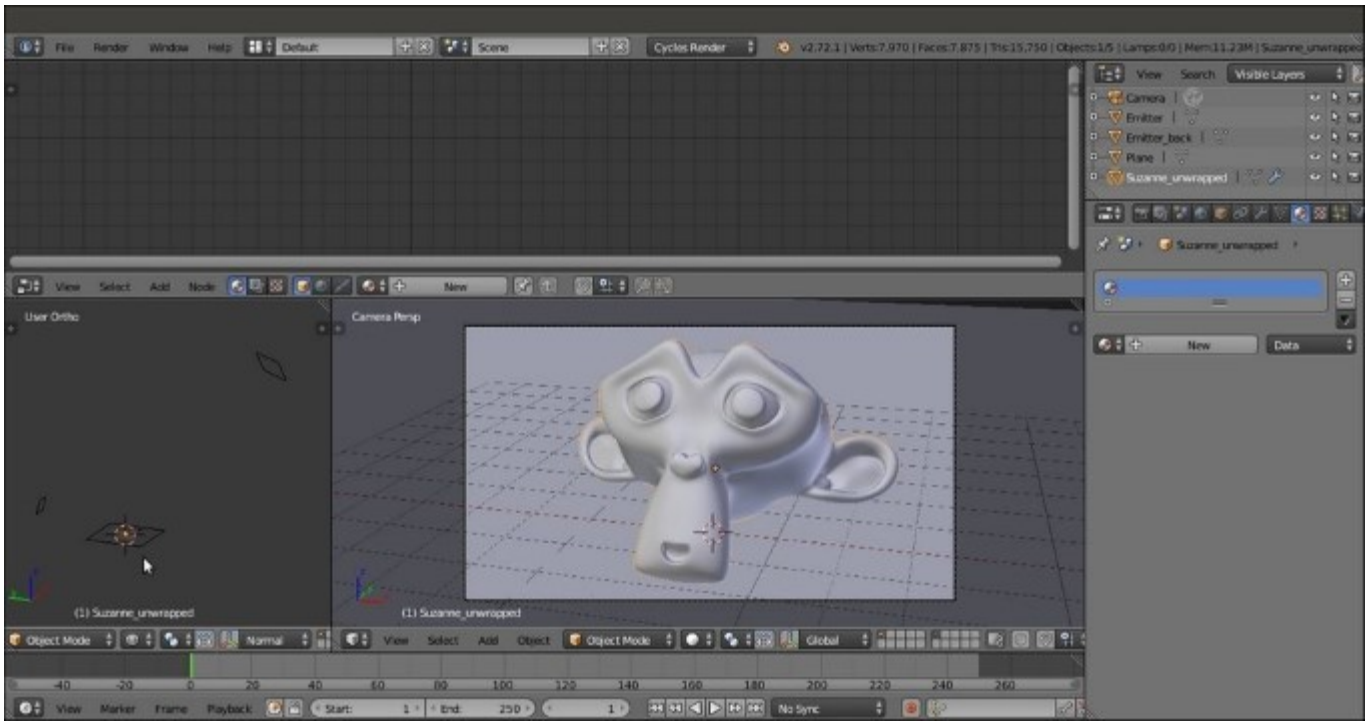
The Cycles SSS node

Getting ready

To see how the true **Subsurface Scattering** node works, let's first use it as the only component of the shader, and later mix it with a basic diffuse-glossy shader.

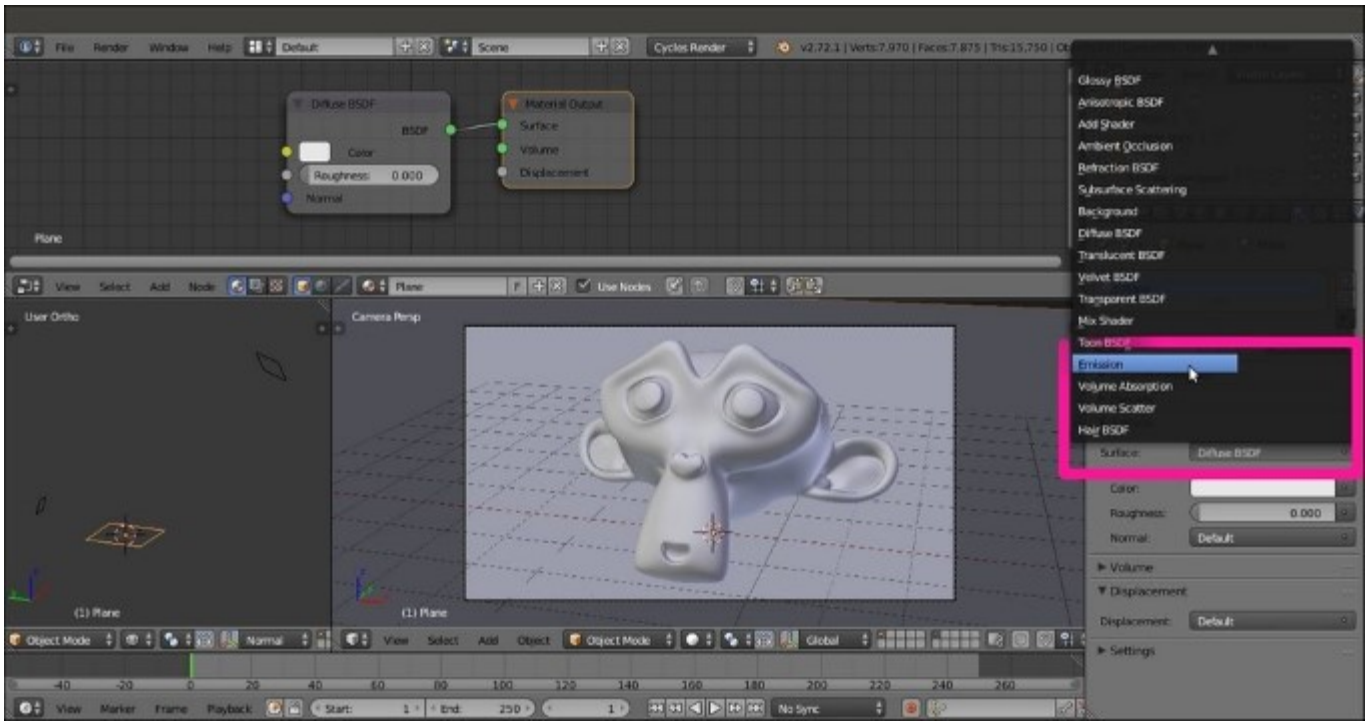
Let's start by setting the Plane under Suzanne as a light emitter to enhance the backlight effect of the SSS effect:

1. Start Blender and open the `9931OS_07_start.blend` file, where there is an unwrapped Suzanne mesh leaning on a Plane, with two mesh-light emitters and the Camera as shown in the following screenshot:



Screenshot of the provided 9931OS_07_start.blend file

2. Go to **Outliner** and select the **Plane** object. As you can see in the **Node Editor** window, it has an already set material called `Plane`.
3. Go to the **Material** window under the main **Properties** panel, and in the **Surface** subpanel, switch the **Diffuse BSDF** shader with an **Emission** shader as shown in the following screenshot:



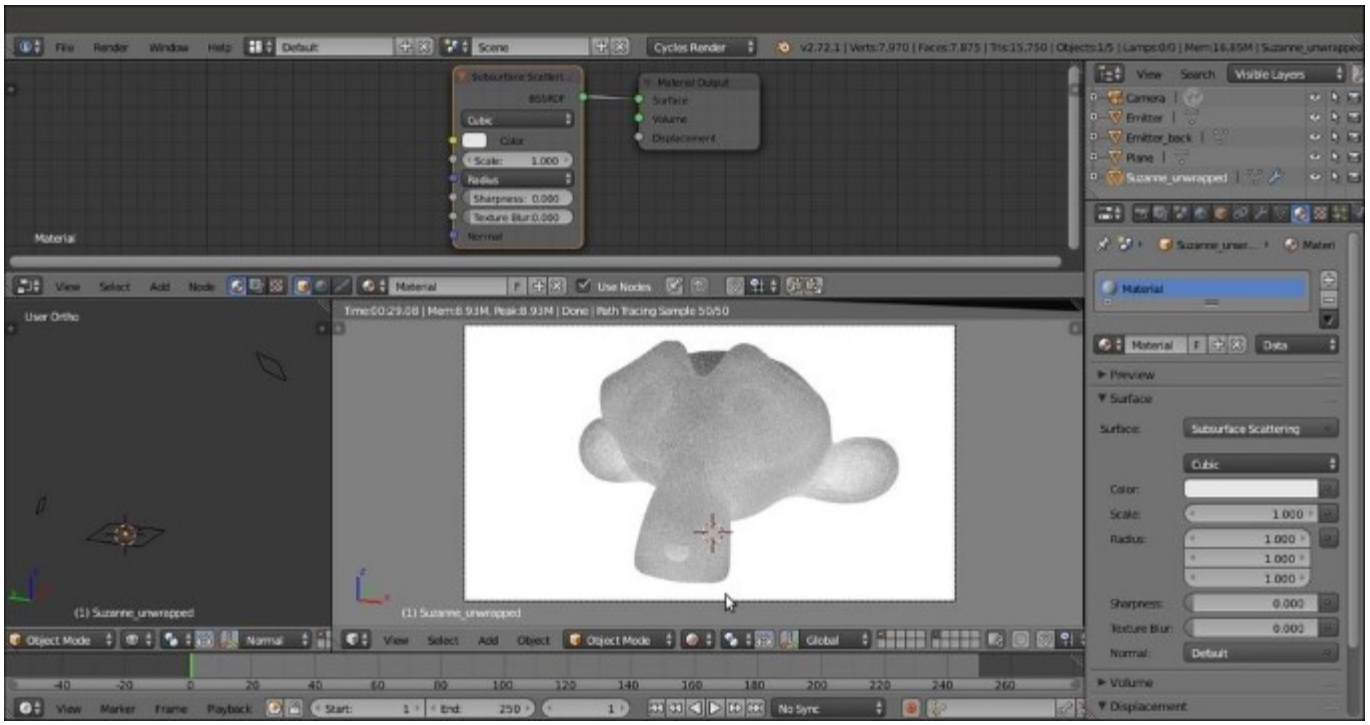
Switching the Diffuse BSDF shader with an Emission shader through the Material window

4. Set the **Strength** value to `5.000`.
5. With the mouse arrow in the viewport, press *Shift* + *Z* to go to the **Rendered** view.

How to do it...

Now let's begin creating the SSS material using the following steps:

1. Select **Suzanne** and click on the **New** button in the **Surface** subpanel under the **Material** window in the main **Properties** panel, or in the **Node Editor** window.
2. Using only the **Material** window, replace the **Diffuse BSDF** shader with a **Subsurface Scattering** node as shown in the following screenshot:

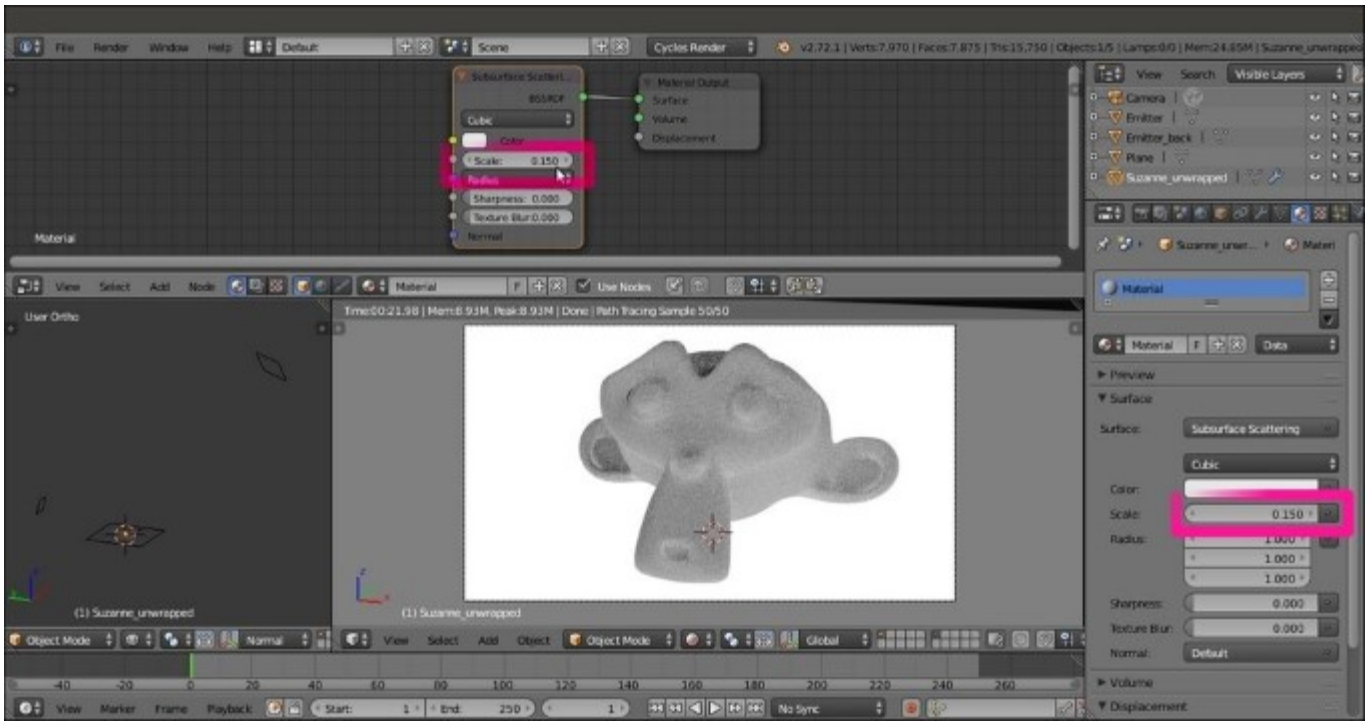


The Rendered preview of Suzanne with the SSS node as the material

As you can see, the scattering effect is clearly visible in the **Rendered** preview, but actually, it's so strong that all the facial features of poor Suzanne are confused and result in a jelly-like, muddish material.

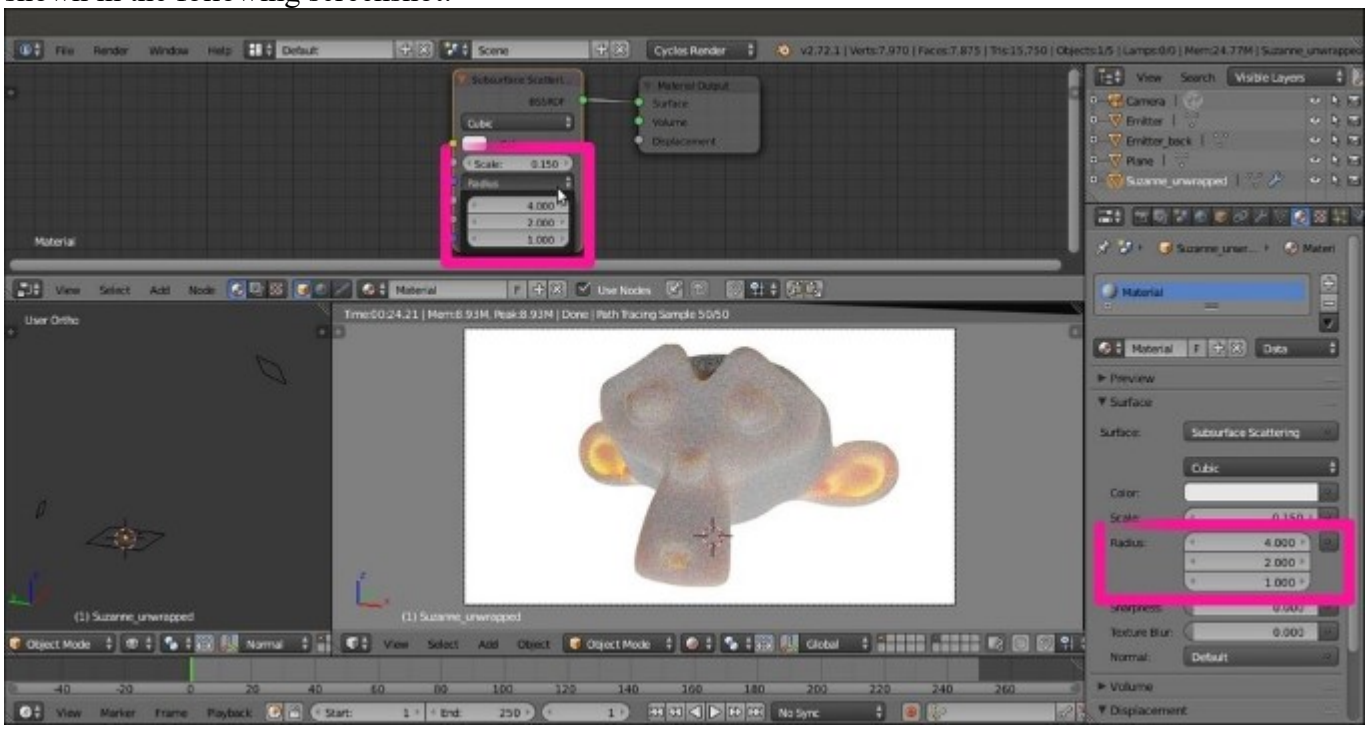
By default, the **Scale** value of the **Subsurface Scattering** node is set to `1.000`, evidently a bit too high for an object that is supposed to be 2 meters tall (remember that by default, one Blender unit is supposed to be equal to 1 real world meter).

3. Gradually lower the **Scale** parameter, either in **Node Editor** or in the **Material** window, to select a value in the range of `0.100` to `0.200`. In my case, I arrived at `0.150`. Now some of Suzanne's facial features are clearly discernible, as shown in the following screenshot:



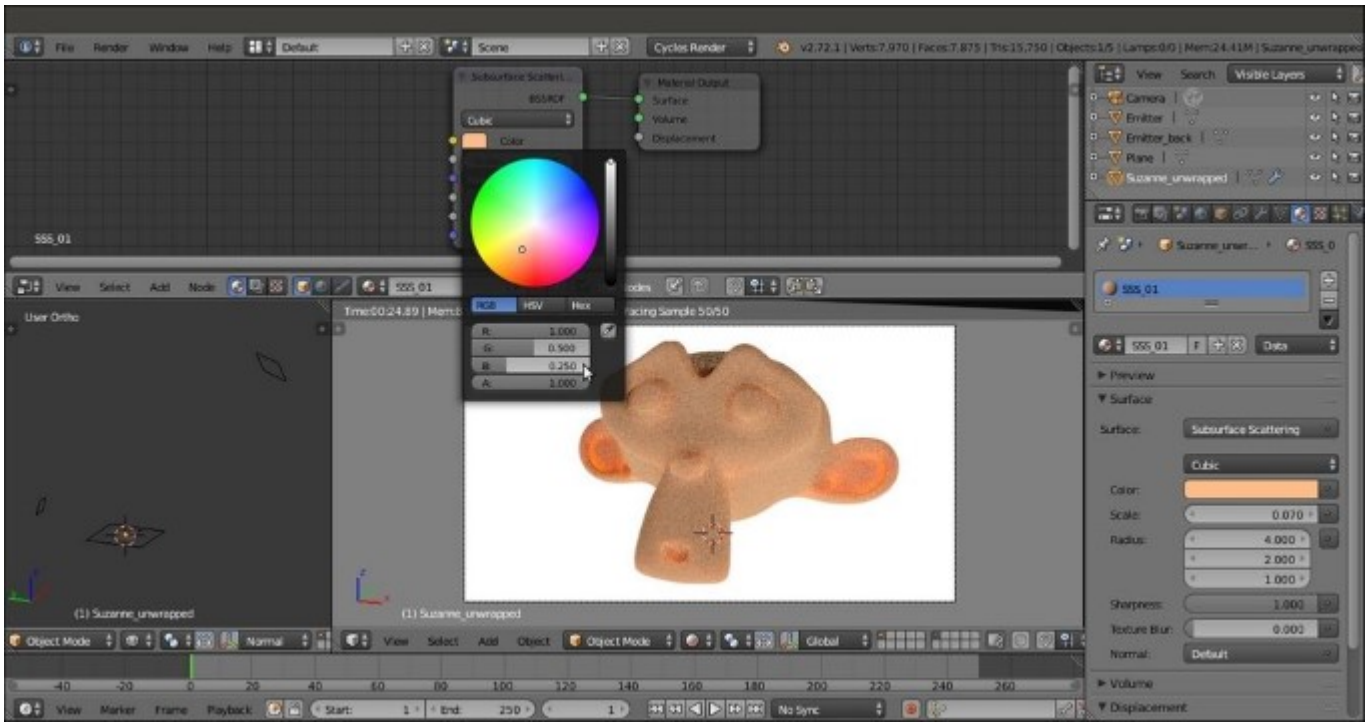
Modifying the SSS node's Scale value

4. Click on the **Radius** button on the node interface in the **Node Editor** window (or directly in the **Material** window), and change the values of **R** to `4.000`, **G** to `2.000`, and **B** to `1.000` as shown in the following screenshot:



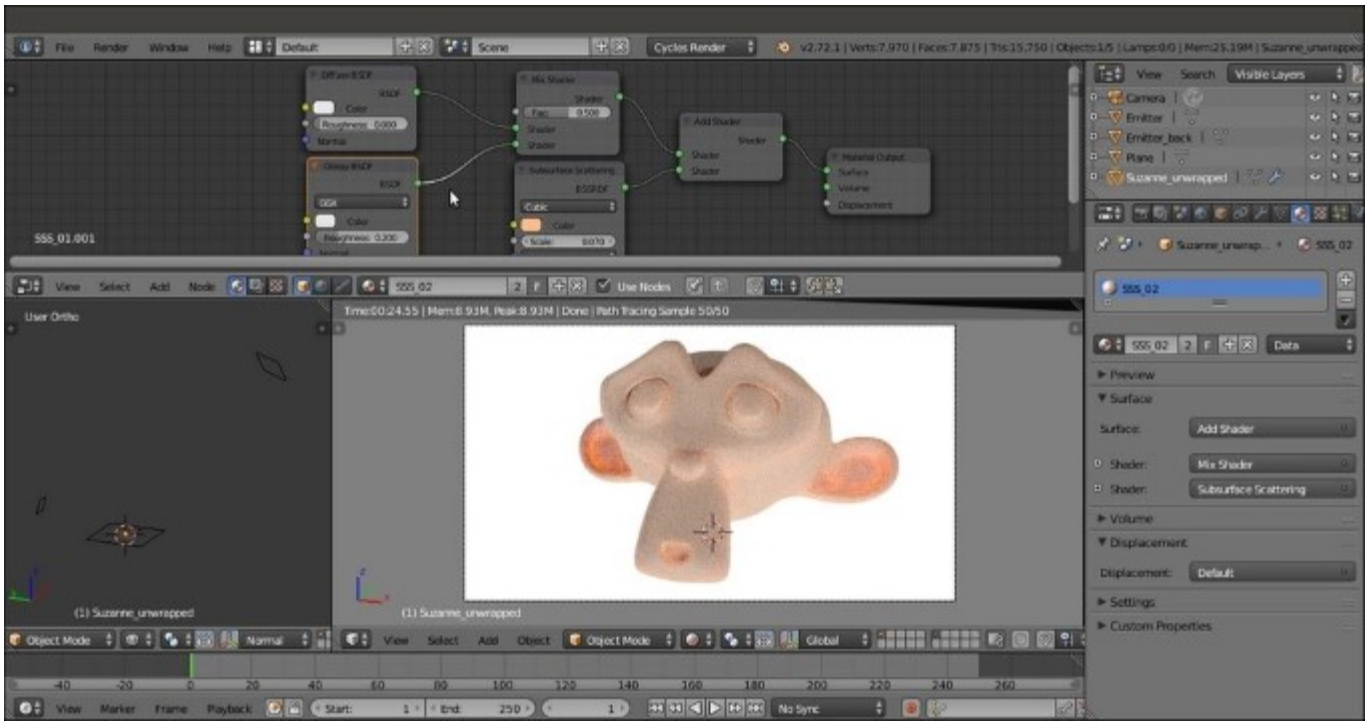
Modifying the SSS node's Radius values

5. Lower the **Scale** value to `0.070`; set the **Sharpness** value to `1.000`; and click on the **Color** box to set values of **R** to `1.000`, **G** to `0.500`, and **B** to `0.250`.
6. Rename the material `SSS_01` and save the file as `SSS_material`, as shown in the following screenshot:



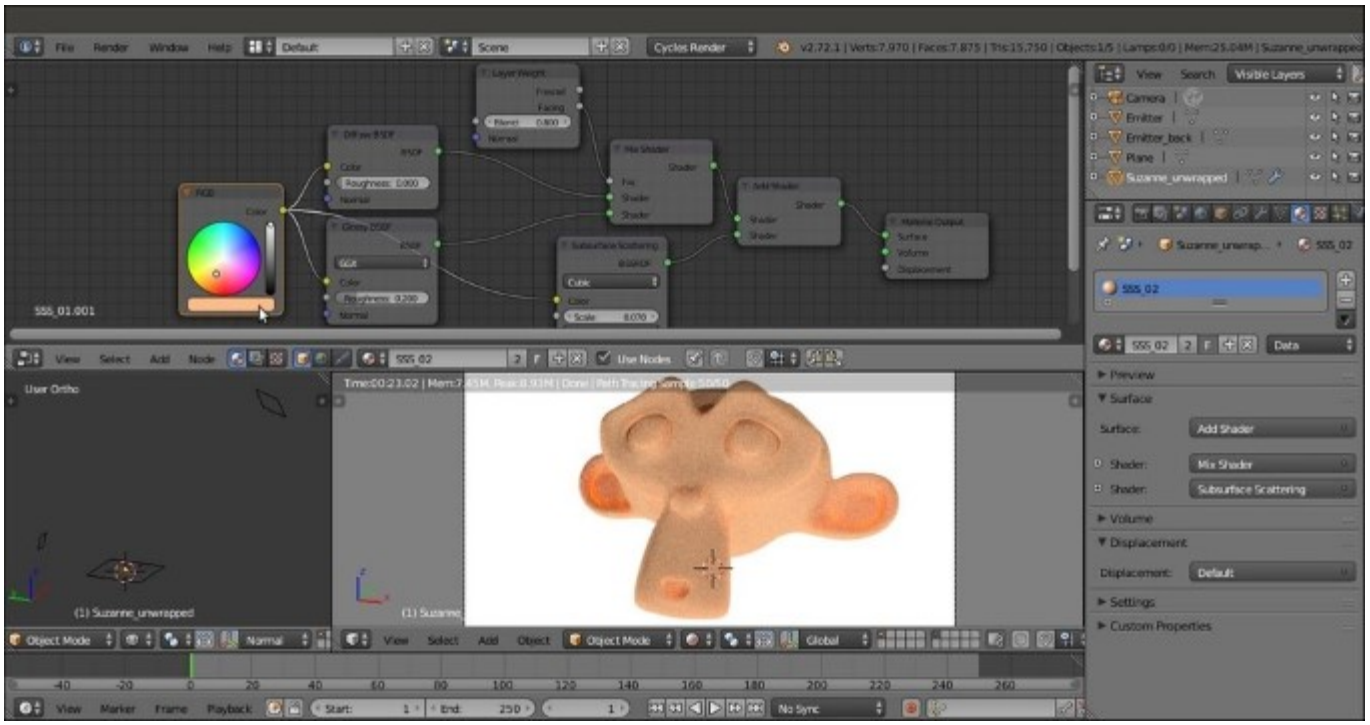
Setting a flesh color for the Suzanne SSS

7. Now click on the **F** icon to the right side of the **Material datablock** name to enable the *fake user*. Then click on the number **2** icon and rename the new material `SSS_02`. Enable the *fake user* for this material as well.
8. Add an **Add Shader** node (press Shift + A and navigate to **Shader** | **Add Shader**) and paste it between the **SSS** node and the **Material Output** node.
9. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), and connect it to the first **Shader** input socket of the **Add Shader** node so that the previous connection coming from the **SSS** node automatically switch to the second **Shader** input socket.
10. Add a **Diffuse BSDF** node and a **Glossy BSDF** shader node (press *Shift* + *A* and navigate to **Shader** | **…**), and connect them to the first and to the second **Shader** input sockets of the **Mix Shader** node respectively, as shown in the following screenshot:



Adding the SSS node to a Diffuse-Glossy shader

11. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader** node. Set the **Blend** value to `0.800`.
12. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and connect its output to the **Color** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **SSS** nodes. Set the **RGB** node's **Color** values for **R** to `1.000`, **G** to `0.500`, and **B** to `0.250` as shown in the following screenshot:



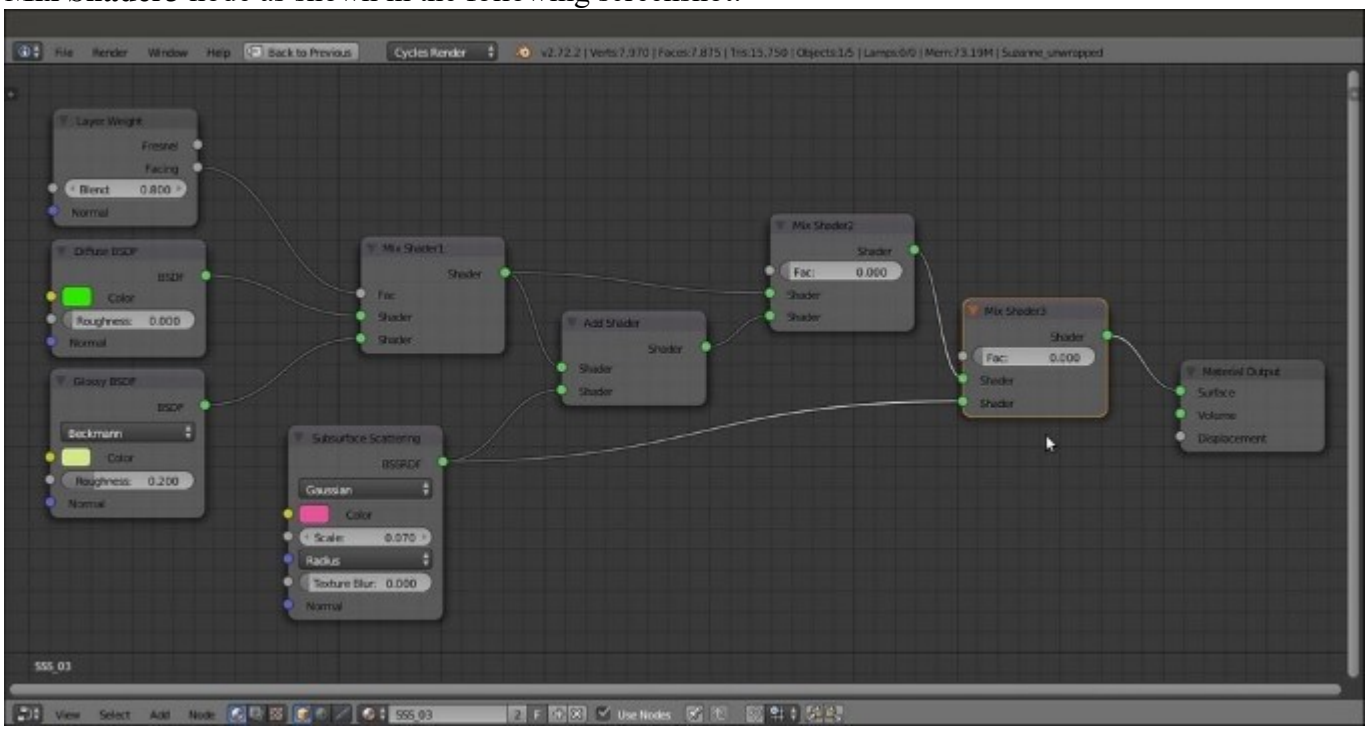
Setting the same flesh color for all the shader nodes

13. Save the file.

Let's try now a slightly different setting, with two sliders for the mixture of **SSS** and basic shaders. We will also give distinct colors to the **Diffuse BSDF**, **Glossy BSDF**, and **SSS** components of the shader to highlight their distribution on the mesh.

1. First, select the **Plane** object, and in the **Material** window, switch the **Emission** shader with a **Diffuse BSDF** shader node.
2. Reselect **Suzanne** and click on the number **2** button close to the **Material datablock** name. Rename the new material `SSS_03`. Then enable the *fake user* for this material as well.
3. Delete the **RGB** node. Then set the **Diffuse BSDF** shader node's **Color** values for **R** to `0.031`, **G** to `0.800`, and **B** to `0.000` (bright green); and the **Glossy BSDF** node's **Color** values for **R** to `0.646`, **G** to `0.800`, and **B** to `0.267` (yellow). Set the **Glossy BSDF** node's **Roughness** value to `0.200` and **Distribution** to **Beckmann**. Set the **Subsurface Scattering** shader node's **Color** values for **R** to `0.800`, **G** to `0.086`, and **B** to `0.317` (a vivid pink). Change **Falloff** from **Cubic** to **Gaussian**.
4. Label the **Mix Shader** node as `Mix Shader1`, press *Shift* + *D* to duplicate it, and label the duplicate as `Mix Shader2`. Paste it between the **Add Shader** and **Material Output** nodes.
5. Connect the **Mix Shader1** node's output to the first **Shader** input socket of the **Mix Shader2** node so that the connection from the **Add Shader** automatically switches to the second **Shader** input socket.
6. Press *Shift* + *D* to duplicate the **Mix Shader2** node, label the duplicated node as `Mix Shader3`, and paste it between the **Mix Shader2** and the **Material Output** nodes.

7. Connect the output of the **Subsurface Scattering** node to the second **Shader** input socket of the **Mix Shader3** node as shown in the following screenshot:



Adding one more Mix Shader node to further tweak the SSS amount

8. Save the file as `SSS_material_02.blend`.

How it works...

The scattering amount for the three **RGB** color channels is set in the **Radius** item on the node interface, while **Scale** is to set the dimensions the object would have in the real world. Starting with a default value of `1.000`, the **Scale** value must usually be proportionally inverse lowered. The bigger the object desired in the real world, the lower the **Scale** value in the node. Otherwise, the scattering effect may become too strong.

The best way to mix the **Subsurface Scattering** node with the rest of any shader is by using the **Add Shader** node. However, with this node, it's not possible to establish the amount of influence of the **SSS** on the shader, so a trick must be performed. The Diffuse-Glossy component of the shader is again mixed with the output of the **Add Shader** node, through a **Mix Shader** node.

In the previously explained `SSS_03` material, there are two **Mix Shader** nodes that can be used to tweak the influence of the effect. By raising their **Fac** values, it's also possible to switch from total absence to full scattering effect, as shown in the following compilation of screenshots:

Different effects of different Fac values of the last Mix Shader node

See also

- Refer to http://en.wikipedia.org/wiki/Subsurface_scattering

Simulating Subsurface Scattering in Cycles using the Translucent shader

In this recipe, we will create a fake Subsurface Scattering material using the Translucent BSDF shader node as shown in the following screenshot:

The rendered result of the fake SSS of this recipe

As someone suggested, this material could actually be quite good to make candles.

Getting ready

Start Blender and open the `9931OS_07_start.blend` file:

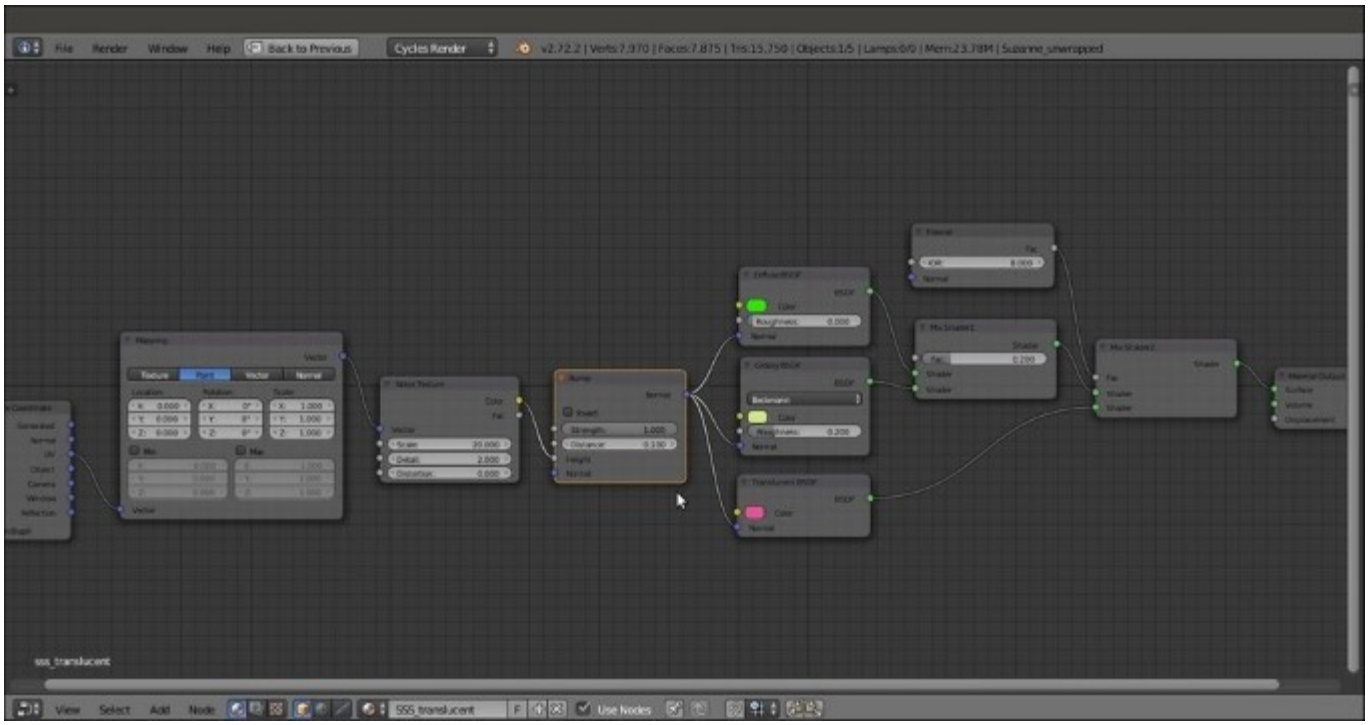
1. Go to the **Render** window, and in the **Sampling** subpanel, click on the **Method to sample lights and materials** button to switch from **Path Tracing** to **Branched Path Tracing**. Enable the **Square Samples** item, and under **AA Samples**, set the **Render** value to `8`. Finally, click on the **Pattern** button to select the **Correlated Multi-Jitter** item.
2. Save the file as `9931OS_SSS_translucent.blend`.

How to do it...

Let's go ahead and create the material using the following steps:

1. Select the **Suzanne** object and click on the **New** button in the **Node Editor** window toolbar, or in the **Material** window to the right. Rename the material `SSS_translucent`.

2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** shader again, and in the second slot, select a **Glossy BSDF** shader node.
3. Set the **Diffuse BSDF** shader node's **Color** values for **R** to `0.031`, **G** to `0.800`, and **B** to `0.000`. Set the **Glossy BSDF** node's **Color** values for **R** to `0.646`, **G** to `0.800`, and **B** to `0.267`. Set the **Glossy BSDF** node's **Roughness** value to `0.200` and **Distribution** to **Beckmann**.
4. Select the **Mix Shader** node and go to the **Properties** side-panel of the **Node Editor** window (if not present, move the mouse to the **Node Editor** window and press the *N* key to make it appear). In the **Label** slot inside the **Node** subpanel, label the **Mix Shader** node as `Mix Shader1`. Then set its **Fac** value to `0.200`.
5. Add a new **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader2`, and paste it between the **Mix Shader1** node and the **Material Output** node.
6. Add a **Translucent BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Translucent BSDF**) and connect it to the second **Shader** input socket of the **Mix Shader2** node. Set the **Color** values of **R** to `0.800`, **G** to `0.086`, and **B** to `0.317`.
7. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**).
8. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of this node to the **Vector** input socket of the **Noise Texture** node. Set the **Noise Texture** node's **Scale** value to `20.000`.
9. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **Color** output of the **Noise Texture** node to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF** nodes. Leave the **Bump** strength at `1.000`.
10. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader2** node. Set the **IOR** value to `8.000` as shown in the following screenshot:



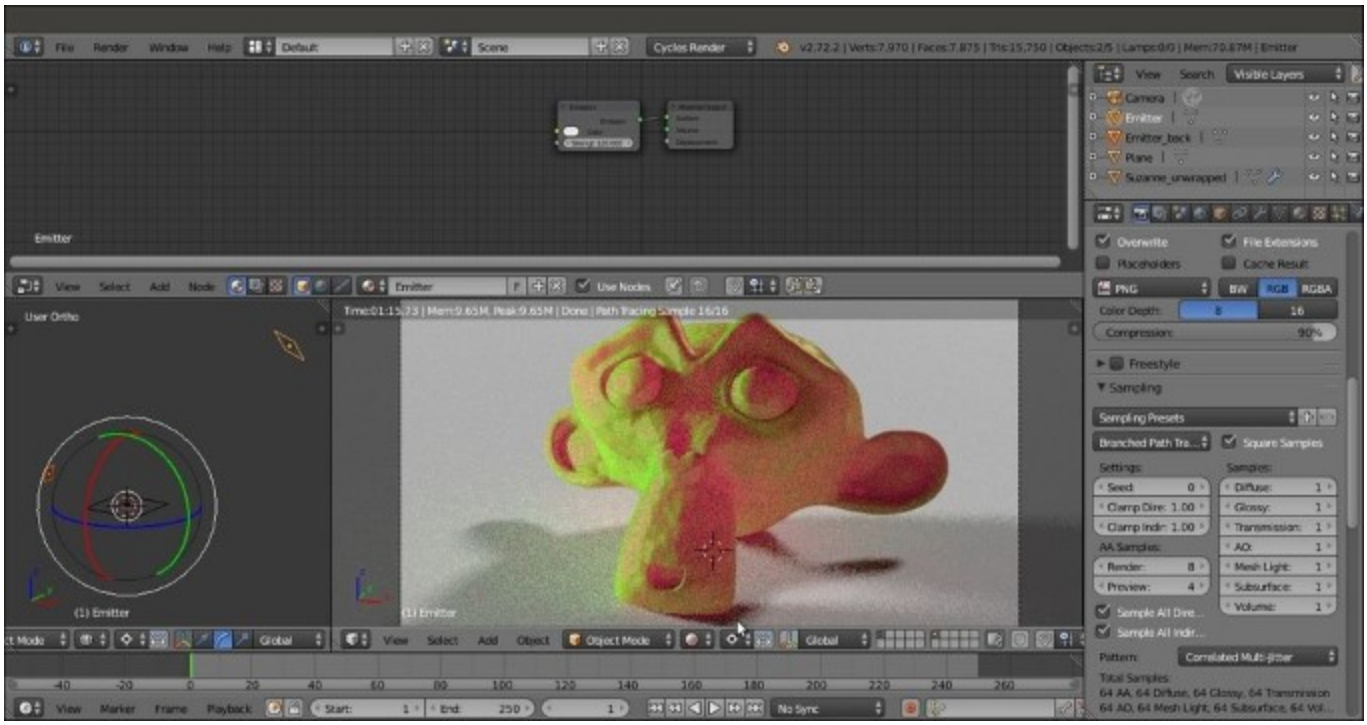
The overall view of the material network

11. Save the file.

How it works...

This is probably the simpler form of the fake Subsurface Scattering effect you can get in Cycles. It is obtained by simply blending a translucent effect with a basic Diffuse-Glossy shader. By varying the amount of the **IOR** value in the **Fresnel** node (set quite high as a starting point), it is possible to establish the amount of translucency on the mesh. We also added a **Noise Texture** bump effect to the material, just to make it appear more jelly-like.

Note that we gave almost complementary colors to the **Diffuse BSDF** and **Translucent BSDF** shaders to show the effect more clearly, but colors similar to each other can work better. Also note that the translucent effect actually follows the direction of the lighting. Try to rotate the **Emitter** and the **Emitter_back** planes around the Suzanne mesh to verify this in real time, through the **Rendered** view as shown in the following screenshot:



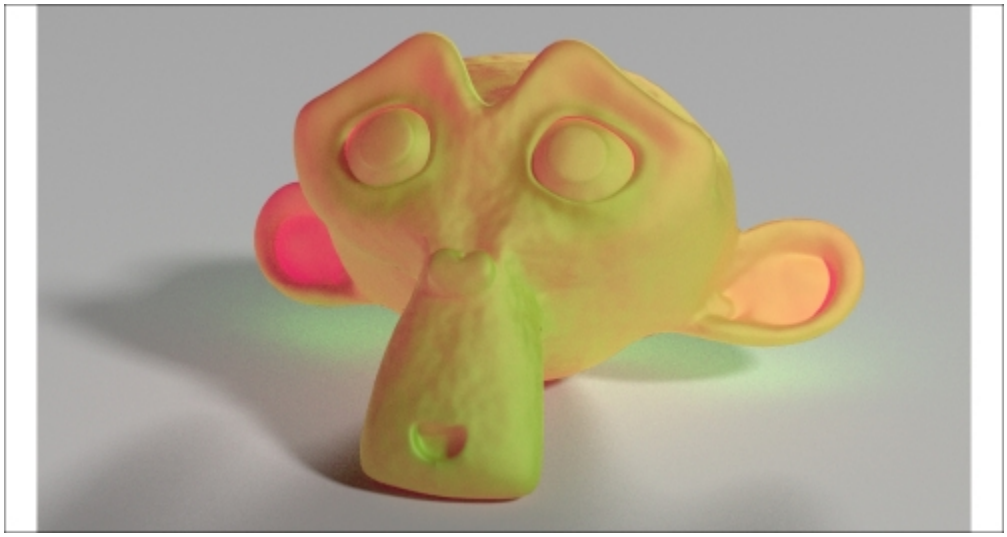
A preview of the fake SSS material lit from a different angle

Note

Note that for the three components of the shader (**Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF**) we used (and will also use for the following recipes) the same colors of the `SSS_03` material. This was done to make an easier comparison between the effects obtained in the recipes.

Simulating Subsurface Scattering in Cycles using the Vertex Color tool

In this recipe, we will create a fake Subsurface Scattering material as shown in the following screenshot, using the Vertex Color tool:

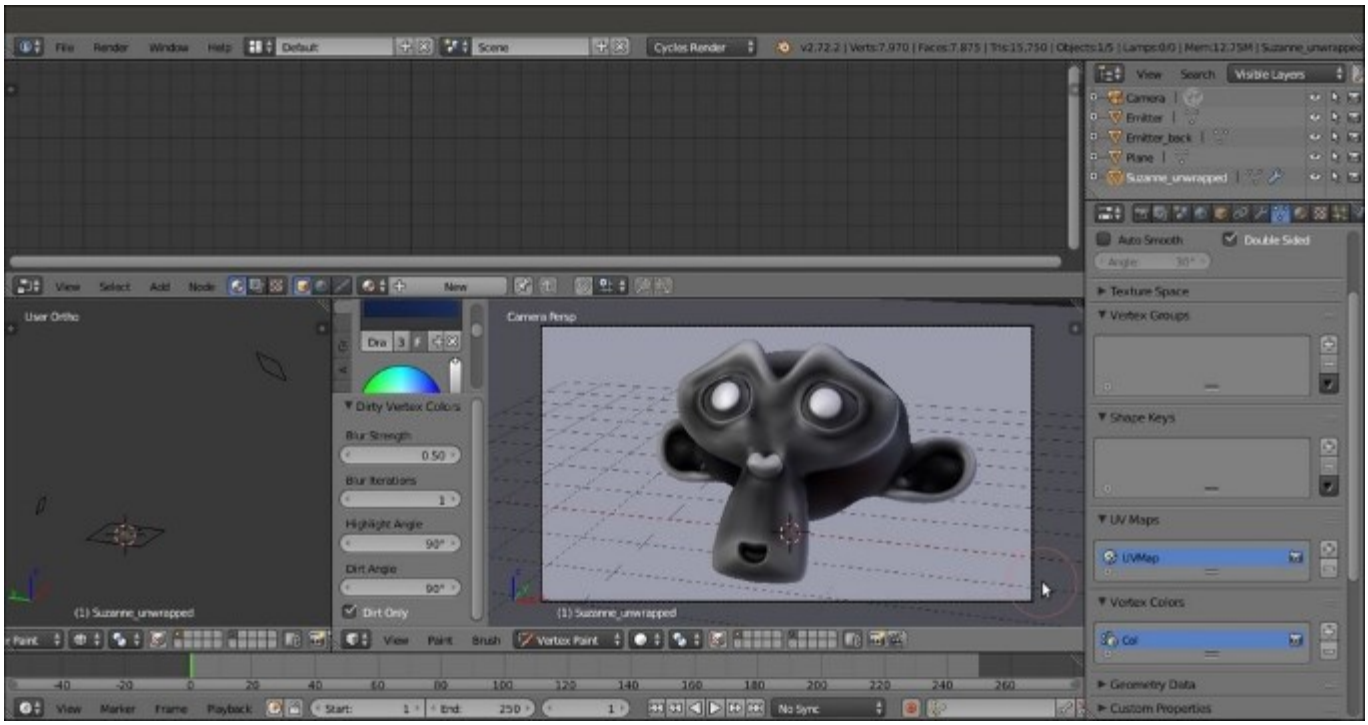


The Rendered result of the vertex color fake SSS material of this recipe

Getting ready

Start Blender and open the `9931OS_07_start.blend` file.

1. Go to the **Render** window, and in the **Sampling** subpanel, click on the **Method to sample lights and materials** button to switch from **Path Tracing** to **Branched Path Tracing**. Enable the **Square Samples** item, and under **AA Samples**, set the **Render** value to `8`. Finally, click on the **Pattern** button to select the **Correlated Multi-Jitter** item.
2. Select the **Suzanne** mesh, click on the **Mode** button in the **Camera** view toolbar, and choose **Vertex Paint** (or just press the *V* key). Now Suzanne goes into **Vertex Paint** mode.
3. Click on the **Paint** item to the left of the **Mode** button and select **Dirty Vertex Colors**. Then press *T*, and in the last operation subpanel (**Dirty Vertex Color**) at the bottom of the **Tool Shelf** panel, set **Blur Strength** to `0.50`, **Highlight Angle** to `90°`, and **Dirt Angle** to `90°`. Enable the **Dirt Only** item as shown in the following screenshot:



A screenshot of Suzanne in Vertex Paint mode and the Dirty Vertex Color values at the bottom of the Tool Shelf

The Suzanne mesh inside the `9931OS_07_start.blend` file already had a Vertex Color layer named `Col`. With the previous procedure, we overwrote it.

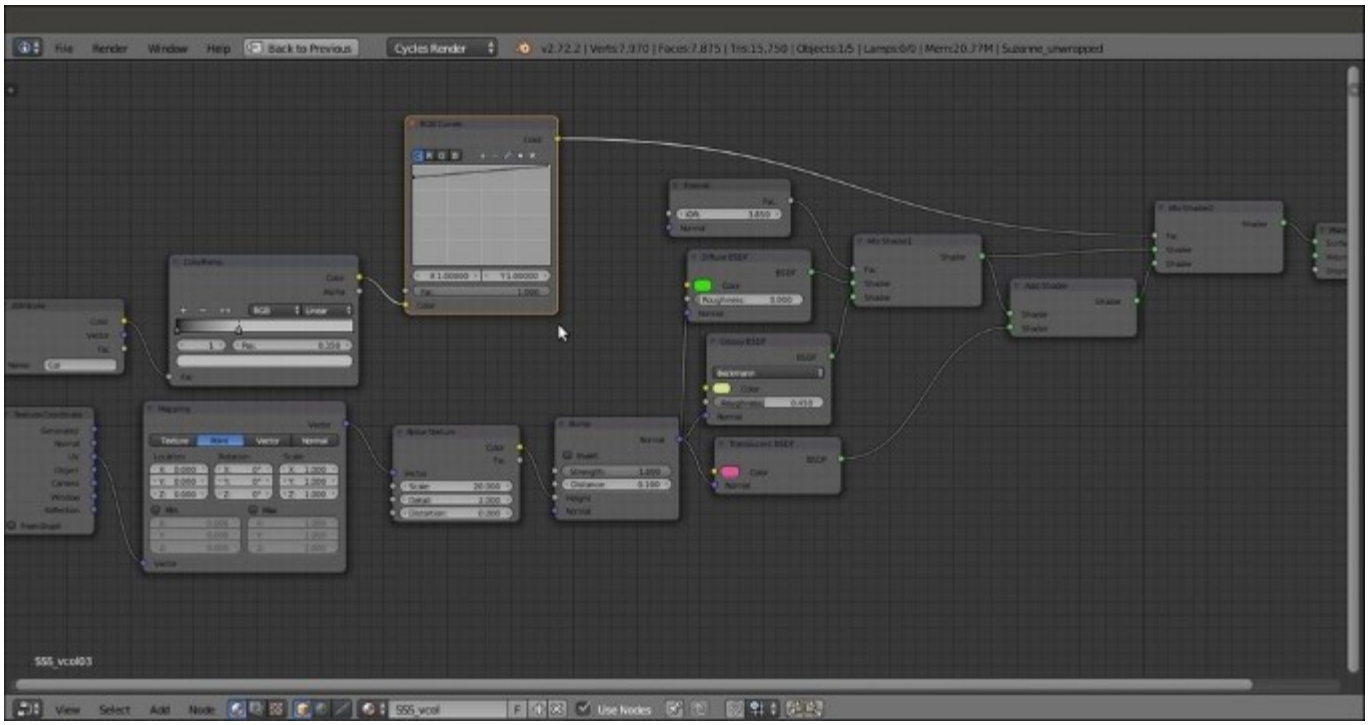
4. Go to the **Object data** window under the main **Properties** panel to see it in the **Vertex Colors** subpanel. Then go back to **Object Mode** and press *T* to get rid of the **Tool Shelf** panel.
5. Save the file as `9931OS_07_SSS_vcol.blend`.

How to do it...

After the vertex color preparation, let's go for the material itself by following these steps:

1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window under the main **Properties** panel. Rename the material `SSS_vcol`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with an **Add Shader** node. In the first **Shader** slot, select a **Mix Shader** node. In the second **Shader** slot, select a **Translucent BSDF** shader node. In the **Properties** side panel to the right of the **Node Editor** window, label the **Mix Shader** node as `Mix Shader1`.
3. Go to the **Mix Shader1** node. In the first **Shader** slot, select a **Diffuse BSDF** shader node. In the second **Shader** slot, select a **Glossy BSDF** shader node. Set the **Glossy BSDF** node's **Roughness** value to `0.450` and **Distribution** to **Beckmann**.
4. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**), connect it to the **Fac** input socket of the **Mix Shader1** node, and set the **IOR** value to `3.850`.

5. Set the **Diffuse BSDF** node's **Color** values for **R** to `0.031`, **G** to `0.800`, and **B** to `0.000` (the same bright green as in the *Simulating Subsurface Scattering in Cycles using the Translucent shader* recipe); and the **Translucent BSDF** node's **Color** values for **R** to `0.800`, **G** to `0.086`, and **B** to `0.317` (the same pink as in the *Using the Subsurface Scattering shader node* recipe). Set the **Glossy BSDF** node's **Color** values for **R** to `0.646`, **G** to `0.800`, and **B** to `0.267`, again it's the same yellowish color as in the *Using the Subsurface Scattering shader node* recipe).
6. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**).
7. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of this node to the **Vector** input socket of the **Noise Texture** node. Set the **Noise Texture** node's **Scale** value to `20.000`.
8. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **Color** output of the **Noise Texture** node to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF** nodes.
9. Add a new **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader2`, and paste it between the **Add Shader** and **Material Output** nodes. Then move the connection from the **Add Shader** node to the second **Shader** input socket, and connect the output of the **Mix Shader1** node to the first **Shader** input socket of the **Mix Shader2** node.
10. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). In the **Name** slot of the **Attribute** node, write the vertex color layer name, that is, `Col`. Then connect the **Color** output of **Attribute** node to the **Fac** input socket of the **ColorRamp** node. In the **ColorRamp** node, move the white color stop to `0.350` position.
11. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**) and connect the **Color** output of the **ColorRamp** node to the **Color** input socket of this node. Then connect its **Color** output to the **Fac** input socket of the **Mix Shader2** node.
12. Inside the **RGB Curves** node's interface window, move the first curve control point coordinate values for **X** to `0.00000` and **Y** to `0.88125`, and the second point coordinate values for **X** to `1.00000` and **Y** to `1.00000`.
13. Save the file. The overall network will be as shown in the following screenshot:



The overall network; note the Vertex Color output intensified by ColorRamp and RGB Curves nodes

How it works...

Compared to the former recipe, in this case, we used information about the Vertex Color, enhanced by the **ColorRamp** node, to drive the mixing of the translucency with the other components of the shader. It's clear that the final result is largely due to vertex painting. We obtained this result quickly through the Dirty Vertex Color tool, but that could also be painted by hands (imagine you're painting a mask for a skull under the face skin).

Simulating Subsurface Scattering in Cycles using the Ray Length output in the Light Path node

In this recipe, we will create a fake Subsurface Scattering material using the **Ray Length** output of the **Light Path** node.

The Rendered result of the fake SSS material of this recipe

Getting ready

Start Blender and open the `9931OS_07_start.blend` file.

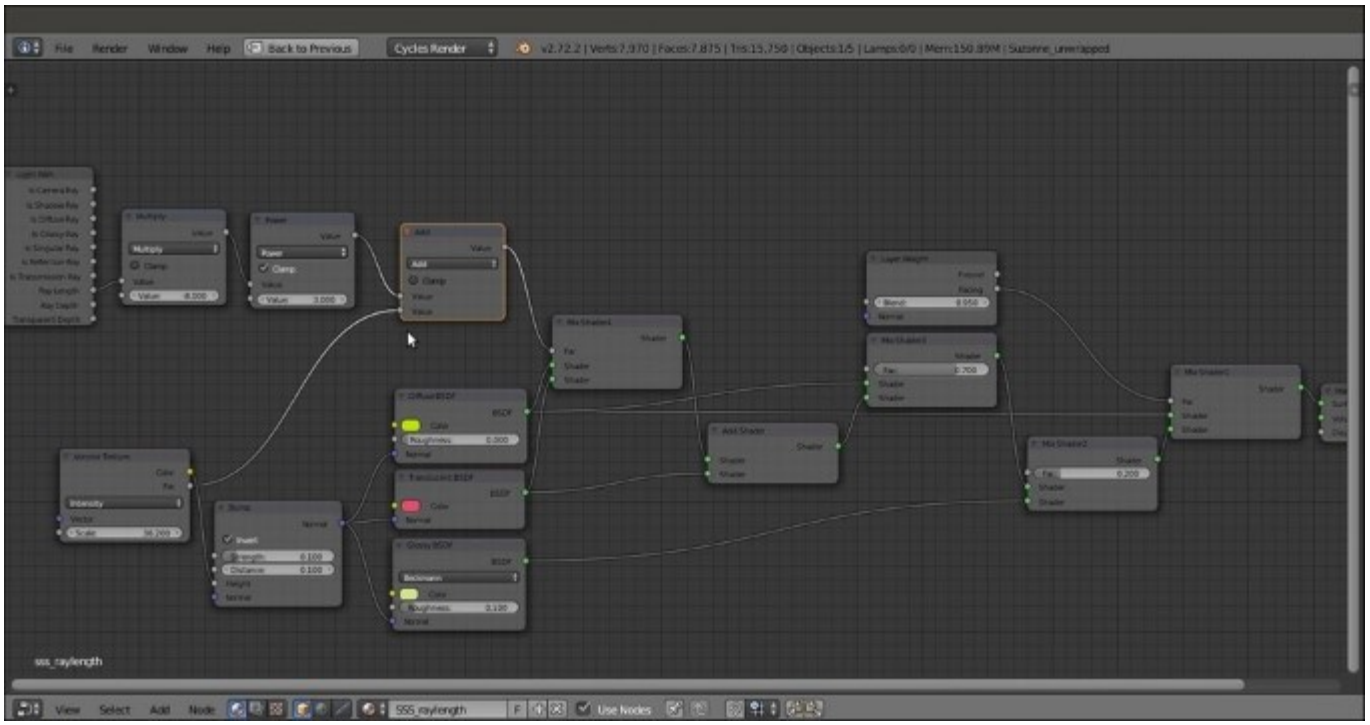
1. Go to the **Render** window, and in the **Sampling** subpanel, click on the **Method to sample lights and materials** button to switch from **Path Tracing** to **Branched Path Tracing**. Enable the **Square Samples** item, and under **AA Samples**, set the **Render** value to `8`. Finally, click on the **Pattern** button to select the **Correlated Multi-Jitter** item.
2. Save the file as `9931OS_07_SSS_raylength.blend`.

How to do it...

Let's create the material using the following steps:

1. Select the **Suzanne** object. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window to the right of the screen. Rename the material `SSS_raylength`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. Label it as `Mix Shader1`. In its first **Shader** slot, select a **Diffuse BSDF** shader. In its second **Shader** slot, select a new **Mix Shader** node. Label this node as `Mix Shader2`.

3. Go to the **Mix Shader2** node. In its first **Shader** slot, select a new **Mix Shader** node and label it as `Mix Shader3`. In the second **Shader** slot, select a **Glossy BSDF** node.
4. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader1** node. Set the **Blend** value to `0.950`.
5. Set the **Fac** value of the **Mix Shader2** node to `0.200` and the **Fac** value of the **Mix Shader3** node to `0.700`. Set the **Glossy BSDF** node's **Roughness** to `0.100`.
6. Connect the **Diffuse BSDF** node output to the first **Shader** input socket of the **Mix Shader3** node. Add an **Add Shader** node (press *Shift* + *A* and navigate to **Shader** | **Add Shader**) and connect it to the second **Shader** input socket of the **Mix Shader3** node.
7. Go to the **Add Shader** node, and in the first **Shader** slot, select the last **Mix Shader** node. Label it as `Mix Shader4`. In the second **Shader** slot, select a **Translucent BSDF** shader node.
8. Connect the output of the **Diffuse BSDF** node to the first **Shader** input socket of the **Mix Shader4** node, and the output of the **Translucent BSDF** node to the second **Shader** input socket.
9. Set the **Diffuse BSDF** shader node's **Color** values for **R** to `0.031`, **G** to `0.800`, and **B** to `0.000`; the **Glossy BSDF** shader node's **Color** values for **R** to `0.646`, **G** to `0.800`, and **B** to `0.267`; and the **Translucent BSDF** shader node's **Color** values for **R** to `0.800`, **G** to `0.086`, and **B** to `0.317`.
10. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Color** output of the **Voronoi Texture** node to the **Height** input socket of the **Bump** node and the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF** nodes. Set the **Voronoi Texture** node's **Scale** value to `32.600` and the **Bump** node's **Strength** value to `0.100`. Then enable the **Invert** item in the **Bump** node.
11. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**) and a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set the **Math** node's **Operation** to **Multiply**. Connect the **Ray Length** output of the **Light Path** node to the first **Value** input socket of the **Math** node. Then set the second **Value** input socket to `-8.000`.
12. Press *Shift* + *D* to duplicate the **Math** node. Set **Operation** to **Power**. Connect the output of the **Multiply** math node to the first **Value** input socket of this node. Set the second **Value** input socket to `3.000`. Enable the **Clamp** item.
13. Press *Shift* + *D* to duplicate the **Power-Math** node, set **Operation** to **Add**, and connect the output of the **Power** node to its first **Value** input socket. Connect its output to the **Fac** input socket of the **Mix Shader4** node.
14. Connect the **Fac** output of the **Voronoi Texture** node to the second **Value** input socket of the **Add-Math** node.



The completed network of the material

15. Save the file.

How it works...

In this recipe, we used the **Ray Length** output of the **Light Path** node to drive the amount of translucency on the mesh. **Ray Length** does exactly what its name suggests. It returns the length of a light ray passing through an object. So basically, it is possible for Cycles to know the thickness of a mesh. On the thicker parts, the translucency will show less or even for nothing, whereas it will be more visible on the thinner parts of the mesh.

Note

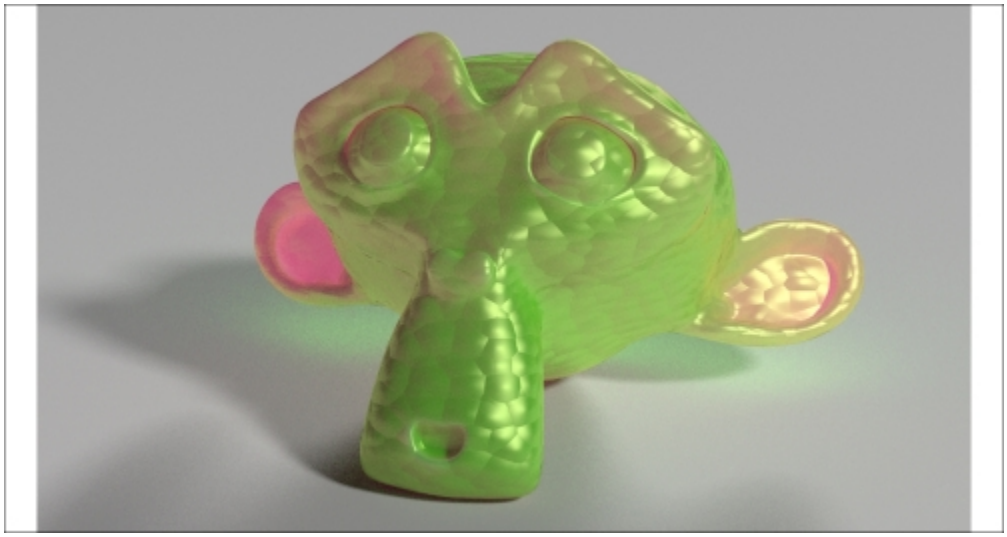
Note that in the shader network, the **Ray Length** output was intensified by a set of **Math** nodes and added to the **Voronoi Texture** node's output. Then it was connected to the factor input of the **Mix Shader** node to drive the blending of the Diffuse and of the Translucent components.

Creating a fake Subsurface Scattering node group

In this recipe, we will create a fake Subsurface Scattering node group that can be mixed with other nodes to add the fake scattering effect to a material. In this screenshot, you can see the effect of the **Subsurface Scattering** node alone on the Suzanne mesh:

The rendered result of the fake SSS node group assigned to Suzanne

In the following screenshot, you can see the effect of the node group added to the usual basic shader material:



Mixed with a Diffuse-Glossy shader

Again, we will use the colors of the previous recipes.

Getting ready

Start Blender and open the `9931OS_07_start.blend` file.

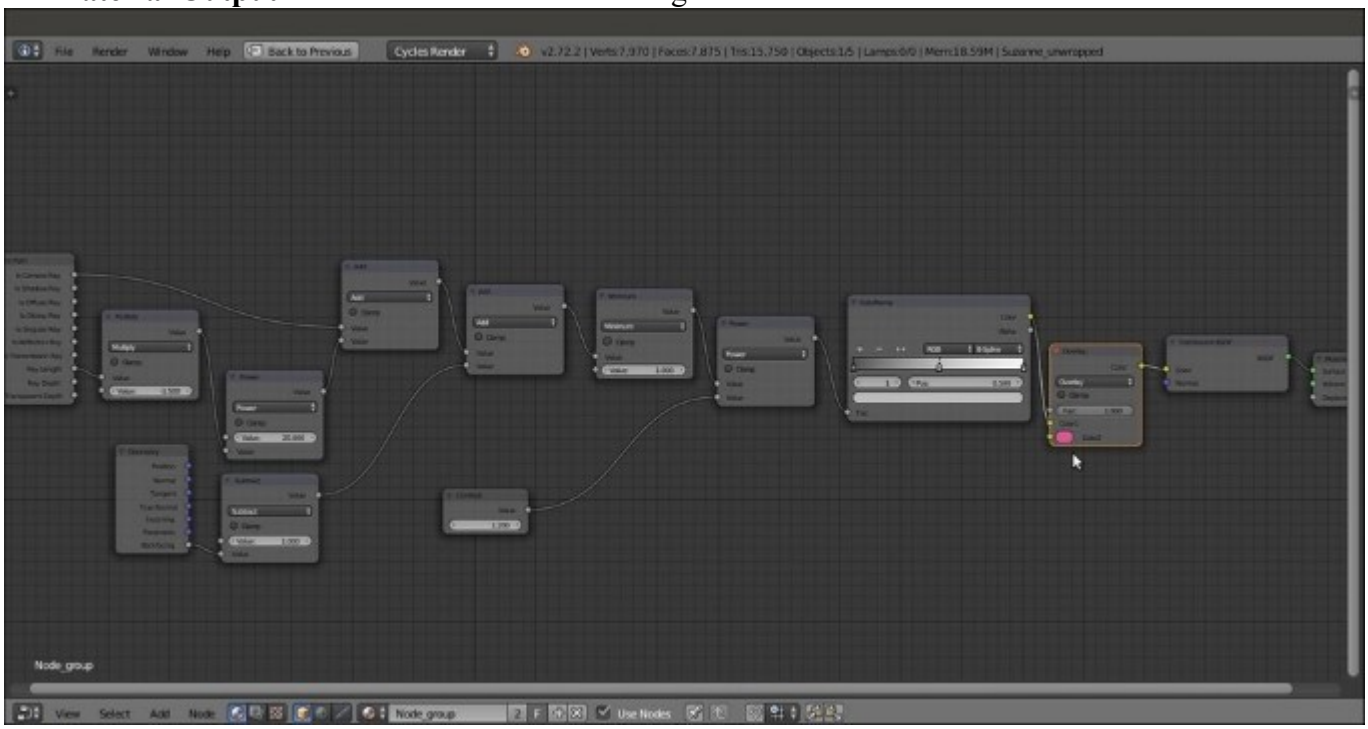
1. Go to the **Render** window, and in the **Sampling** subpanel, click on the **Method to sample lights and materials** button to switch from **Path Tracing** to **Branched Path Tracing**. Enable the **Square Samples** item and under **AA Samples**, set the **Render** value to `8`. Finally, click on the **Pattern** button to select the **Correlated Multi-Jitter** option.
2. Save the file as `9931OS_07_SSS_ngroup.blend`.

How to do it...

Now let's create the material using the following steps:

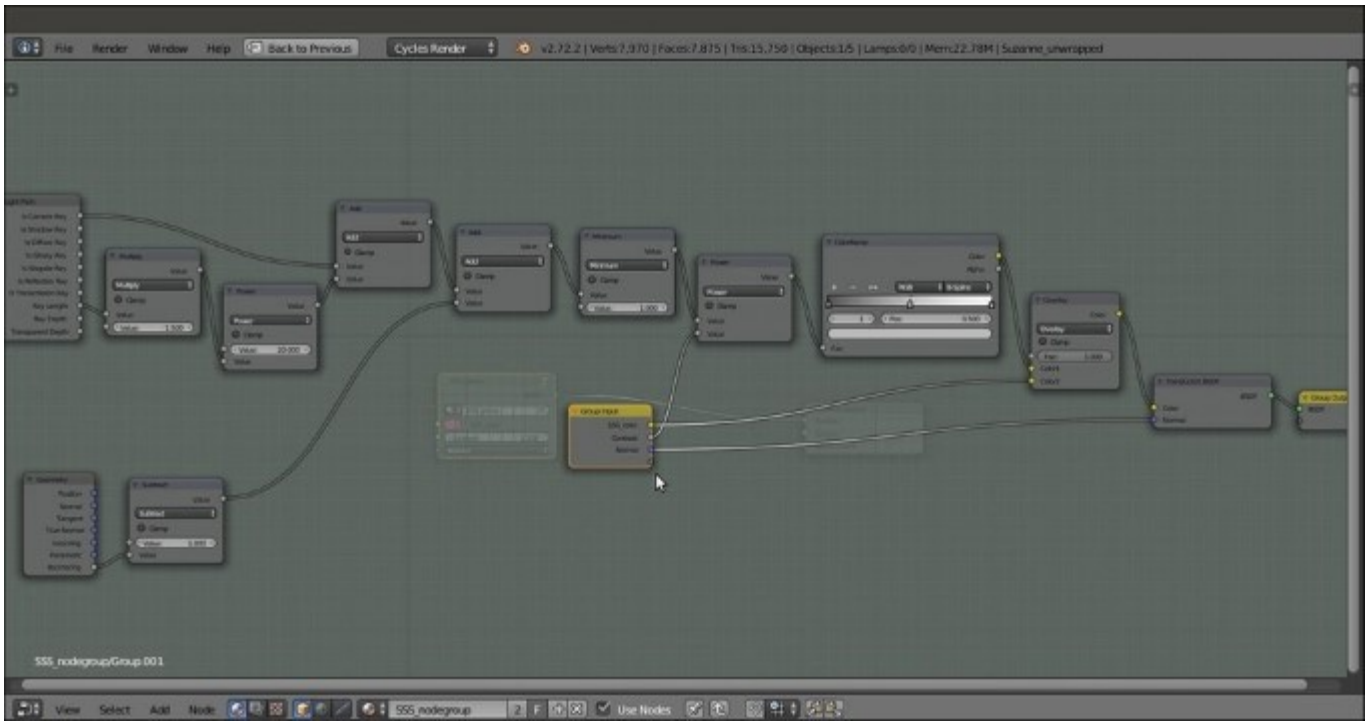
1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window under the main **Properties** panel. In the **Node Editor** window, delete the **Diffuse BSDF** shader node.
2. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**) and a **Geometry** node (press *Shift* + *A* and navigate to **Input** | **Geometry**).
3. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**). Set **Operation** to **Multiply** and connect the **Ray Length** output of the **Light Path** node to the first **Value** input socket. Set the second **Value** input socket to `-1.500`.
4. Press *Shift* + *D* to duplicate the **Multiply-Math** node, and set **Operation** to **Power**. Connect the **Multiply-Math** node output to the second **Value** input socket of the **Power** node. Set the first **Value** to `20.000`.
5. Press *Shift* + *D* to duplicate the **Power** node, and set **Operation** to **Add**. Connect the **Power** node output to the second **Value** input socket of the **Add-Math** node, and the **Is Camera Ray** output of the **Light Path** node to the first **Value** input socket of the **Add-Math** node.
6. Press *Shift* + *D* to duplicate the **Add** node. Set the **Operation** to **Minimum**. Connect the output of the **Add** node to the first **Value** input socket of the **Minimum** node, and set the second **Value** input socket to `1.000`.
7. Press *Shift* + *D* to duplicate the **Power** node, and place it after the **Minimum** node. Connect the output of the **Minimum** node to the first **Value** input socket of the duplicated **Power** node.
8. Add a **Value** node (press *Shift* + *A* and navigate to **Input** | **Value**), label it as `Contrast`, and connect its output to the second **Value** input socket of the last **Power-Math** node. Set **Value** to `1.200`.
9. Press *Shift* + *D* to duplicate any of the **Math** nodes, set **Operation** to **Subtract**, and connect the **Backfacing** output of the **Geometry** node to its second **Value** input socket. Set the first **Value** input socket to `1.000`.
10. Press *Shift* + *D* to duplicate the **Subtract** node, set **Operation** to **Add**, and paste it between the first **Add** and **Minimum** nodes. Connect the output of the **Subtract** node to the second **Value** input socket of the last **Add** node.

11. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and connect the output of the last **Power-Math** node to its **Fac** input socket. Add a **Translucent BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Translucent BSDF**) and connect the **Color** output of the **ColorRamp** node to its **Color** input socket.
12. Set the **ColorRamp** node's **Interpolation** to **B-Spline**. Click on the **Add** button to add a new stop with **Color** values for **R** as `0.500`, **G** as `0.500`, and **B** as `0.500` at the position of `0.500`.
13. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Set **Blend Type** to **Overlay** and the **Fac** value to `1.000`. Connect the **Color** output of the **ColorRamp** node to the **Color1** input socket. Set **Color2** values for **R** as `0.500`, **G** as `0.054`, and **B** as `0.077`.
14. Connect the **Color** output of the **Overlay** node to the **Color** input socket of the **Translucent BSDF** node. Connect the output of the **Translucent BSDF** node to the **Surface** input socket of the **Material Output** node as shown in the following screenshot:



The network to be grouped

15. Now select all the nodes except the **Value** and **Material Output** nodes. Press *Ctrl* + *G* to create a **Node Group**.
16. Rename the exposed input socket to the left of the node group as `Contrast`, set the value on the group interface, and then delete the original **Value** node.
17. Click and drag the **Color2** socket of the **Overlay** node to the empty socket on the **Group Input** node, and rename the exposed socket as `Subsurface Scattering_color`. Then click and drag the **Normal** socket of the **Translucent BSDF** shader node to the empty socket as shown in the following screenshot:



The network inside the open-for-editing node group

18. Press *Tab* to close the node group, and rename it `SSS_group`.

So now, we have made the Subsurface Scattering node group, ready to be mixed with any surface material.

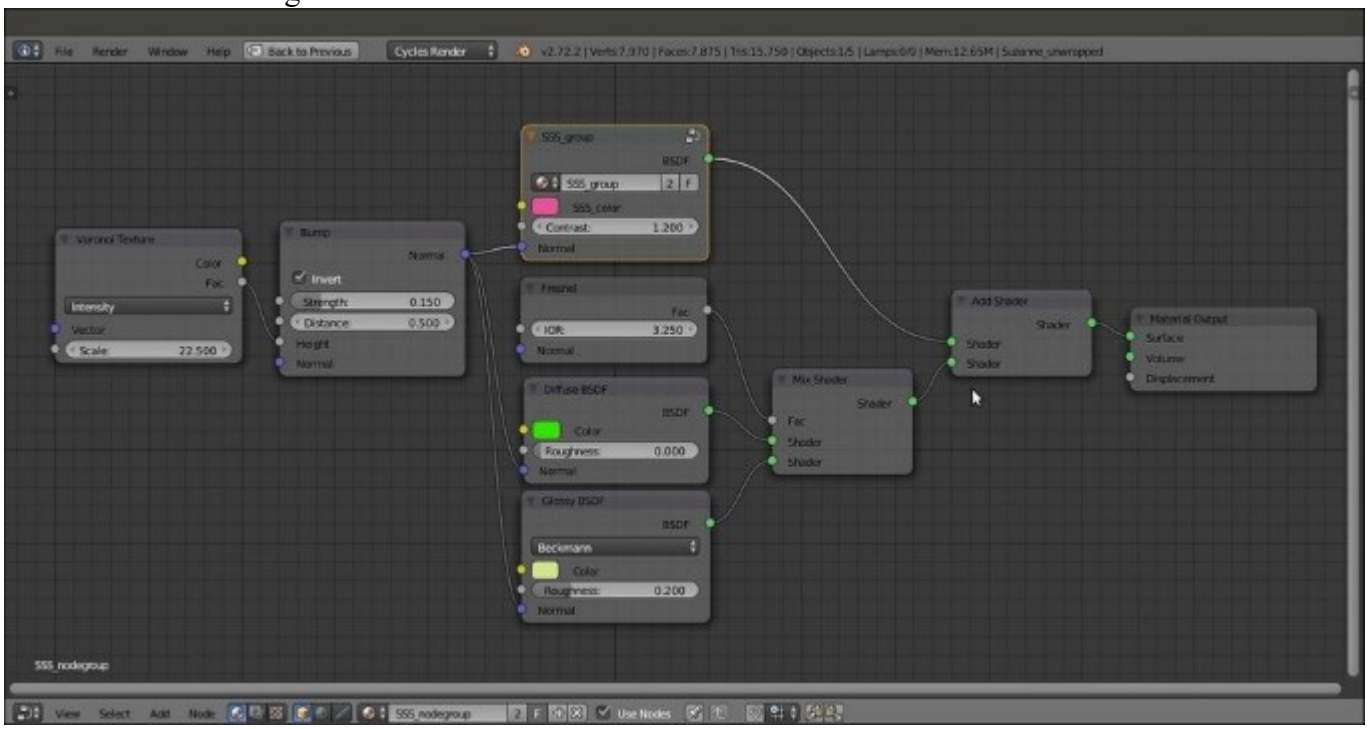
Let's now create a simple material to mix the node group using the following steps:

1. Add a **Mix Shader** node, a **Diffuse BSDF** node, and a **Glossy BSDF** shader node (press *Shift* + *A* and navigate to **Shader** | **...**). Connect the **Diffuse BSDF** node output to the first **Shader** input socket of the **Mix Shader** node and the **Glossy BSDF** shader output to the second **Shader** input socket.
2. Connect the **Mix Shader** output to the **Surface** input socket of the **Material Output** node.
3. Set the **Color** values of the **Diffuse BSDF** shader node for **R** to `0.031`, **G** to `0.800`, and **B** to `0.000`. Set the **Color** values of the **Glossy BSDF** node for **R** to `0.646`, **G** to `0.800`, and **B** to `0.267`.
4. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Fac** output of the texture node to the **Height** input socket of the **Bump** node, and the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF** and **Glossy BSDF** shaders, and also of the `SSS_group` node group.
5. Set the **Bump** node's **Strength** to `0.150` and enable the **Invert** item. Set the **Voronoi Texture** node's **Scale** value to `22.500`.

6. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader** node. Set the **IOR** value to `3.250`.

Now let's simply add the Subsurface Scattering node group:

1. Add an **Add Shader** node (press *Shift* + *A* and navigate to **Shader** | **Add Shader**) and paste it between the **Mix Shader** node and the **Material Output** node. Switch the connection from the first socket to the second socket (this is actually not required in this case because the shaders are added anyway).
2. Connect the output of the `SSS_group` node to the first socket of the **Add Shader** node as shown in the following screenshot:



The SSS node group added to the diffuse and glossy components of an average shader

How it works...

The key of this material is obviously the **Light Path** node, with its several kinds of output. In this case, we are interested in two of them:

- The **Ray Length** and **Is Camera Ray** output of the **Light Path** node are added together. **Ray Length** defines the thickness of the mesh, and it's also clamped by the first **Multiply** node and the **Power** node. The **Is Camera Ray** output gets Cycles to render only those surface points that are directly hit by light rays emerging from the Camera. When added to each other, the two types of output produce a stencil effect, gray-scale values distributed according to the thickness of the mesh.

- Next, the **Backfacing** output of the **Geometry** node is added to take into consideration the color of the back mesh faces. All of this is multiplied by the second **Power** node for the **Contrast** value and further clamped by the **ColorRamp** node.
- At this point, the result is mixed with the **Subsurface Scattering_color** output by the **Overlay** node, and finally connected to the **Color** input socket of the **Translucent BSDF** shader, resulting in the semi-transparent-looking shader of the first image at the beginning of this recipe.

Chapter 8. Creating Organic Materials

In this chapter, we will cover the following topics:

- Creating an organic-looking shader with procedurals
- Creating a wasp-like chitin material with procedural textures
- Creating a beetle-like chitin material with procedural textures
- Creating tree shaders – the bark
- Creating tree shaders – the leaves
- Creating a layered human skin material in Cycles
- Creating fur and hair
- Creating a gray alien skin with procedurals

Introduction

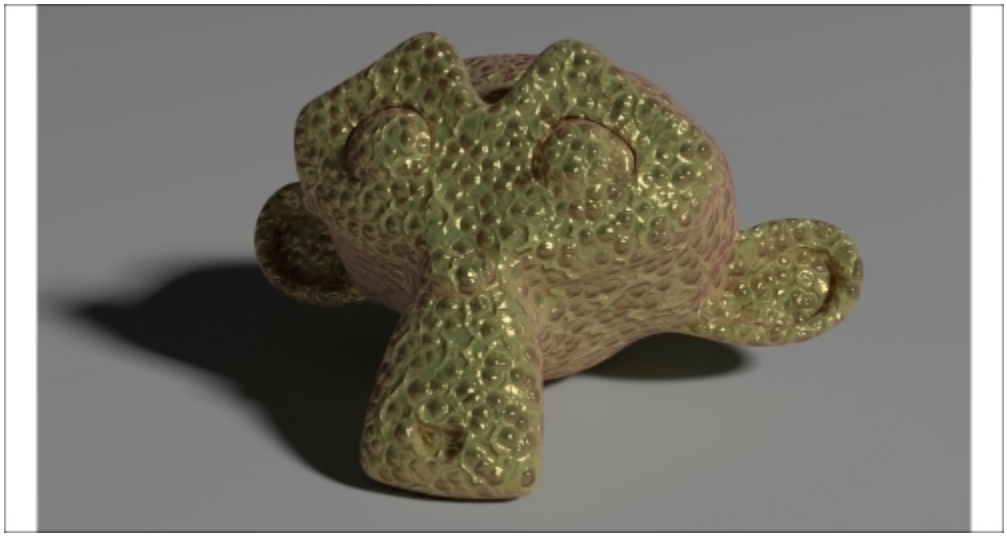
Following on from the natural materials we have seen in Chapter 3, *Creating Natural Materials in Cycles,* and in Chapter 5, *Creating Complex Natural Materials in Cycles*, it's now time to take a look at organic shaders.

Once again, while building the materials, we tried to use only the Cycles procedural textures. In several cases, this hasn't been the case by the way: on one side, because it hasn't been possible, and on the other side, because image maps usually work better than procedurals.

In any case, procedurals have often been added to the shader to refine the details or to add a natural-looking randomness to a pattern that repeats too much.

Creating an organic-looking shader with procedurals

In this recipe, we will create a sort of organic, disgusting-looking material, as shown in the following screenshot:



The disgusting organic material as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_08_start.blend` file, where there is an already set scene with an unwrapped **Suzanne** primitive object leaning on a Plane, an Emitter mesh-light, and a Camera.

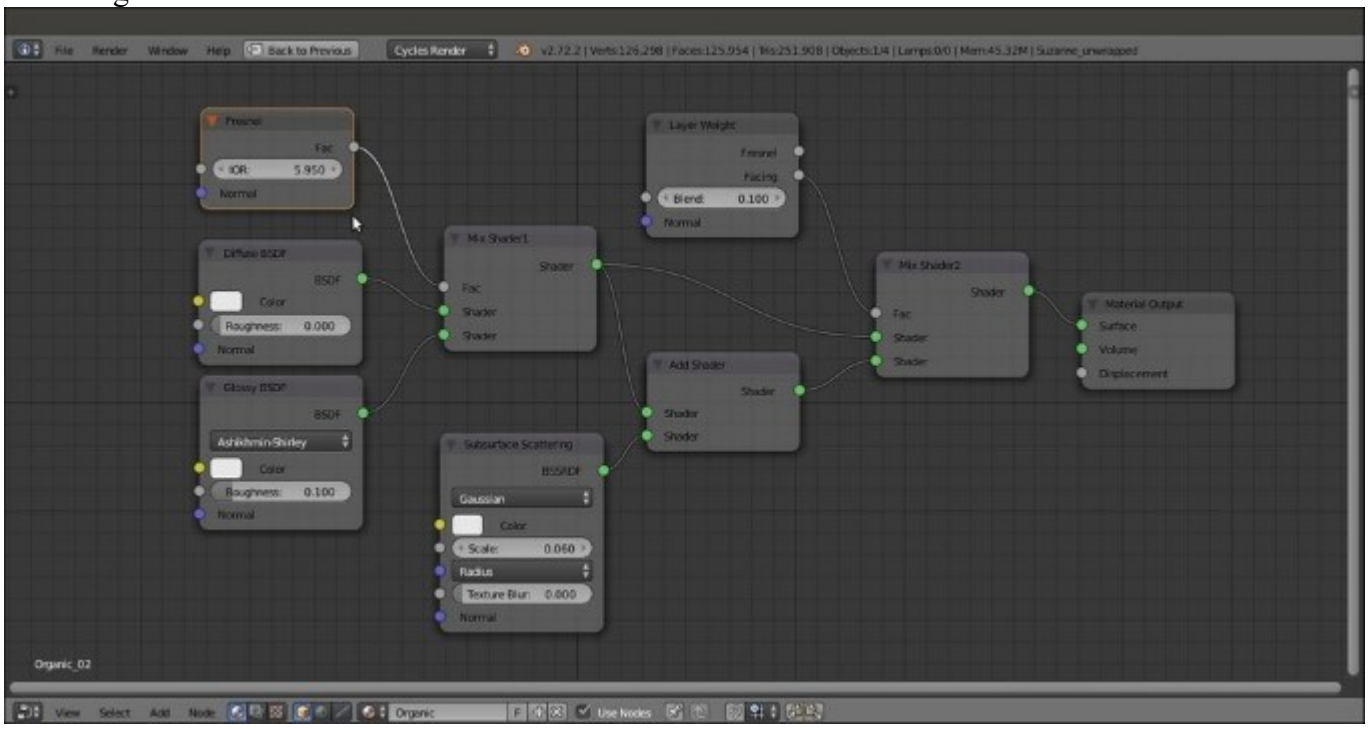
Go to the **Render** window, and in the **Sampling** subpanel, change **Pattern** from **Sobol** to **Correlated Multi-Jitter**.

How to do it...

Let's go straight to the material creation by using the following steps:

1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window under the main **Properties** panel and rename the new material `Organic`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and label it as `Mix Shader2`. In the first **Shader** input socket, select a **Mix Shader** node and label it as `Mix Shader1`, and in the second one, select an **Add Shader** node.
3. Go to the **Mix Shader1** node, and in the first **Shader** input socket, load a **Diffuse BSDF** node, and in the second one, load a **Glossy BSDF** node. Change the **Glossy BSDF** shader node's **Distribution** to **Ashikhmin-Shirley**, and set the **Roughness** value to `0.100`.

4. Add a **Subsurface Scattering** node (press *Shift* + *A* and navigate to **Shader** | **Subsurface Scattering**). Set the **Falloff** value to **Gaussian**, the **Scale** value to `0.060`, and the **Radius** values to `4.000`, `2.000`, and `1.000` (top to bottom).
5. Connect the **Mix Shader1** output to the first **Shader** input socket of the **Add Shader** node, and the output of the **Subsurface Scattering** node to the second **Shader** input socket of the **Add Shader** node.
6. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader2** node. Set the **Blend** value to `0.100`.
7. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect its output to the **Fac** input socket of the **Mix Shader1** node. Set the **IOR** value to `5.950` as shown in the following screenshot:

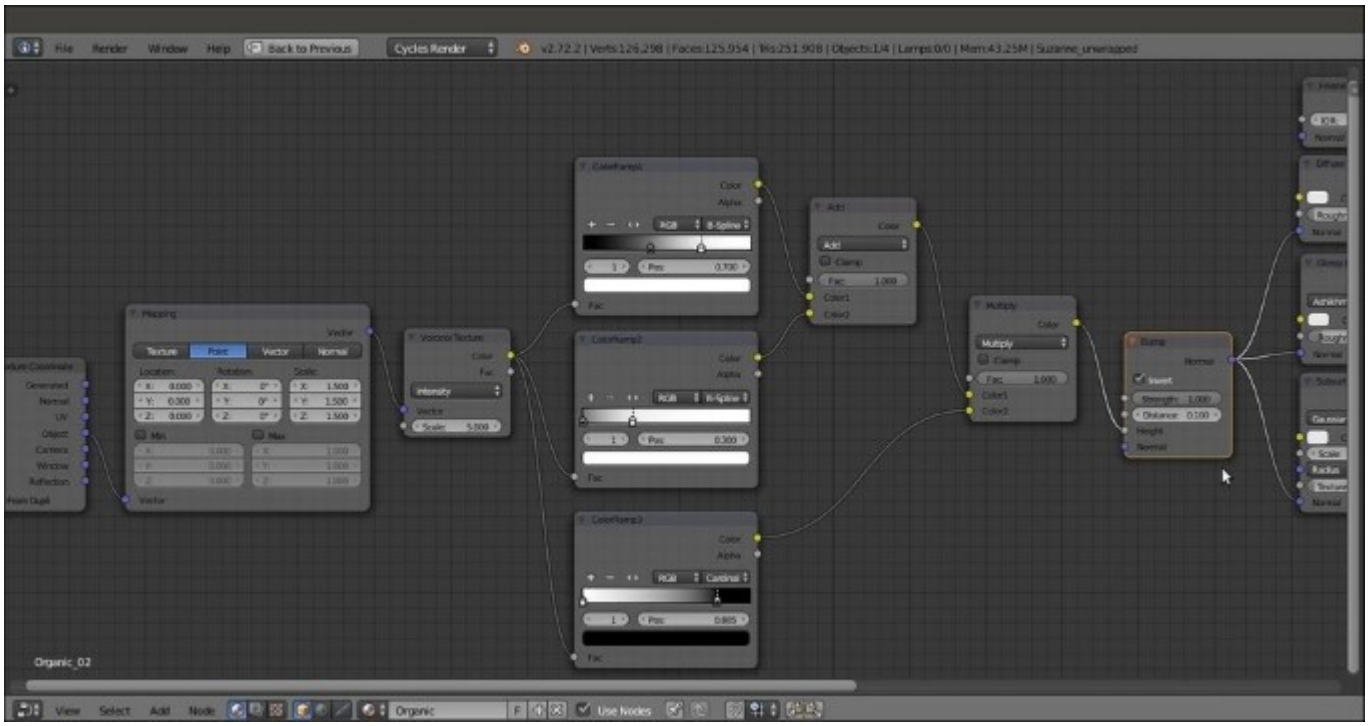


The basic shader nodes

8. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**). Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of this to the **Vector** input of the **Voronoi Texture** node. Set the **Scale** value of the **Mapping** node to `1.500` for the three axes.
9. Add three **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and label them as `ColorRamp1`, `ColorRamp2`, and `ColorRamp3`. Connect the **Color** output of the **Voronoi Texture** node to the **Fac** input sockets of the three **ColorRamp** nodes.
10. In the **ColorRamp1** node, set **Interpolation** to **B-Spline**, the black color stop to the `0.400` position, and the white color stop to the `0.700` position. In the **ColorRamp2** node, set

Interpolation to **B-Spline** as well. Leave the black color stop at the `0.000` position, and move the white color stop to the `0.300` position. In the **ColorRamp3** node, set **Interpolation** to **Cardinal**, leave the black color stop at the `0.000` position, and move the white color stop to the `0.805` position.

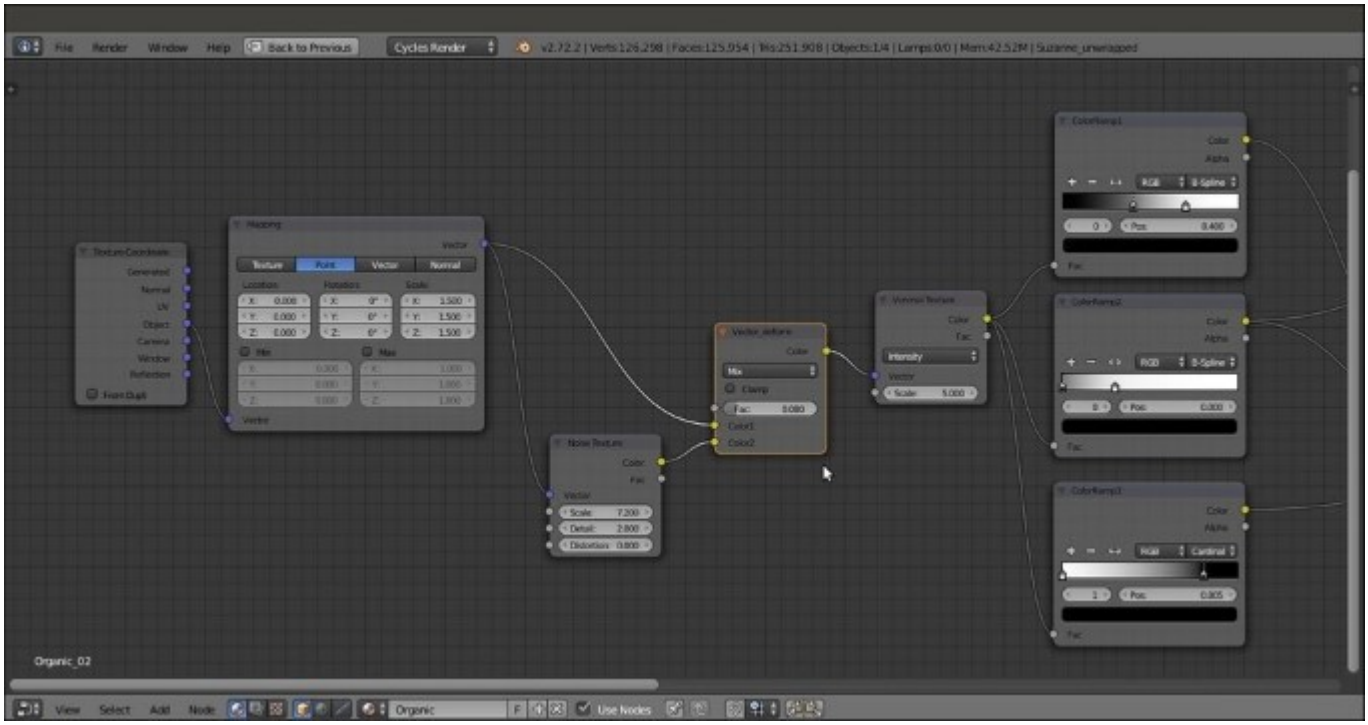
11. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set **Blend Type** to **Add** and the **Fac** value to `1.000`, and then connect the **Color** output of the **ColorRamp1** node to the **Color1** input socket, and the **Color** output of the **ColorRamp2** node to the **Color2** input socket.
12. Press *Shift* + *D* to duplicate the **Add** node and change **Blend Type** of the duplicate to **Multiply**. Connect the output of the **Add** node to the **Color1** input socket, and the **Color** output of the **ColorRamp3** node to the **Color2** input socket.
13. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the output of the **Multiply** node to the **Height** input socket of the **Bump** node. Connect the **Normal** output of this to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Subsurface Scattering** nodes. Enable the **Invert** option on the **Bump** node, as shown in the following screenshot:



The Bump node

14. Now, box-select (press the *B* key) the **Texture Coordinate** node and the **Mapping** nodes, and move them to the left to make room for new nodes.
15. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and label it as `Vector_deform`. Paste it between the **Mapping** and **Voronoi Texture** nodes.
16. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), connect to its **Vector** input socket the **Mapping** node output, and set the **Scale** value to `7.200`. Connect

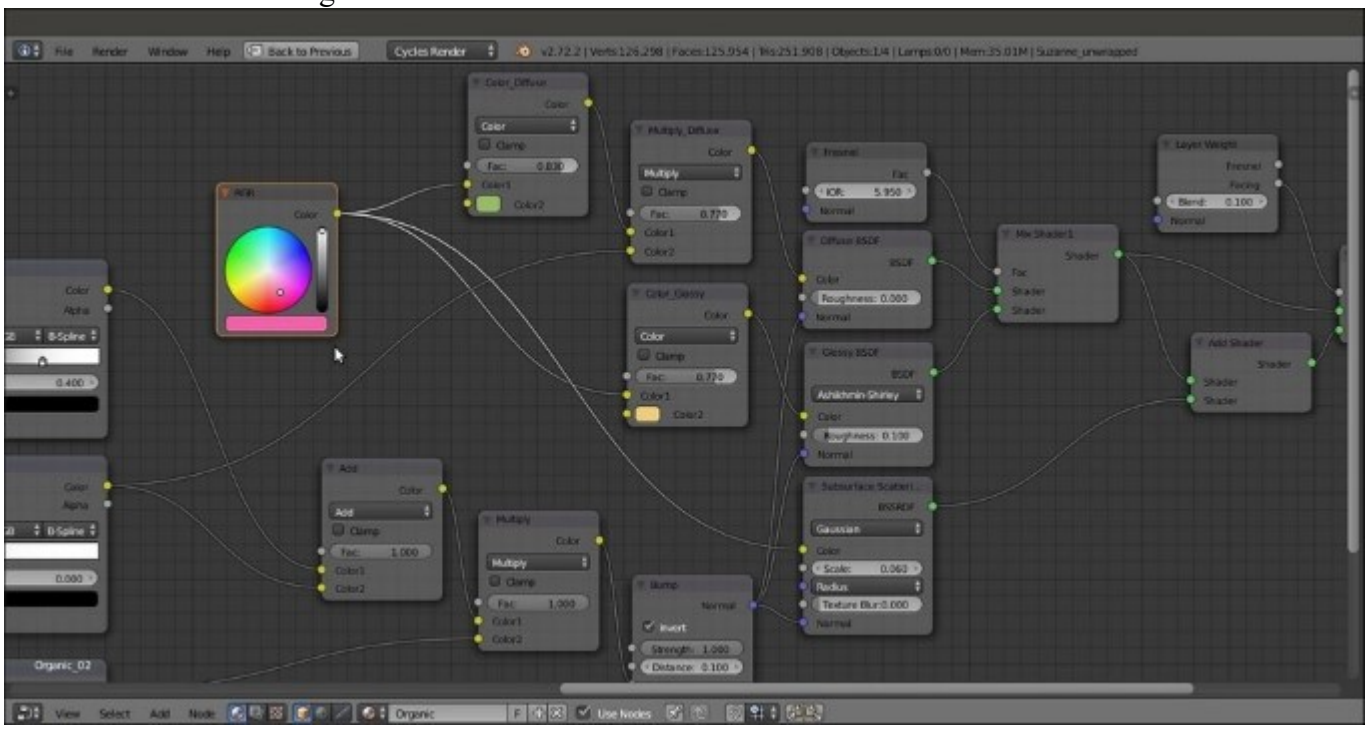
the **Noise Texture** node's **Color** output to the **Color2** input socket of the **Vector_deform** node. Set the **Fac** value of the **Vector_deform** node to `0.080`, as shown in the following screenshot:



Deforming the mapping coordinates of the bump textures through a procedural noise

17. Add an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**) and a new **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**). Label the **MixRGB** node
18. as `Color_Diffuse`.
19. Press *Shift* + *D* to duplicate the **Color_Diffuse** node and label the duplicate as `Color_Glossy`.
20. Connect the **Color_Diffuse** node's output to the **Color** input socket of the **Diffuse BSDF** shader node, and the **Color_Glossy** node's output to the **Color** input socket of the **Glossy BSDF** shader node.
21. Connect the output of the **RGB** node to the **Color1** input sockets of both the **Color_Diffuse** and **Color_Glossy** nodes. Connect the **RGB** node also to the **Color** input socket of the **Subsurface Scattering** node.
22. Press *Shift* + *D* to duplicate the **Color_Diffuse** node, set **Blend Type** of the duplicate to **Multiply**, and label it as `Multiply_Diffuse`; then, paste it between the **Color_Diffuse** and **Diffuse BSDF shader** nodes.
23. Connect the **Color** output of the **ColorRamp2** node to the **Color2** input socket of the **Multiply_Diffuse** node. Set the **Fac** value of this to `0.770`.
24. Go to the **Color_Diffuse** node and set the **Fac** value to `0.830`, and change the **Color2** value of **R** to `0.315`, **G** to `0.500`, and **B** to `0.130`.
25. Go to the **Color_Glossy** node and set the **Fac** value to `0.770`, and change the **Color2** values of **R** to `0.860`, **G** to `0.611`, and **B** to `0.203`.

26. Go to the **RGB** node and set the **Color** values for **R** to `0.900`, **G** to `0.123`, and **B** to `0.395`, as shown in the following screenshot:



Adding the color nodes

27. Save the file as `99310S_organic.blend`.

How it works...

- From step 1 to 7, we built a shader that is very similar to the shaders that we have already seen for `SSS_materials`.
- From step 8 to 13, we built the bump pattern by using a single **Voronoi Texture** node tuned through three **ColorRamp** nodes with different settings.
- From step 14 to 16, we added, through the very low value of a **MixRGB** node, the values of a **Noise Texture** node to the vector of the **Voronoi Texture** node to obtain a less regular pattern.
- From step 17 to 25, we built the color pattern by establishing a base color by the **RGB** node and introducing a variation through the **MixRGB** nodes connected to the **Color** input sockets of the shader components. Note that the base pink color set in the **RGB** node goes straight to the SSS node. The **MixRGB** varied greenish color is multiplied by one of the bump outputs and then goes to the diffuse component of the shader, while the varied yellowish color is for the glossy component instead.

Creating a wasp-like chitin material with procedural textures

In this recipe, we will create a material similar to chitin (the characteristic substance of the exoskeletons of insects) colored with a yellow and black pattern like a wasp, as shown in the following screenshot:

The insect wasp-like material as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_08_start.blend` file, where there is an already set scene with an unwrapped **Suzanne** primitive object leaning on a Plane, an Emitter mesh-light, and a Camera.

Go to the **World** window and enable the **Ambient Occlusion** item with the **Factor** value `0.10`.

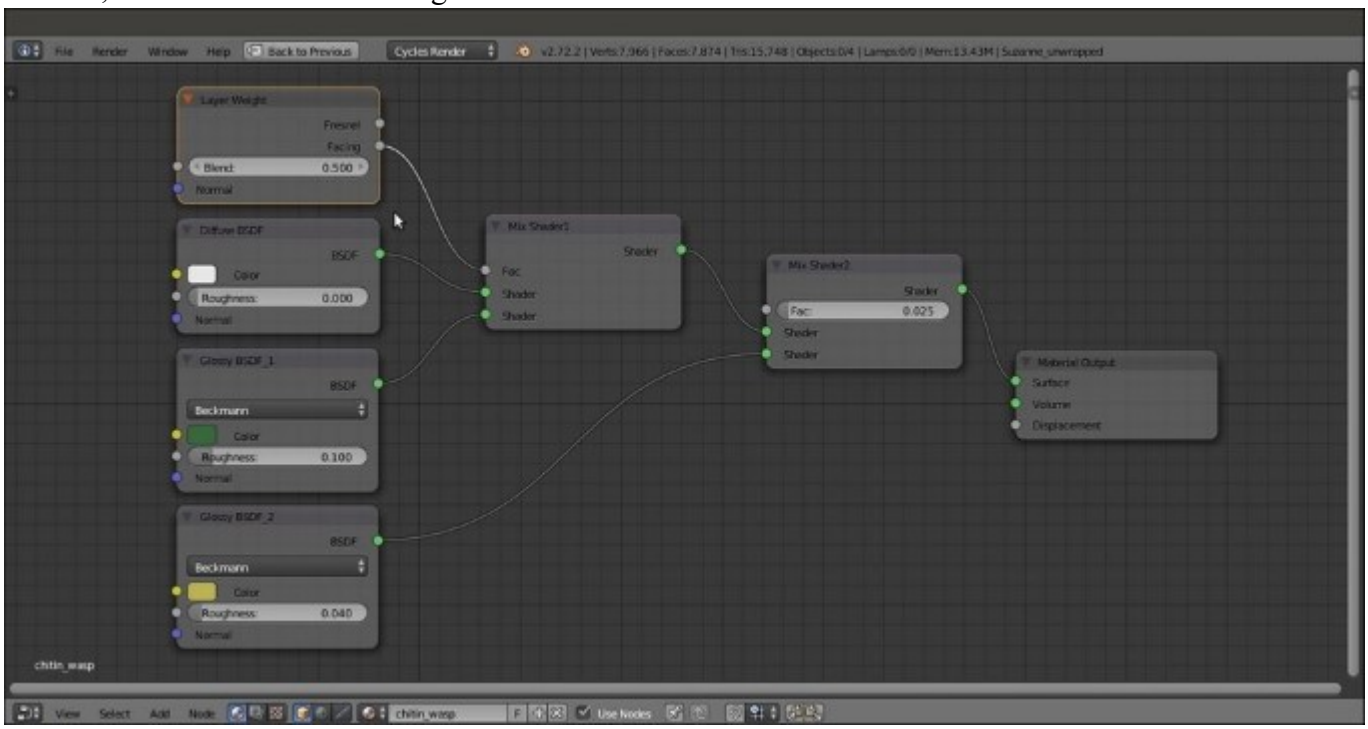
How to do it...

Let's start immediately with the material creation using the following steps:

1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window under the main **Properties** panel to the right, and rename the new material `chitin_wasp`.
2. Now, in the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and label it as `Mix Shader2`. In the first **Shader** slot, select a new **Mix Shader** node. In the second one, select a **Glossy BSDF** shader node. Label the new **Mix Shader** node as `Mix Shader1`, and the **Glossy BSDF** node as `Glossy BSDF_2`.
3. Go to the **Mix Shader1** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader, and in the second one, select a new **Glossy BSDF** shader node. Label the latter as `Glossy BSDF_1`,

and set its **Roughness** value to `0.100` and **Distribution** to **Beckmann**, and change the **Color** value for **R** to `0.039`, **G** to `0.138`, and **B** to `0.046`.

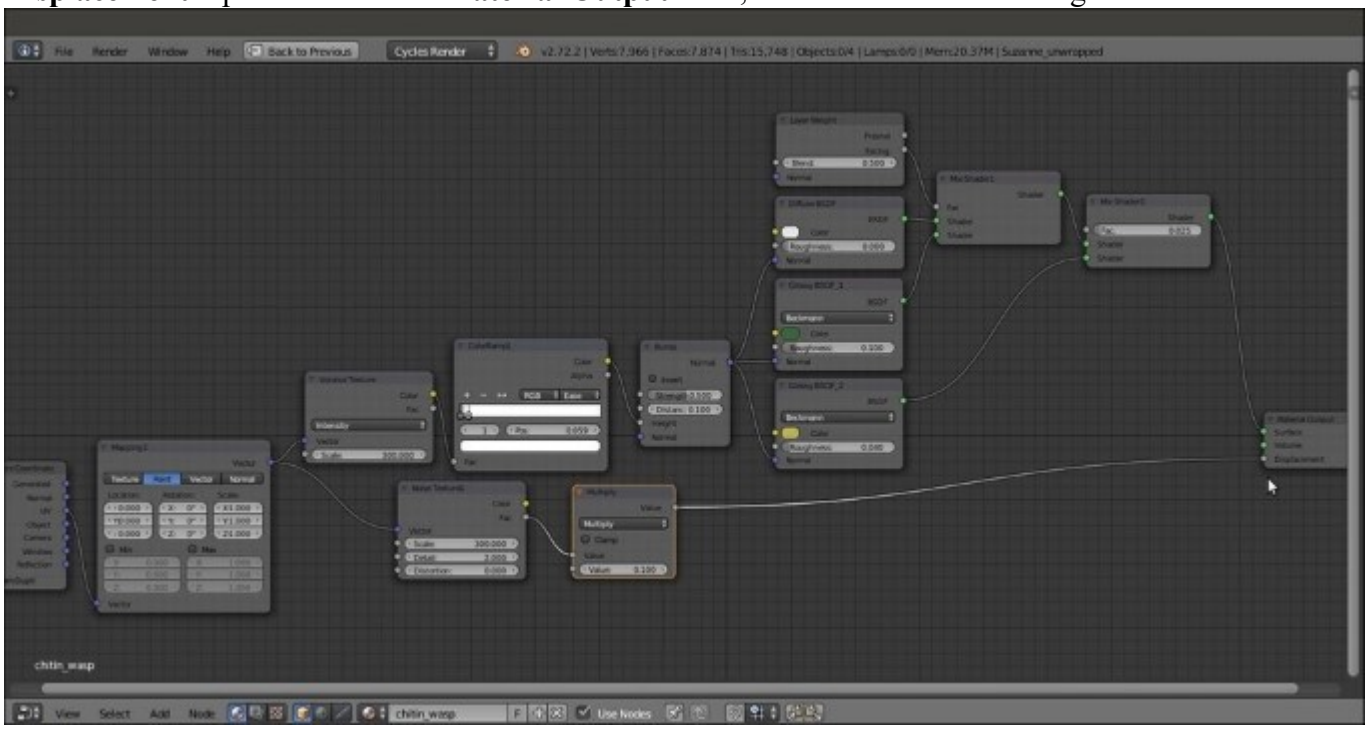
4. Set the **Glossy BSDF_2** node's **Roughness** value to `0.040` and **Distribution** to **Beckmann**, and change its **Color** values for **R** to `0.500`, **G** to `0.440`, and **B** to `0.086`. Set the **Fac** value of the **Mix Shader2** node to `0.025`.
5. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader1** node. Leave the **Blend** value as `0.500`, as shown in the following screenshot:



The nodes for the base shader

6. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node. Label the latter as `Mapping1`.
7. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**). Connect the **Mapping1** node's **Vector** output to their **Vector** input sockets. Set the **Scale** values of both the texture nodes to `300.000` and then label the **Noise Texture** node as `Noise Texture1`.
8. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **Color** output of the **Voronoi Texture** node to the **Height** input socket of the **Bump** node. Connect the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF** node and both **Glossy BSDF** shader nodes. Set the **Bump** node's **Strength** value to `0.500`.
9. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp1`, and paste it between the **Voronoi Texture** node and the **Bump** node. Set **Interpolation** to **Ease** and move the white color stop to the `0.059` position.

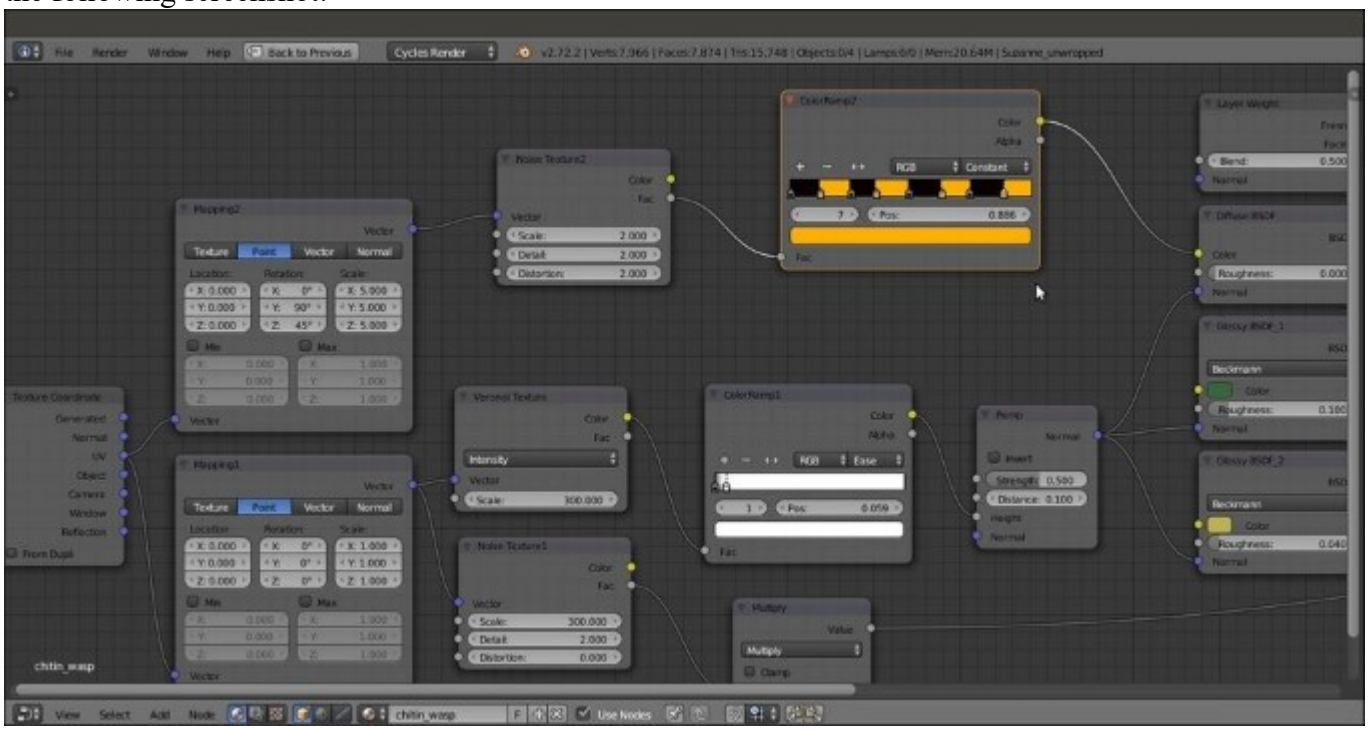
10. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), set **Operation** to **Multiply**, and connect the **Fac** output of the **Noise Texture1** node to the first **Value** input socket of the **Math** node. Set the second **Value** to `0.100` and connect the **Value** output to the **Displacement** input socket of the **Material Output** node, as shown in the following screenshot:



Textures connected either as per the shader bump and the total bump to the Displacement input socket of the Material Output node

11. Add a new **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), label it as `Mapping2`, and connect the **UV** output of the **Texture Coordinate** node to its **Vector** input socket. Set the **Rotation** value for **Y** to `90°` and the **Rotation** value of **Z** to `45°`. Set the **Scale** value for all three axes to `5.000`.
12. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Label them as `Noise Texture2` and `ColorRamp2`.
13. Connect the output of the **Mapping2** node to the **Vector** input socket of the **Noise Texture2** node, and the **Fac** output of this node to the **Fac** input socket of **ColorRamp2**. Connect the output of this node to the **Color** input socket of the **Diffuse BSDF** shader node.
14. Go to the **Noise Texture2** node and set the **Scale** and **Distortion** values to `2.000`. Go to the **ColorRamp2** node and set **Interpolation** to **Constant**, select the white color stop, and change the **Color** values for **R** to `1.000`, **G** to `0.429`, and **B** to `0.000`.
15. Click on the **+** icon button to add new color stops until you have eight color stops almost evenly spaced along the slider (that is: color stop **0** at the `0.000` position, **1** at the `0.125` position, **2** at the `0.250` position, then `0.357`, `0.491`, `0.626`, `0.745`, and `0.886`).
16. Select the last color stop, put the mouse pointer on the color slider, and press *Ctrl* + *C* to copy the yellow color; then, select the color stops numbered **1**, **3**, and **5**, and paste the color (press

Ctrl + V) so as to have a slider subdivided in eight parts, four black and four yellow, as shown in the following screenshot:



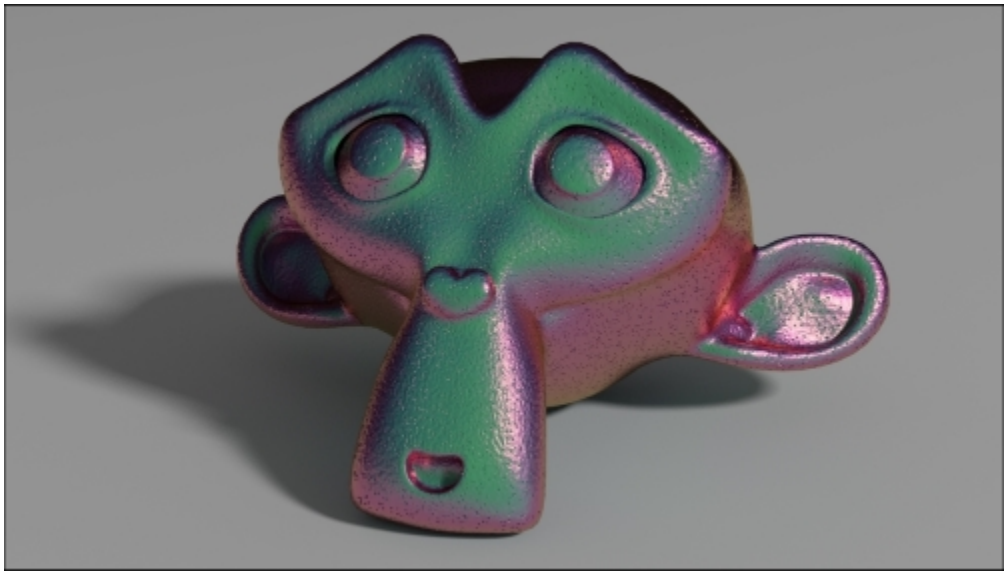
The color pattern connected to the diffuse component

How it works...

- From step 1 to 5, we built the basic shader using two **Glossy BSDF** shaders with different colors to mimic a color shifting in the specularity areas.
- From step 6 to 10, we built the chitin bump, assigning the pores to the per-shader bump but a general noise pattern to the displacement output (which, in this case, still works as a simple bump).
- From step 11 to 16, we built a simple and random wasp-colored pattern; obviously, this can be changed and modified as you prefer, and actually should also be used on a more appropriate model; in this case, it would be better to make use of a painted color texture map to build a more appropriate and symmetrical color pattern.

Creating a beetle-like chitin material with procedural textures

In this recipe, we will create a material similar to iridescent chitin (found in some kinds of beetles), as shown in the following screenshot:



The beetle chitin-like material as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_08_start.blend` file, where there is an already set scene with an unwrapped **Suzanne** primitive object leaning on a Plane, an Emitter mesh-light, and a Camera.

Go to the **World** window and enable the **Ambient Occlusion** option with the **Factor** value as `0.10`.

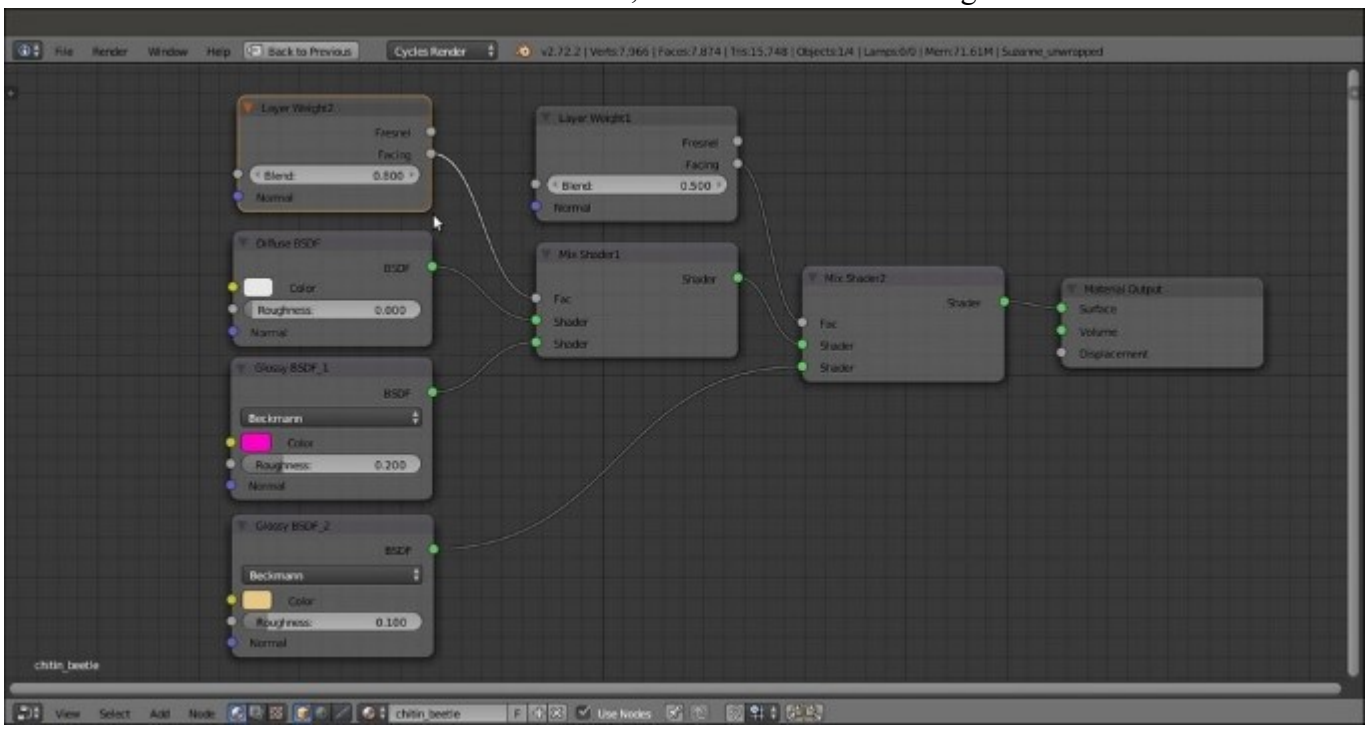
How to do it...

Let's start immediately with the material creation using the following steps:

1. Click on the **New** button in the **Node Editor** window's toolbar or in the **Material** window under the main **Properties** panel to the right, and rename the new material as `chitin_beetle`.
2. Now, in the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node and label it as `Mix Shader2`. In the first **Shader** slot, select a new **Mix Shader** node; in the second **Mix Shader** node, select a **Glossy BSDF** shader node. Label the new **Mix Shader** node as `Mix Shader1`, and the **Glossy BSDF** one as `Glossy BSDF_2`.
3. Go to the **Mix Shader1** node, and in the first **Shader** slot, select a **Diffuse BSDF** shader, and in the second one, select a new **Glossy BSDF** shader node; label this node as `Glossy BSDF_1`

and set its **Roughness** value to `0.200` and **Distribution** to **Beckmann**, and change the **Color** values for **R** to `1.000`, **G** to `0.000`, and **B** to `0.562`.

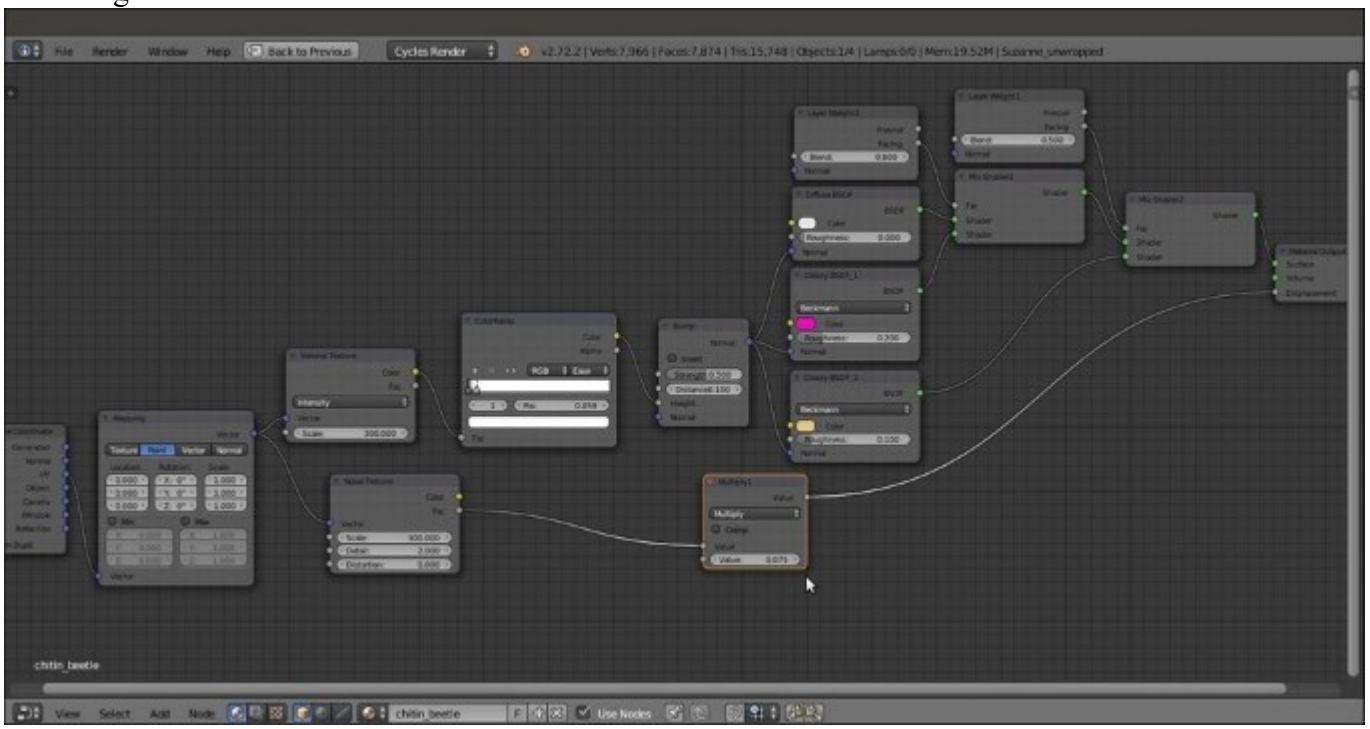
4. Set the **Glossy BSDF_2** node's **Roughness** value to `0.100` and **Distribution** to **Beckmann**, and change its **Color** values for **R** to `0.800`, **G** to `0.574`, and **B** to `0.233`.
5. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), label it as `Layer Weight1`, and connect its **Facing** output to the **Fac** input socket of the **Mix Shader2** node. Leave the **Blend** value at `0.500`.
6. Add a second **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), label it as `Layer Weight2`, and connect its **Facing** output to the **Fac** input socket of the **Mix Shader1** node. Leave the **Blend** value at `0.800`, as shown in the following screenshot:



The shader part of the material

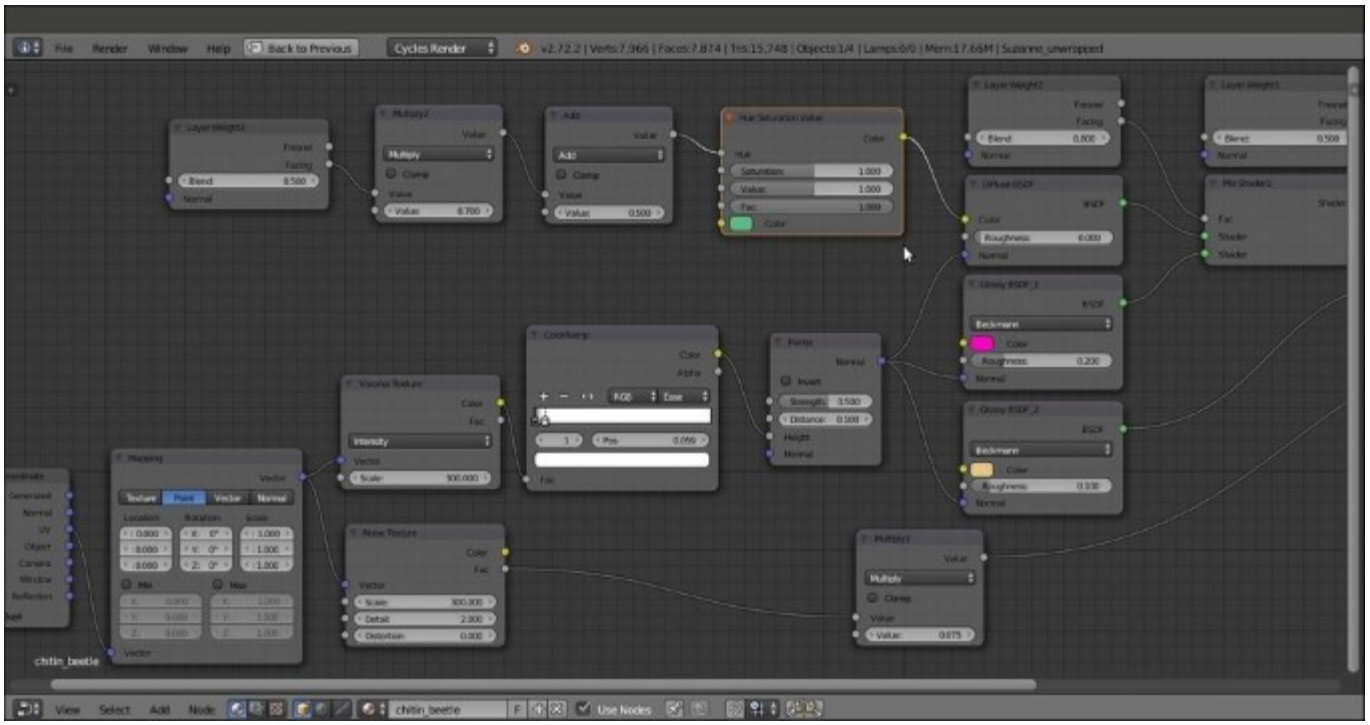
7. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**). Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input of the **Mapping** node.
8. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**); connect the **Mapping** output to their **Vector** input sockets. Set the **Scale** values of both the texture nodes to `300.000`.
9. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **Color** output of the **Voronoi Texture** node to the **Height** input socket of the **Bump** node; connect the **Normal** output of this node to the **Normal** input sockets of **Diffuse BSDF** and of both the **Glossy BSDF** shader nodes. Set the **Bump** node's **Strength** value to `0.500`.

10. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and paste it between the **Voronoi Texture** node and the **Bump** node. Set **Interpolation** to **Ease** and move the white color stop to the `0.059` position.
11. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), set **Operation** to **Multiply**, and label it as `Multiply1`; connect the **Fac** output of the **Noise Texture** node to the first **Value** input socket of the **Math** node. Set the second **Value** to `0.075` and connect the **Value** output to the **Displacement** input socket of the **Material Output** node, as shown in the following screenshot:



The bump is both "per shader" and as "total" bump (as in the previous wasp material recipe)

12. Add a new **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**), two **Math** nodes (press *Shift* + *A* and navigate to **Converter** | **Math**), and a **Hue Saturation Value** node (press *Shift* + *A* and navigate to **Color** | **Hue/Saturation**); label the new **Layer Weight** node as `Layer Weight3`.
13. Connect the **Facing** output of the **Layer Weight3** node to the first **Value** input socket of one of the **Math** nodes; set its **Operation** to **Multiply** and the second **Value** to `0.700`, and label it as `Multiply2`.
14. Connect the **Multiply2** node's output to the first **Value** input socket of the second **Math** node, and the output of this node to the **Hue** input socket of the **Hue Saturation Value** node; connect the output of this node to the **Color** input socket of the **Diffuse BSDF** shader node.
15. Change the **Hue Saturation Value** node's **Color** values for **R** to `0.103`, **G** to `0.500`, and **B** to `0.229`, and just for this example, leave the other values as they are, as shown in the following screenshot:



Adding the final diffuse color

How it works...

- The introductory steps of this shader work almost the same as for the `chitin_wasp` material, that is, the basic shader from step 1 to 6 and the chitin bump from step 7 to 11.
- From step 12 to 15, we build the color component coming from the **Hue Saturation Value** node, and thanks to the combination of the **Layer Weight3** and **Math** nodes, this appears mainly in the mesh faces perpendicular to the point of view, sliding in the other spectrum colors on the facing-away mesh sides, basically behaving as a sort of Fresnel effect. The addition of the **Hue Saturation Value** node allows for further color tweaking.

Creating tree shaders – the bark

There are several different ways to make trees in a 3D package: starting from the simpler low-poly objects, such as the billboards used in video games (simple planes mapped with tree images on a transparent background), to middle complex objects where a trunk mesh is attached to a foliage mass made of little alpha textured planes, each one representing a leaf or even a twig, to more complex and heavy meshes, where every little branch and leaf is actually modeled.

In case you need them, you can find several free tree models in the Blender format and also their billboard versions at http://yorik.uncreated.net/greenhouse.html.

For this two-part tree shader recipe, we will instead use a model coming from the many environment assets of the CG short *Big Buck Bunny*, the second open movie produced by the Blender Foundation. All the movie assets are free to be downloaded, distributed, and reused even for commercial projects because the short is licensed under the Creative Commons Attribution 3.0 license (refer to its official website at http://creativecommons.org/licenses/by/3.0/).

The general shape of the tree and the leaves is pretty toyish. This is because they are elements that have been drawn to match the toon style of the furry characters, but it's actually perfectly suited for our demonstration purposes. The final rendered tree from *Big Buck Bunny* is shown in the following screenshot for your reference:

The final rendered tree from Big Buck Bunny

The tree model is composed of several parts: on the first layer, there are the **tree_trunk**, the **tree_branch**, and the **tree_branches** meshes, and on the second layer are the leaves, made by a single leaf object dupliverted on the tiny faces of the **leaves_dupli** object. (That is, the **leaf_tobeswitched** object is parented to the **leaves_dupli** object, and then, in the **Object** window and under the **Duplication** subpanel, the **Faces** duplication method has been selected, the **Scale** item checked, and the **Inherit Scale** value set to `1110.000`. This way, the **leaf_tobeswitched** object is instanced on the **leaves_dupli** object's many faces according to their location, rotation, and scale.)

On the 11th layer, there are three leaf objects with three different levels of detail: a simple flat Plane, a subdivided and curved Plane, and a modeled leaf. Their presence is only to supply the low, middle, and high resolution mesh data. By selecting the **leaf_tobeswitched** object and by going to the **Object data** window, it is possible to switch between the **leaf_generic_low**, **leaf_generic_mid**, and **leaf_generic_hi** foliage levels of detail.

In the first part of this two-part recipe, we will create the material for the bark, as shown in the following screenshot:

The bark material

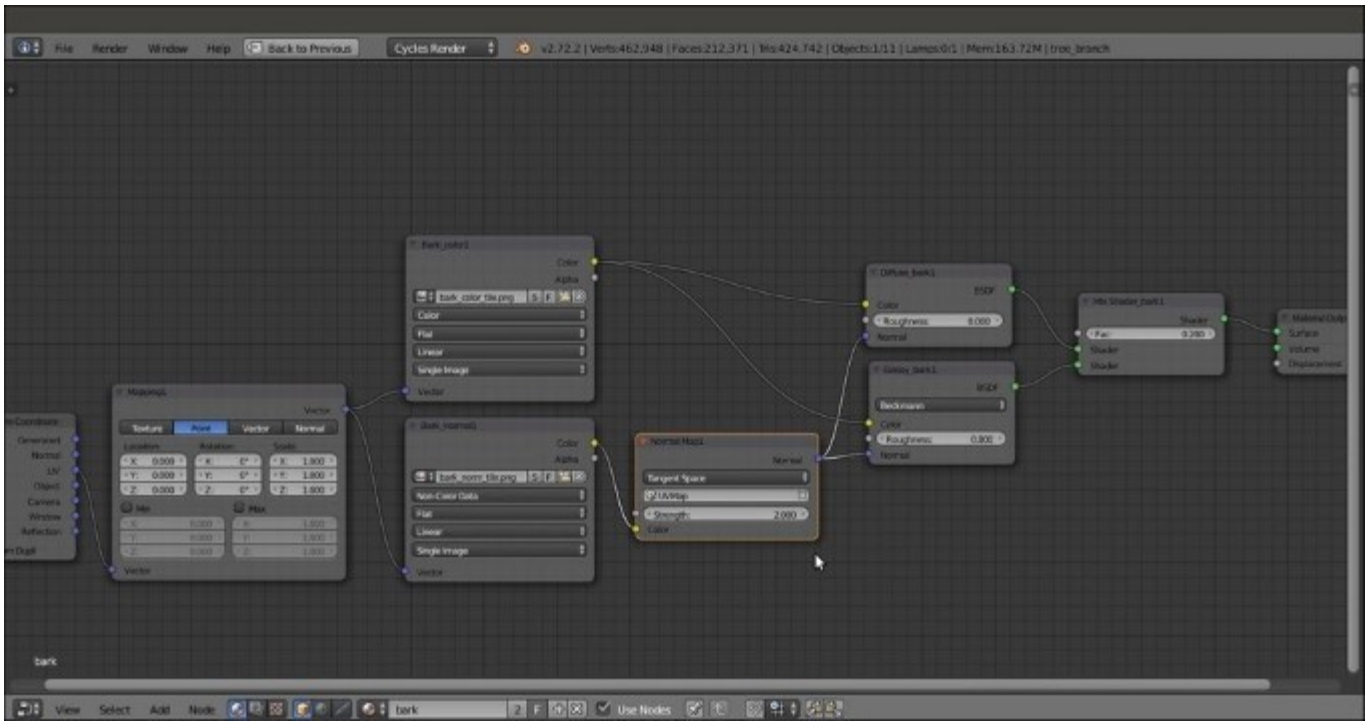
Getting ready

Start Blender and open the `9931OS_08_tree_start.blend` file. For this recipe, deactivate the second layer, and in **Outliner**, select the **tree_trunk** object.

How to do it...

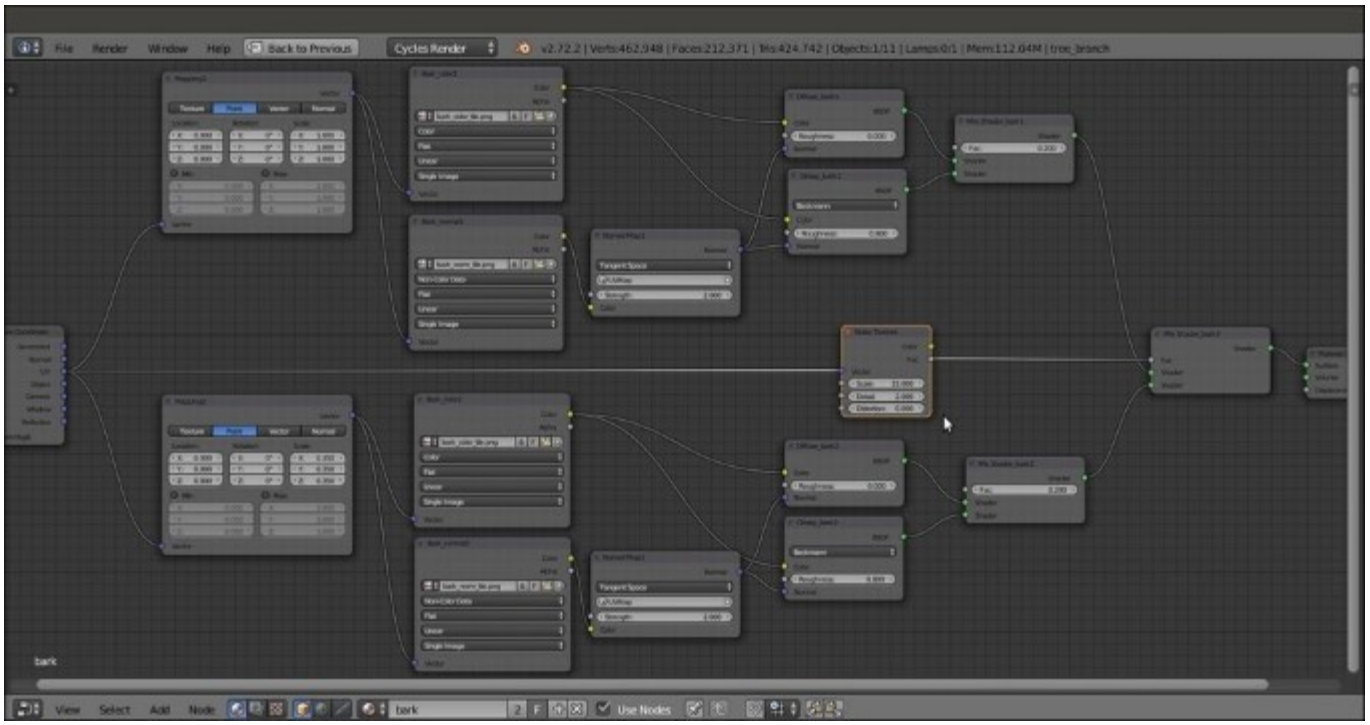
Let's start by creating the bark material using the following steps:

1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window, and rename the material as `bark`.
2. Still in the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node, and label it as `Mix Shader_bark1`. In the first **Shader** slot, select a **Diffuse BSDF** shader node, and in the second one, select a **Glossy BSDF** shader node; then, label them as `Diffuse_bark1` and `Glossy_bark1`. Set the **Glossy_bark** distribution to **Beckmann**, the **Roughness** value to `0.800`, and the **Mix Shader_bark1** node's **Fac** value to `0.200`.
3. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**); label the last two as `Mapping1` and `Bark_color1`.
4. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping1** node, and the output of this node to the **Vector** input socket of the **Bark_color1** node. Connect the **Color** output of the **Bark_color1** node to the **Color** input sockets of both the **Diffuse_bark1** and **Glossy_bark1** shader nodes.
5. Click on the **Open** button of the **Bark_color1** node, browse to the `textures` folder, and load the `bark_color_tile.png` image.
6. Press *Shift* + *D* to duplicate the **Bark_color1** node, label it as `Bark_normal1`, and connect the **Mapping1** node output to its **Vector** input socket. Make the image datablock single-user by clicking on **2**, which appears on the right side of the image name. Click on the **Open Image** button (the one with the folder icon), browse again to the `textures` folder, and load the `bark_norm_tile.png` image. Set **Color Space** to **Non-Color Data**.
7. Add a **Normal Map** node (press *Shift* + *A* and navigate to **Vector** | **Normal Map**), label it as `Normal Map1`, and connect the **Color** output of the **Bark_normal1** node to the **Color** input socket of the **Normal Map1** node, and then set the **Strength** value to `2.000`. Click on the **UV Map for tangent space maps** button upwards of the **Strength** one and select **UVMap** (the trunk mesh has two different sets of UV coordinates, which we'll see later).
8. Connect the **Normal** output of the **Normal Map1** node to the **Normal** input sockets of both the **Diffuse_bark1** and the **Glossy_bark1 shader** nodes, as shown in the following screenshot:



The basic bark material that uses a normal map

9. Now, box-select (press the *B* key and then draw a rectangle) all the nodes except for the **Texture Coordinate** and **Material Output** nodes and press *Shift* + *D* duplicate them. Move them down and change their labels by substituting the 1 suffix with 2. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the duplicated **Mapping2** node, and set the **Scale** of this node to `0.350` for all three axes.
10. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), label it as `Mix Shader_bark3`, and paste it right before the **Material Output** node. Connect the output of the **Mix Shader_bark2** node to the second **Shader** input socket of the **Mix Shader_bark3** node.
11. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**), connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Noise Texture** node, and connect the **Fac** output of this node to the **Fac** input socket of the **Mix Shader_bark3** node.
12. Set the **Noise Texture** node's **Scale** value to `15.000`, as shown in the following screenshot:



Making the bark material a bit more complex

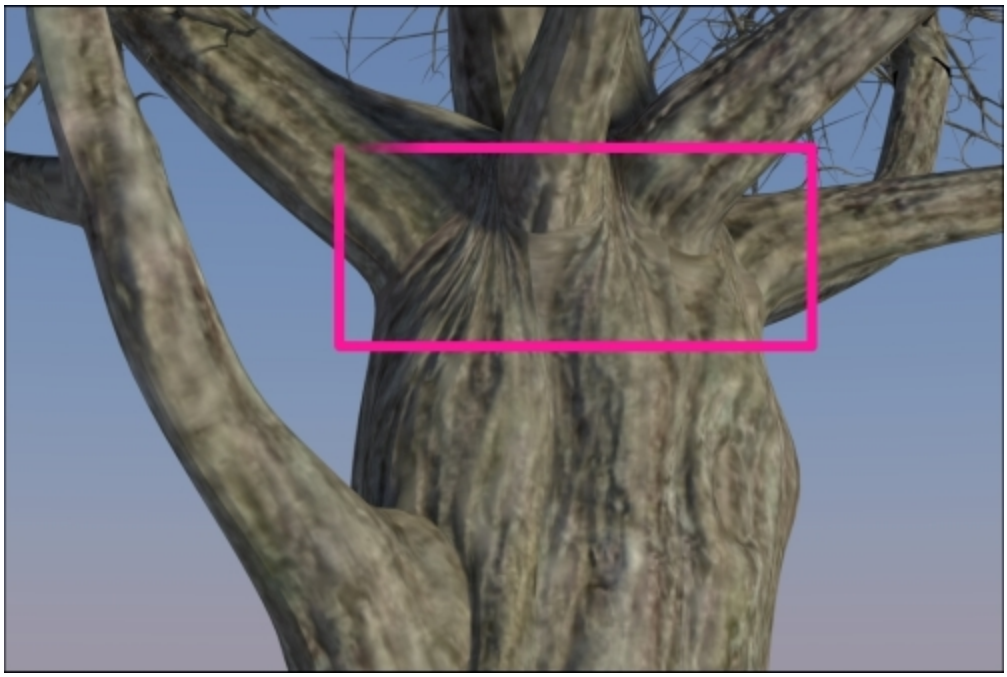
13. Now, press *Shift* and select the **tree_branch** and **tree_branches** meshes, and as the last one, reselect the **tree_trunk** mesh to make it the active object; then, press *Ctrl + L*. In the **Make Links** pop-up menu, select the **Materials** item to assign the bark material to the other two meshes.

How it works...

- For this material, we built a simple shader using two tileable image maps, a color one for the Diffuse and the Glossy components, and a normal map for the bump.
- Then, we duplicated everything and mixed the second material copy with different scale values to the first one by the factor of a Noise procedural texture, to add variety to the bark pattern and to avoid that unpleasant repeating effect that often shows up with tileable image textures.

There's more...

At this point, if you look carefully at the **Rendered** view of the tree trunk, you'll see that sadly, there are ugly seams where the trunk's main body joins the big low branches as shown in the following screenshot:

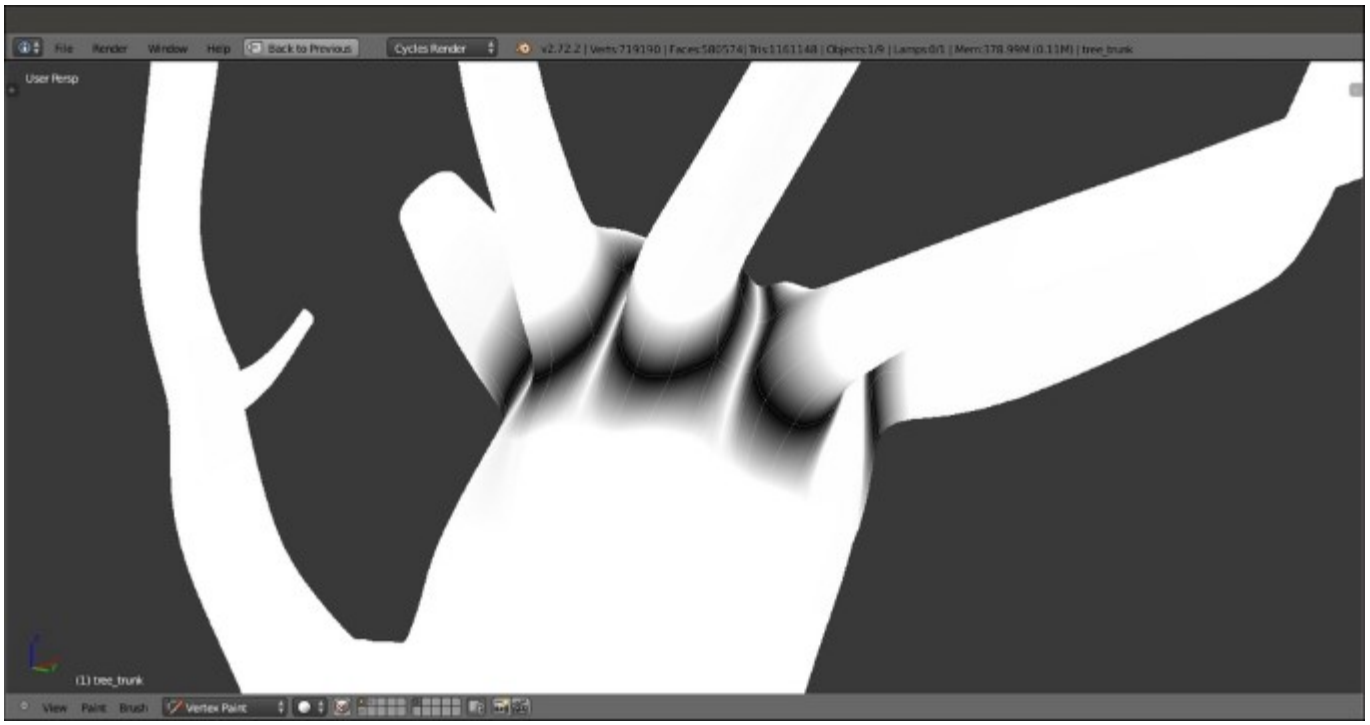


The visible seams at the branches joining

This is due to the fact that the unwrap of the mesh has separated the branches' UV islands from the main trunk ones. Although the effect can be barely visible, let's say that you absolutely want to avoid this; that's why we are now going to see a solution for the problem, by using a second set of UV coordinates and a Vertex Color layer.

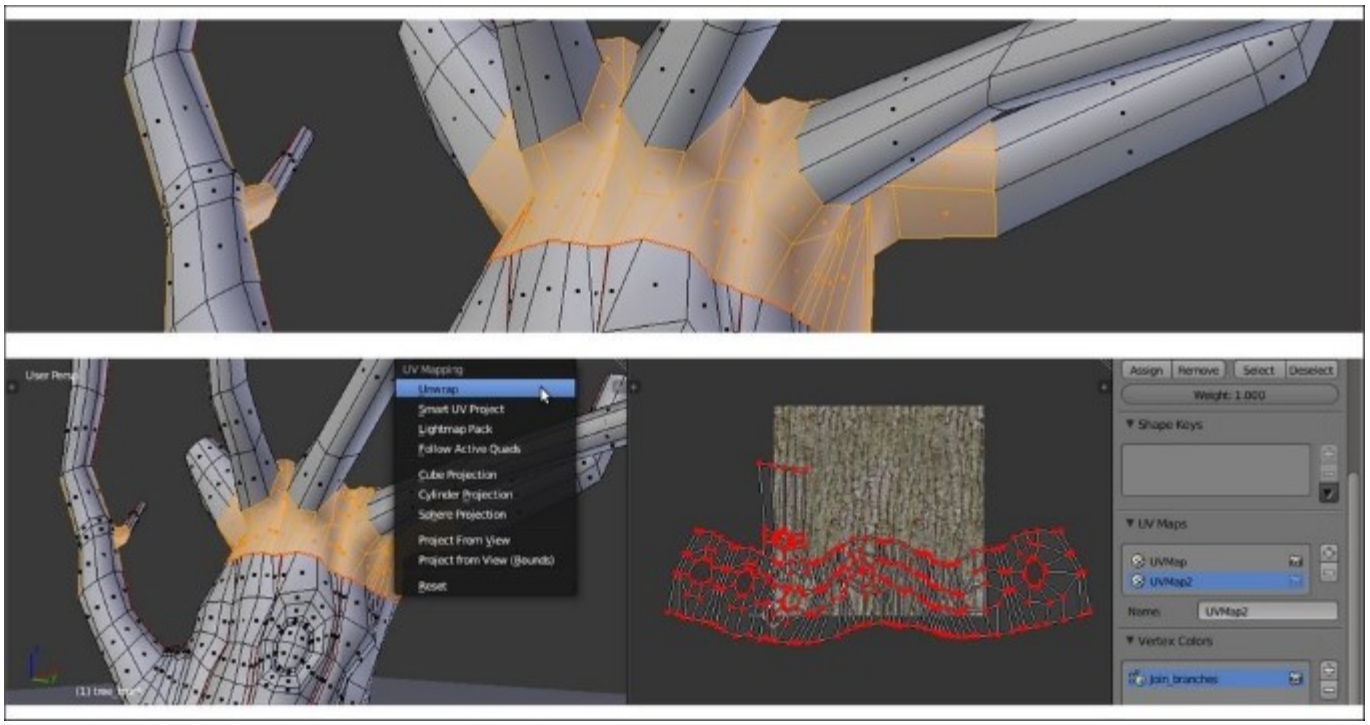
This is what we are going to do:

1. Select the **trunk** mesh and go into the **Vertex Paint** mode; the mesh turns totally white, because that is the color assigned to the vertexes by default. Start to paint with pure black on the vertexes located at the joining of the low branches with the trunk, achieving this result:



The trunk model seen in the Vertex Paint mode

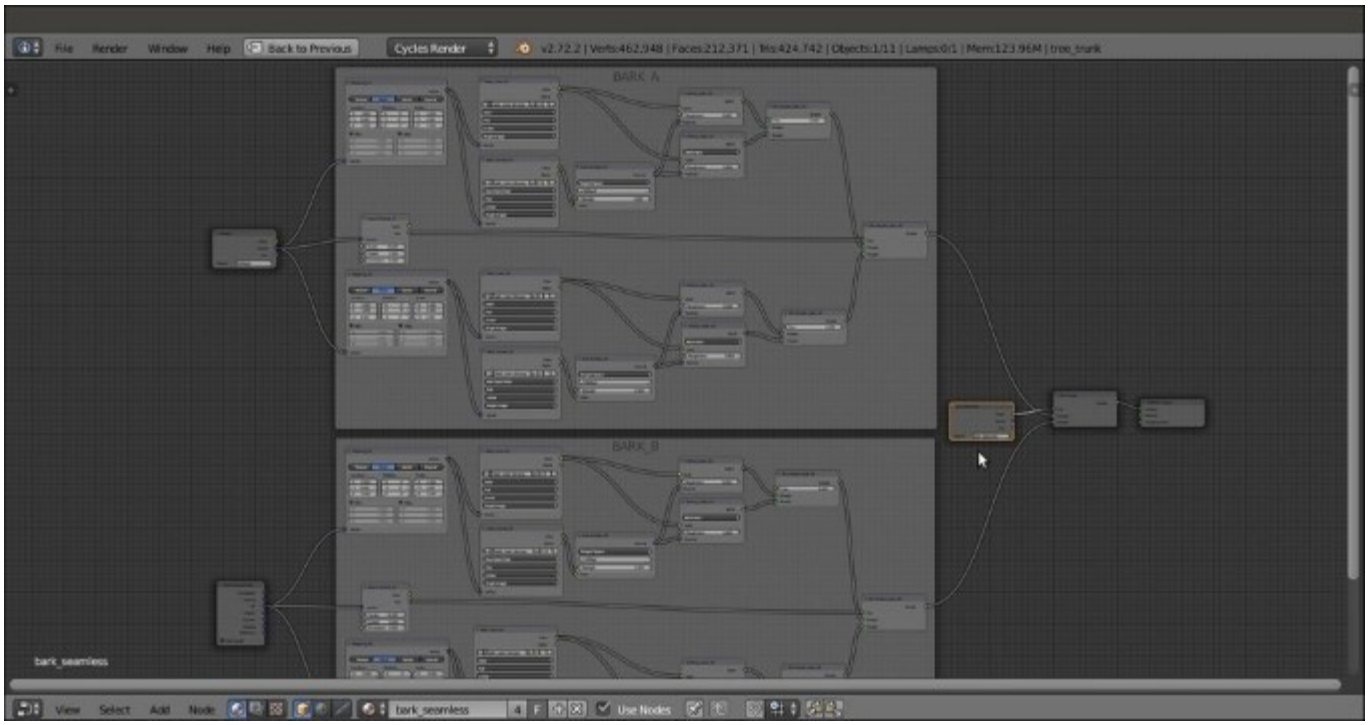
2. As you can see, the joining vertices edge loops are black but are smoothly blending into the white of the default mesh vertex color. This will be used as a stencil map to blend two different instances of the same bark material mapped on different UV coordinates. Go to the **Object data** window and rename the **Vertex Color** layer as `Join_branches`.
3. Switch to **Edit Mode** and select all the faces including the necessary vertices' edge loops; in the **Object data** window, under the **UV Maps** subpanel, click on the **+** icon (add **UV Map**) and rename the new UV coordinates layer as `UVMap2`. Place the mouse cursor on the 3D viewport, press **U**, and select **Unwrap** in the **UV Mapping** pop-up menu, as shown in the following screenshot:



The trunk model in Edit Mode and the UV islands in the UV/Image Editor window

4. Go out of **Edit Mode**. Click on the user number to the right of the material data block in the **Node Editor** window toolbar and rename the new material as `bark_seamless`.

   Now, by looking at the following screenshot, it is clear what we have to do:



Two identical bark materials mapped on different UV layers and mixed on the ground of the Vertex Paint output

5. Make a duplicate of the bark material and blend the two shaders (inside the **BARK_A** and **BARK_B** frames respectively) using a **Mix Shader** node, modulated by the **Join_branches** vertex color stencil. Use an **Attribute** node both for the **Vertex Color** layer output and to set the **UVMap2** coordinates layer for the copy of the bark material. Now, the output looks similar to what is shown in the following screenshot:

The final result: no more seams

As you can see in the preceding screenshot, there are no more visible seams; the two differently UV mapped materials smoothly blend together.

Creating tree shaders – the leaves

In this second tree recipe, we will create the leaves shaders, as shown in the following screenshot:

The leaves as they appear in the final rendering

Getting ready

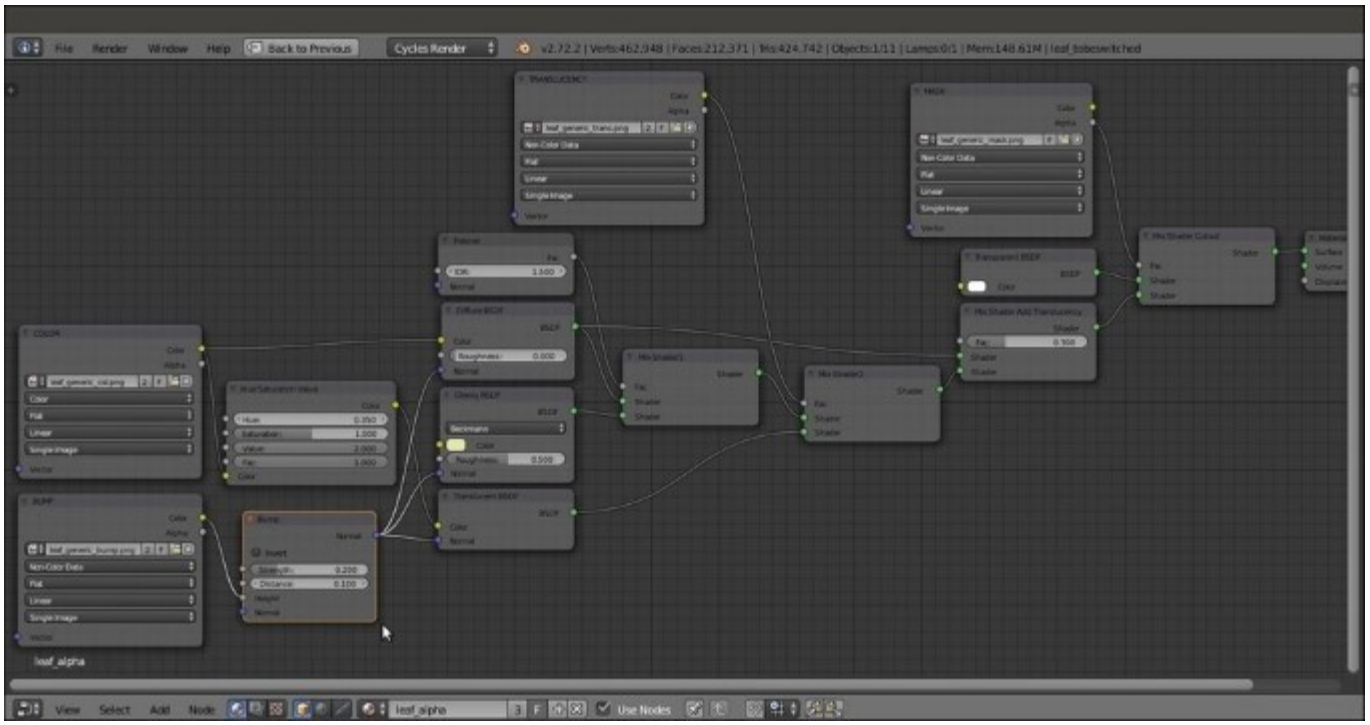
Carrying on with the blend file of the previous recipe, now, activate (hold *Shift* while clicking) the 2nd and the 11th scene layers, and in **Outliner**, select the **leaf_generic_mid** object.

How to do it...

Let's proceed with the creation of the leaves shaders:

1. Click on the **New** button in the **Node Editor** window toolbar or in the **Material** window, and rename the material as `leaf_alpha`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node and label it as **Mix Shader Cutout**; in the first **Shader** slot, select a **Transparent BSDF** shader node, and in the second one, select a new **Mix Shader** node, which will be labeled as **Mix Shader Add Translucency**.
3. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), label it as `MASK`, and connect its **Alpha** output to the **Fac** input socket of the **Mix Shader Cutout** node.

4. Click on the **Open** button of the **MASK** node, browse to the `textures` folder, and load the `leaf_generic_mask.png` image (which actually is a simple black leaf silhouette with a transparent alpha channel). Set **Color Space** to **Non-Color Data**.
5. Add a **Diffuse BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Diffuse BSDF**), a **Glossy BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**), and a **Translucent BSDF** node (press *Shift* + *A* and navigate to **Shader** | **Translucent BSDF**).
6. Add two new **Mix Shader** nodes (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), and label them as `Mix Shader1` and `Mix Shader2`.
7. Connect the output of the **Diffuse BSDF** shader to the first **Shader** input socket of the **Mix Shader1** node, and the output of the **Glossy BSDF** shader to the second **Shader** input socket. Set the **Glossy BSDF** node's **Distribution** to **Beckmann**, and change the **Color** values for **R** to `0.794`, **G** to `0.800`, and **B** to `0.413`, and the **Roughness** value to `0.500`.
8. Connect the output of the **Mix Shader1** node to the first **Shader** input socket of the **Mix Shader2** node, and the output of the **Translucent** node to the second one; connect the output of the **Mix Shader2** node to the second **Shader** input socket of the **Mix Shader Add Translucency** node.
9. Connect the output of the **Diffuse BSDF** shader node to the first **Shader** input socket of the **Mix Shader Add Translucency** node. Set its **Fac** value to `0.300` (this value establishes the amount of translucency in the shader).
10. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), label it as `TRANSLUCENCY`, and connect its **Color** output to the **Fac** input socket of the **Mix Shader2** node. Click on the **Open** button, browse to the usual `textures` folder, and load the `leaf_generic_trans.png` image. Set **Color Space** to **Non-Color Data**.
11. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**), connect it to the **Fac** input socket of the **Mix Shader1** node, and set **IOR** to `1.500`.
12. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), label it as `COLOR`, and connect its **Color** output to the **Color** input socket of the **Diffuse BSDF** shader node and to the **Color** input socket of the **Translucent BSDF** node. Click on the **Open** button, browse to the `textures` folder, and load the `leaf_generic_col.png` image.
13. Add a **Hue Saturation Value** node (press *Shift* + *A* and navigate to **Color** | **Hue/Saturation**) and paste it between the **COLOR** image texture node and the **Translucent BSDF** shader node. Set the **Hue** value to `0.350` and **Value** to `2.000`.
14. Add a last **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and label it as `BUMP`; add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the **BUMP** image node's **Color** output to the **Height** input socket of the **Bump** node, and the **Normal** output socket of this node to the **Normal** input sockets of the **Diffuse BSDF**, **Glossy BSDF**, and **Translucent BSDF** shader nodes.
15. Click on the **Open** button, browse to the `textures` folder, and load the `leaf_generic_bump.png` image. Set **Color Space** to **Non-Color Data** and the **Bump** node's **Strength** value to `0.200`, as shown in the following screenshot:



The leaf_alpha material network

How it works...

- From step 1 to 11, we built the basic shader of the leaf, using an image that has alpha channel data to cut out the leaf shape on the Plane and a gray-scale image to drive the translucency effect.
- From step 12 to 15, we added the color of the leaf, using it also with a hue and intensity variation for the translucency color, and then we added the bump.

There's more...

Now, assign the same material to both the `leaf_generic_low` and `leaf_generic_hi` meshes on the 11th layer.

The modeled leaf mesh doesn't need the alpha channel, so select the **leaf_generic_hi** object, and in the toolbar of the **Node Editor** window, click on **user data number** to make it **single-user**. Rename the new material as `leaf` and delete the **MASK** and **Transparent BSDF** nodes, and then press *Alt + D* to remove the **Mix Shader Cutout** node from the link and delete it as well.

Remember that the examples in the preceding and following images are made with very stylized models that come from the *Big Buck Bunny* short movie; real objects have more subtle details and more random repeating patterns, but in this case, this just depends on the image textures you are going to use for your material.

Such a shader is of good use not only for leaves, but also for other kinds of plants; in many cases, it's enough to give variations to the color.

Creating a layered human skin material in Cycles

In this recipe, we will create a layered skin material by using the open-content character Sintel.

Sintel is the main character of the third open movie of the same name produced by the Blender Foundation; the Sintel character and all the other movie assets are licensed under the Creative Commons Attribution 3.0 license (http://creativecommons.org/licenses/by/3.0/). The following screenshot is of Sintel's face:

Sintel's face in the final rendering

Getting ready

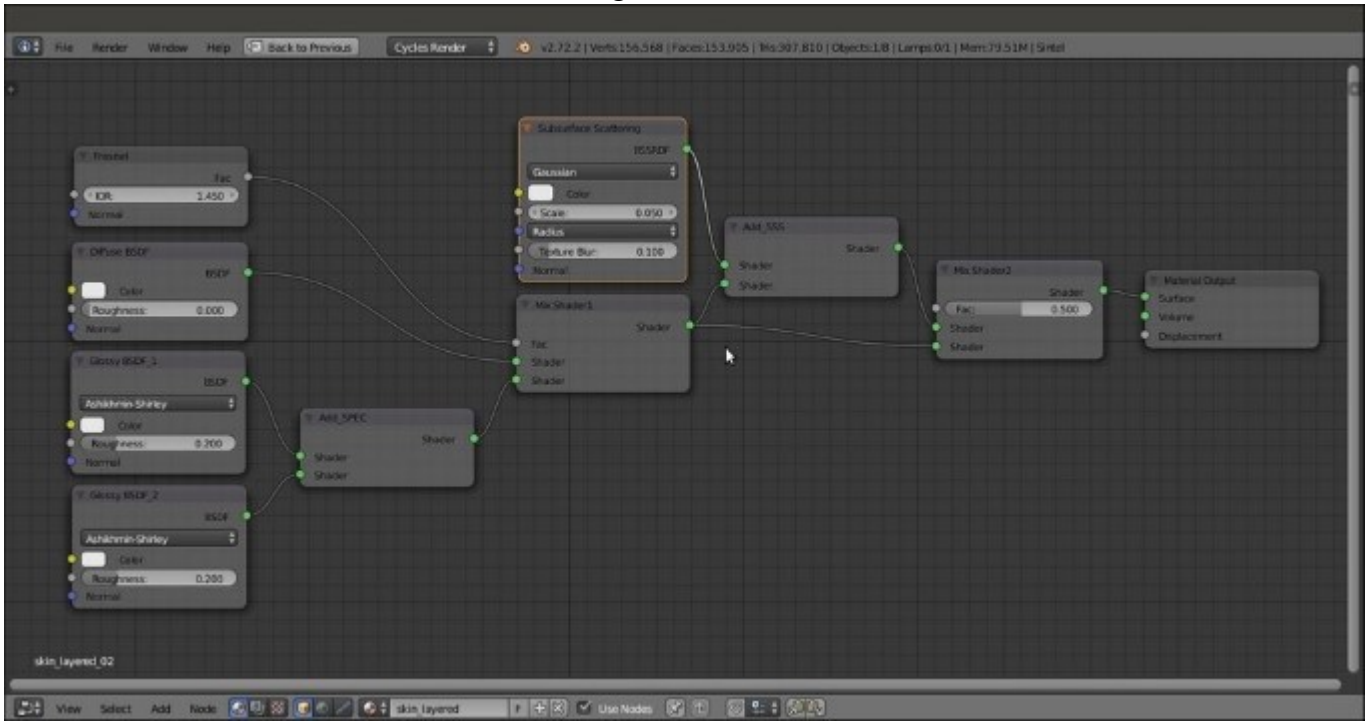
Start Blender and open the `9931OS_08_skin_start.blend` file, where there is an already set scene with the Sintel character standing on a Plane, a Sun lamp, and a Camera.

Except for Sintel's body skin, all the other mesh objects have either gesso-like materials or eyes already assigned.

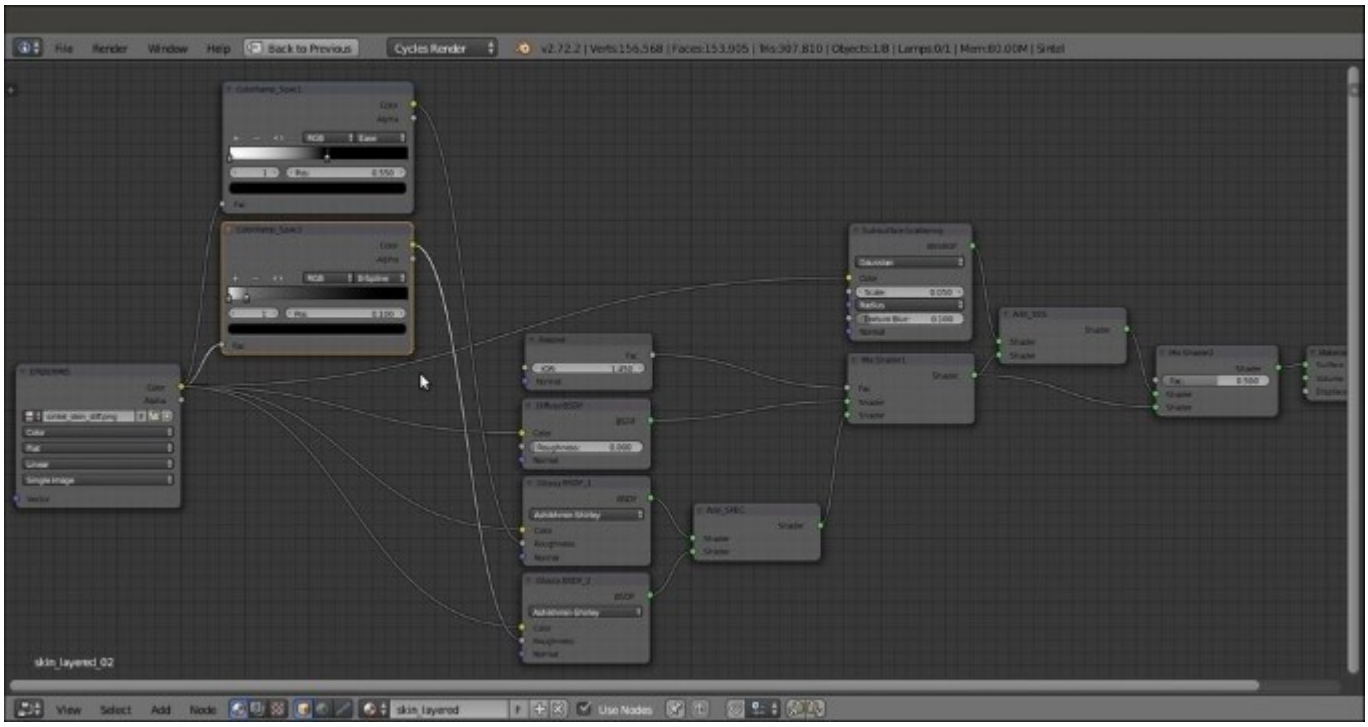
How to do it...

Let's start with the layered skin shader creation:

1. Be sure to have the **Sintel** object selected, and then click on the **New** button in the **Node Editor** window toolbar or in the **Material** window under the main **Properties** panel and rename the material as `skin_layered`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node; go to the **Active Node** panel to the right of the **Node Editor** window (if not present, put the mouse in the **Node Editor** window and press *N* to make it appear), and in the **Label** slot, rename the **Mix Shader** as `Mix Shader1`.
3. In the first **Shader** slot of this new **Mix Shader1** node, select a **Diffuse BSDF** shader node, and in the second one, select an **Add Shader** node; label this node as `Add SPEC`.
4. Add two **Glossy BSDF** shader nodes (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**) and label them as `Glossy BSDF_1` and `Glossy BSDF_2`. Set their **Distribution** to **Ashikhmin-Shirley**, and then connect their output to the first and second **Shader** input sockets of the **Add SPEC** node respectively.
5. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader1** node. Set **IOR** to `1.450`.
6. Press *Shift* + *D* to duplicate the **Mix Shader1** node, label the duplicate as **Mix Shader2**, and paste it between the **Mix Shader1** and the **Material Output** nodes.
7. Press *Shift* + *D* to duplicate the **Add SPEC** node, label the duplicate as `Add SSS`, and connect its output to the first **Shader** input socket of the **Mix Shader2** node, so that the connection that comes from the **Mix Shader1** node automatically switches to the second **Shader** input socket. Connect the output of the **Mix Shader1** node also to the second **Shader** input socket of the **Add SSS** node.
8. Add a **Subsurface Scattering** node (press *Shift* + *A* and navigate to **Shader** | **Subsurface Scattering**) and connect its output to the first **Shader** input socket of the **Add SSS** node. Set **Falloff** to **Gaussian**; **Scale** to `0.050`; **Radius** to `4.000`, `2.000`, and `1.000`; and the **Texture Blur** value to `0.100`, as shown in the following screenshot:



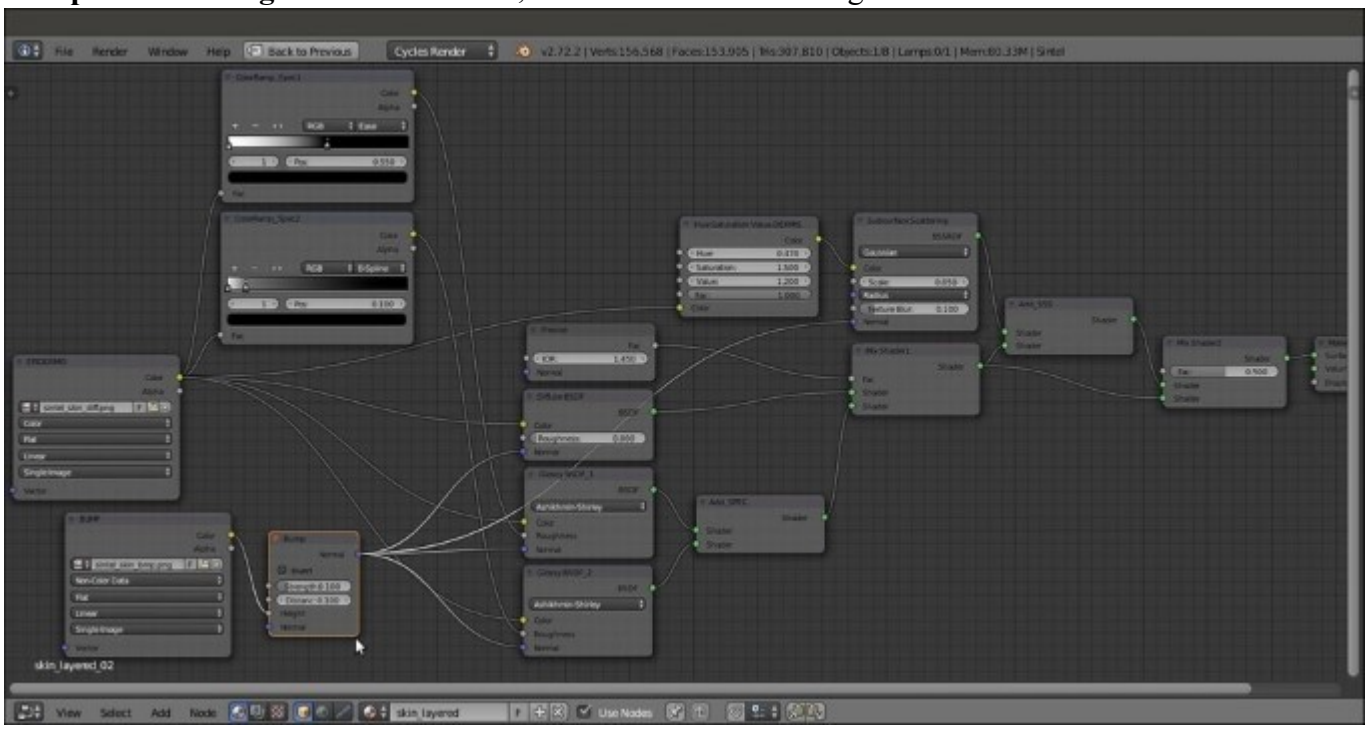
9. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and label it as `EPIDERMIS`; connect its **Color** output to the **Color** input sockets of the **Diffuse BSDF** and **Subsurface Scattering** nodes and the two **Glossy BSDF** nodes.
10. Click on the **Open** button of the **EPIDERMIS** image texture node, browse to the `textures` folder, and load the `sintel_skin_diff.png` image.
11. Add two **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and label them as `ColorRamp_Spec1` and `ColorRamp_Spec2`. Connect the **Color** output of the **EPIDERMIS** node also to the **Fac** input socket of both the **ColorRamp** nodes.
12. Connect the **Color** output of the **ColorRamp_spec1** node to the **Roughness** input socket of the **Glossy BSDF_1** shader node; set **Interpolation** to **Ease**, and move the black color stop to the `0.550` position and the white color stop to the `0.000` position.
13. Connect the **Color** output of the **ColorRamp_spec2** node to the **Roughness** input socket of the **Glossy BSDF_2** shader node; set **Interpolation** to **B-Spline**, and move the white color stop to the `0.100` position and the white color stop to the `0.000` position, as shown in the following screenshot:



Sintel's color map is directly connected to the shader nodes but is modulated through ColorRamp nodes for the roughness of the glossy nodes

14. Add a **Hue Saturation Value** node (press *Shift* + *A* and navigate to **Color** | **Hue/Saturation**), label it as `Hue Saturation Value DERMIS`, and paste it between the **EPIDERMIS** and **Subsurface Scattering** nodes. Set the **Hue** value to `0.470`, the **Saturation** value to `1.500`, and **Value** to `1.200`.

15. Add a new **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**); label this **Image Texture** node as `BUMP`, and connect its **Color** output to the **Height** input socket of the **Bump** node, and the **Normal** output of this node to the **Normal** input sockets of the **Diffuse BSDF** node, of the two **Glossy BSDF** nodes, and of the **Subsurface Scattering** nodes.
16. Click on the **Open** button of the **BUMP** image texture node, browse to the `textures` folder, and load the `sintel_skin_bmp.png` image. Set **Color Space** to **Non-Color Data** and the **Bump** node's **Strength** value to `0.100`, as shown in the following screenshot:



The same color map modified for the SSS node and the bump map connected as per the shader bump

How it works...

In this recipe, we used a layered approach to build the human skin shader, but what does layered mean exactly?

It means that the shader tries to simulate the behavior of real human skin in the most effective possible way. I'm referring to the fact that the human skin is composed of several different overlapping and semi-transparent layers that reflect and absorb light rays in various ways, giving the reddish coloration to certain areas due to the famous subsurface scattering effect.

Now, a perfect reproduction of the real human skin model is not necessary; usually, it's enough to use different image maps for the key components of the shader, each one added on top of the other: the base color, the dermis blood layer, the specularity map, and the bump map.

In our case, we had at our disposal only two image maps, the `sintel_skin_diff.png` color one and the `sintel_skin_bmp.png` gray-scale map, which we used straight for the bump; we could have obtained the missing maps with the aid of an image editor (such as, for example, GIMP), but for the sake of this exercise, to obtain the required missing images, we used the nodes: so, starting from the **EPIDERMIS** layer, that is the color map, we obtained via the **Hue Saturation Value DERMIS** node the blood-vessel layer that lies beneath the epidermis, as shown in the following screenshot:

The normal color map and the blood-vessel version rendered separately

By the use of the two **ColorRamp** nodes and the two gray-scale versions for the specularity component, one sharp specularity map and a softer one are shown in the following screenshot:

The two different glossy maps obtained from the same color map and rendered separately

Then, the `sintel_skin_bmp.png` map has been connected to the **Bump** node for the per-shader bump effect.

Note that because we used the color map to obtain all the others, certain areas of the images are wrong; for example, the eyebrows, shown in pure white on the specularity maps, should have been removed. In any case, this doesn't show that much on the final render, and the result is more than acceptable.

Creating fur and hair

Fur, in the world of computer graphics, is considered among the most difficult things to recreate, both because it's generally quite expensive from a memory management point of view (a single character can easily have millions of hair strands) and also because it can be quite a task to make a believable shader that can work under different light conditions.

Blender is not new to fur creation; the exact goal of the open movie *Big Buck Bunny* was to add tools for fur creation to the Blender Internal rendering engine, and it did it through a new type of primitive, strands, which have to be enabled in the **Particle** panel (the **Strand render** item); strands are very instanced on the particle system, but they can be edited, combed, and tweaked in several ways to obtain the best possible result.

Almost the same concept applies for the Cycles rendering engine; there's no need to enable the **Strand render** item anymore, because strands are rendered automatically by Cycles when the **Hair** item is selected as **Particle Type**.

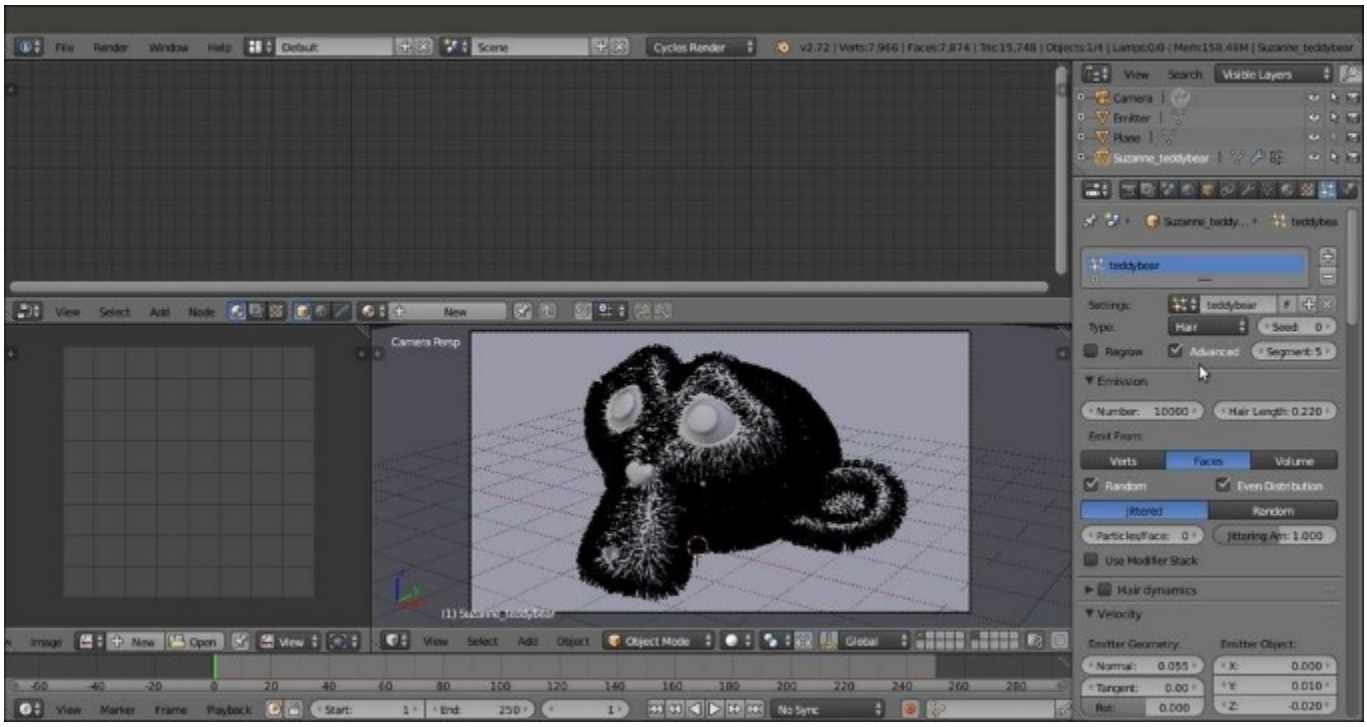
In fact, once the **Hair** item's **Particle Type** has been selected, you will find two more subpanels at the bottom of the **Particle** window: **Cycles Hair Rendering** and **Cycles Hair Settings**. Here is a screenshot of the teddy bear Suzanne in the **Rendered** view:

The Rendered teddy bear Suzanne

Getting ready

Start Blender and open the `9931OS_08_hair_start.blend` file; in the scene, there is a Suzanne primitive (**Suzanne_teddybear**) with a **Hair** particle system already named **teddybear** and set (go to see it in the **Particle** window) to resemble the fur of a cuddly toy.

The **Suzanne_teddybear** mesh is already unwrapped and has a **Vertex Group** named **density**, used in the **Particle** window (the **Vertex Groups** subpanel) to establish the **Density** distribution of the fur on the mesh (in short, to avoid fur on the eyes, the nose, and inside the mouth) as shown in the following screenshot:



A screenshot of the particle system as it appears in the Solid viewport shading mode and the settings to the left

How to do it...

We are going to add three different materials to the **Suzanne_teddybear** object: **base_stuff**, which is the basic material for the raw mesh, an eyes material, and the **teddybear** material for the fur, using the following steps:

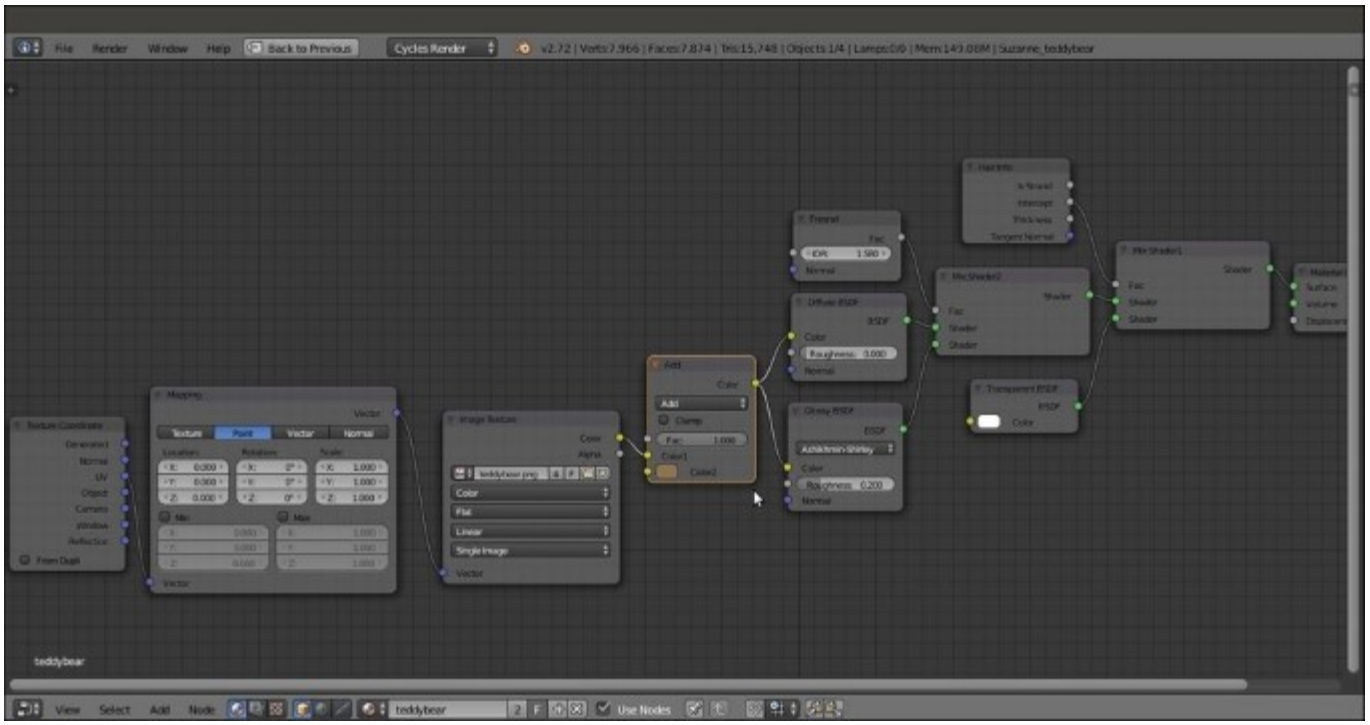
1. Select the **Suzanne** mesh and click on the **New** button in the **Node Editor** window toolbar or in the **Material** window to the right; rename the material as `base_stuff`.
2. Press *Tab* to go into **Edit Mode** and select the eyes vertices (put the mouse pointer over the interested part and press the *L* key to select all the linked vertices); click on the little + icon to the right of the **Material** window (add a new material slot) and add a new material. Click on the **New** button and rename the new material `eyes`, and then click on the **Assign** button. Press *Tab* to go out of **Edit Mode**.
3. Click again on the little + icon to the right of the **Material** window (add a new material slot) to add a third material (not to be assigned to any vertex or face; in fact, we are out of **Edit Mode**); click on the **New** button and rename the new material as `teddybear`.

4. Go to the **Particle** window at the top of the **Render** subpanel, and click on the **Material Slot** button (Material slot used to render particles), where at the moment, **Default Material** is selected instead of the `teddybear` material.
5. Go to the **Cycles Hair Rendering** subpanel to be sure that the **Primitive** item is set to **Curve Segments**, and set **Shape** to **Thick**; then, go to **Cycles Hair Settings** to be sure that the **Shape** value is `-0.50`, the **Root** value is `1.00`, the **Tip** value is `0.05`, the **Scaling** value is `0.01`, and the **Close Tip** item is checked.
6. Now, in the **Material** window, select the **base_stuff** material; in the **Node Editor** window, add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), a **Glossy BSDF** node, and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Texture Coordinate**; repeat similar steps to add other nodes).
7. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of this node to the **Vector** input socket of the **Image Texture** node. Paste the **Mix Shader** between the **Diffuse BSDF** and the **Material Output** nodes and connect the output of the **Glossy BSDF** shader to the second **Shader** input socket of the **Mix Shader** node.
8. Connect the **Color** output of the **Image Texture** node to the **Color** input socket of the **Diffuse BSDF** shader node and of the **Glossy BSDF** shader node; click on the **Open** button and browse to the `textures` folder to load the `teddybear.png` image (a simple color map painted directly in Blender). Set **Distribution** of the **Glossy BSDF** node to **Ashikhmin-Shirley**, the **Roughness** value to `0.300`, and the **Mix Shader** node's **Fac** value to `0.400`.
9. Back in the **Material** window, select the **eyes** material and switch the **Diffuse BSDF** shader with a **Mix Shader** node; in the first **Shader** slot, select a **Diffuse BSDF** shader, and in the second one, select a **Glossy BSDF** shader node.
10. Set the **Mix Shader** node's **Fac** value to `0.200`; change the **Diffuse BSDF** node's **Color** values for **R** to `0.010`, **G** to `0.003`, and **B** to `0.001`; and change the **Glossy BSDF** node's **Roughness** value to `0.100`.

    In the following screenshot, the **teddybear** particle system has been hidden by disabling the viewport's visibility in the **Object modifiers** window:

The base_stuff material in the Node Editor window and in the Preview

11. In the **Material** window, select the **teddybear** material and switch the **Diffuse BSDF** node with a **Mix Shader** node, and label it as `Mix Shader1`; in the first **Shader** slot, select another **Mix Shader** node, and in the second one, select a **Transparent BSDF** node.
12. Label the second **Mix Shader** node as `Mix Shader2`; then, in the first **Shader** slot, select a **Diffuse BSDF** shader node, and in the second one, select a **Glossy BSDF** shader node. Set the **Glossy BSDF** node's **Distribution** to **Ashikhmin-Shirley**, and its **Roughness** to `0.200`.
13. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader2** node; set **IOR** to `1.580`. Add a **Hair Info** node (press *Shift* + *A* and navigate to **Input** | **Hair Info**), and connect the **Intercept** output to the **Fac** input socket of the **Mix Shader1** node.
14. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**), and connect its **Color** output to the **Color** input socket of the **Diffuse BSDF** node; click on the little arrows to the left of the **Open** button to select the already loaded `teddybear.png` image map.
15. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set **Blend Type** to **Add** and the **Fac** value to `1.000`, and paste it between the **Image Texture** node and the **Diffuse shader** node. Set the **Color2** values for **R** to `0.277`, **G** to `0.179`, and **B** to `0.084` and then connect its output also to the **Color** input socket of the **Glossy BSDF** shader node.
16. Optionally, add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and a **Mapping** node (press *Shift* + *A* and navigate to **Vector** | **Mapping**), and connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node and the output of the latter to the **Vector** input socket of the **Image Texture** node, as shown in the following screenshot:



The teddy bear material network; note the transparency driven by the Intercept output of the Hair Info node

How it works...

From step 1 to 3, we prepared the three materials to be used; we went into **Edit Mode** to assign the second material, **eyes**, to the eyes vertices of the mesh, and then we went back in **Object Mode** to add a third material that doesn't need to be assigned to any face of the mesh because they are only to be used for hair rendering.

In steps 4 and 5, we made sure that the right particle system settings for the material are to be rendered as fur.

From step 6 to 8, we built the `base_stuff` material, a simple basic shader made by the Diffuse and Glossy components mixed by the **Mix Shader** node and colored by the UV mapped `teddybear` image texture; note that the texture we used in this first material is also used to give the right color to the hair; it is useful to have it also on the underlying mesh, to cover any hole or missing part in the particle system.

In steps 9 and 10, we built the `eyes` shader, which is again a very basic material made of the dark Diffuse color and the light gray Glossy components simply mixed by the **Mix Shader** node.

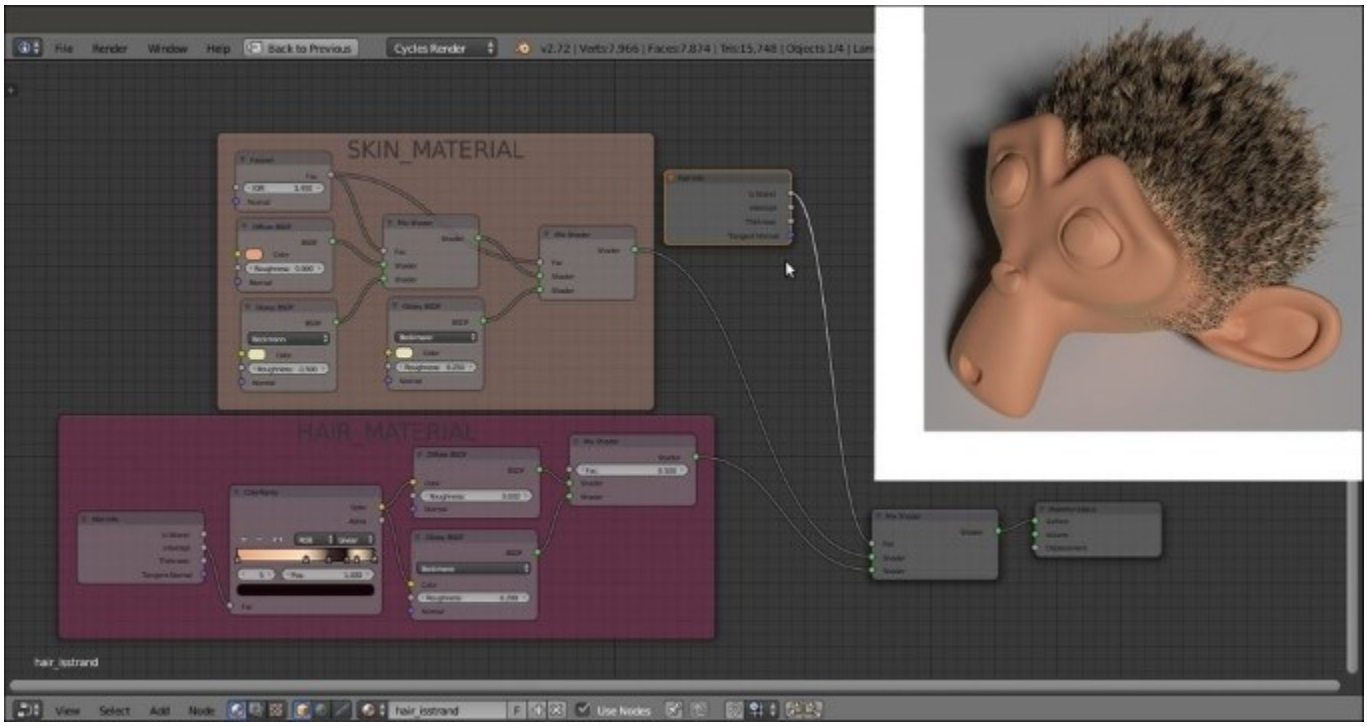
From step 11 to 16, we built the shader to be used by the particle system for the fur, mixing the already used `teddybear.png` image map, mapped on the UV coordinates, with a **MixRGB** node brownish

color outputted to the usual **Diffuse/Glossy** basic shader; note that the **Diffuse/Glossy** shader is then mixed with the **Transparent BSDF** shader by the **Intercept** value of the **Hair Info** node along the length of each hair strand.

There's more...

The `teddybear.png` image texture has been used both in the `base_stuff` and in the **teddybear** materials; this is often not necessary, because in Blender, the particle system hairs get the textures from the surface they are emitted from, so it would have been enough to use the `base_stuff` material also for the fur (by selecting it in the **Material Slot** under the **Render** subpanel in the **Particle** window, because we had more than one material on the **Suzanne** mesh); we had to make a new and different material because we wanted to add a **MixRGB** brownish color to the UV-mapped image and we had to make the shader fade and become transparent towards the strands' tips.

Note that in the **Hair Info** node, there is also the Boolean **Is Strand** output that, similar to the outputs of the **Light Path** node (**Is Camera Ray**, **Is Shadow Ray**, and so on) can be used alternatively to the **Material** button in the **Particle** window to assign a material value of **0** to the emitter mesh and a material value of 1 to the fur strands (`9931OS_08_hair_isstrand.blend`) as shown in the following screenshot:



The set up for the hair_isstrand material and the rendered result

This also means that obviously we can also use different image textures to obtain fur materials different from the material of the particle emitter: for example, in the following screenshot, the `tiger.png` image texture has been used only for the fur, whereas the `base_stuff` material still uses the

`teddybear.png` texture (and, honestly, this is blatantly visible... better to use the same image both for fur and emitter):

The rendered Suzanne_tiger object

The **Suzanne_tiger** object also has two different particle systems to create the fur, **tigerfur_long** and **tigerfur_short**, and three **Vertex Groups** to modulate the fur appearance, **density_long**, **density_short**, and **length**.

To take a look at the **Suzanne_tiger** object, open the `9931OS_08_tiger.blend` file.

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Hair_Rendering
- http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes/More
- http://blenderdiplom.com/en/tutorials/all-tutorials/536-tutorial-fur-with-cycles-and-particle-hair.html
- http://cgcookie.com/blender/2014/04/24/using-cycles-hair-bsdf-node/

Creating a gray alien skin material with procedurals

In this recipe, we will create a gray alien-like skin shader as shown in the following screenshot, using Cycles procedural textures:

The alien Suzanne as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_08_alienskin_start.blend` file, where there is an already set scene with an unwrapped **Suzanne** primitive object.

The **Suzanne_unwrapped_alien** mesh has been modified by a shape key, to morph its monkey features into the head of a gray alien–like creature; in fact, in the **Object data** window, under the **Shape Keys** subpanel, there are the alien shape keys, with **Value** of `1.000`; sliding the slider towards `0.000` gradually restores the original Suzanne shape.

The **Suzanne** mesh also has a **Vertex Colors** layer named **Col** and is obtained by the **Dirty Vertex Colors** tool.

On the second layer, there are two Planes tracked (by a Damped Track constraint, in the **Object Constraints** window) to the Camera to stay perpendicular to its point of view; the **star_backdrop** object is used to create a simple star backdrop for our alien Suzanne, and the **star_backdrop.001** object is used simply to create something to be reflected by the alien-like Suzanne's eyes.

Press *Shift* + *F1* (or go through the **File** | **Append** main menu) to append the **SSS_group** node from the `9931OS_07_SSS_ngroup.blend` file.

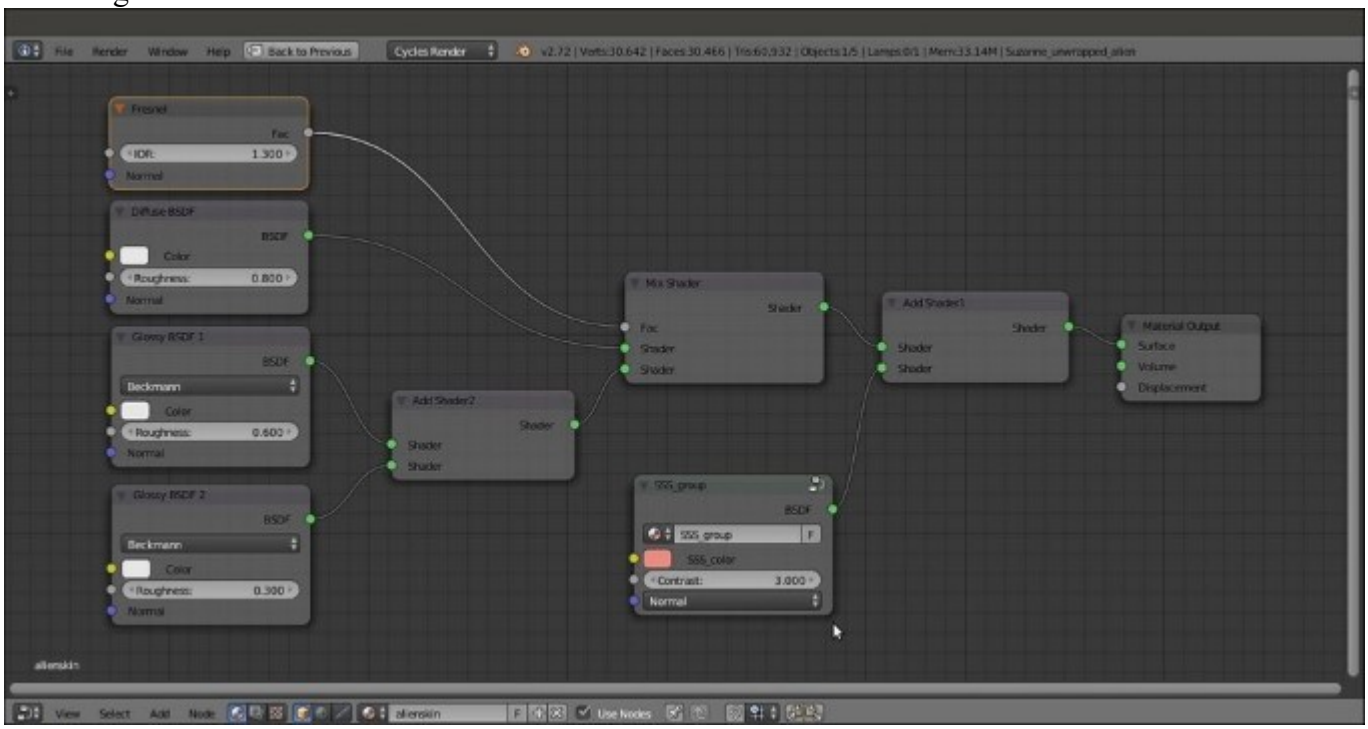
How to do it...

Let's start by first setting the background image material using the following steps:

1. Select the **star_backdrop** Plane and click on the **New** button in the **Node Editor** window toolbar or in the **Material** window. Rename the material `star_backdrop`, and in the **Material** window, switch the **Diffuse BSDF** shader with an **Emission** shader. Set the **Strength** to `0.500`.
2. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and connect the **Color** output to the **Color** input socket of the **Emission** node. Click on the **Open** button and browse to the `textures` folder and load the `centre-of-milky-way_tile_low.png` image.
3. Add an **RGB Curves** node (press *Shift* + *A* and navigate to **Color** | **RGB Curves**) and paste it between the **Image Texture** and **Emission** nodes. Click on the **curve** window to add a control point and set these coordinates: **X** to `0.36667` and **Y** to `0.12778`. Click again to add a second control point and set these coordinates: **X** to `0.65556` and **Y** to `0.81111`.
4. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Emission** and **Material Output** nodes; switch the connection that comes from the **Emission** node to the second **Shader** input socket of the **Mix Shader** node; leave the first **Shader** input socket empty.
5. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**) and connect the **Is Camera Ray** output to the **Fac** input socket of the **Mix Shader** node.
6. Go to **Outliner** and select the **star_backdrop.001** object; click on the double arrows icon to the left side of the data block name (*Browse Material to be linked*) in the **Node Editor** toolbar, and select the `star_backdrop` material; click on the **2** icon button to make it single-user.
7. In the new **star_backdrop.001** material, select the **Mix Shader** node and press *Ctrl* + *X* to delete it and keep the connection; then, also delete the **Light Path** node.
8. Go to the **Object data** window, and in the **Ray Visibility** subpanel, uncheck all the items except for the **Glossy** one.

   Now, let's get started with the creation of the alien skin shader (also with a different material for the eyes).
9. Select the **Suzanne_unwrapped_alien** object; click on the **New** button in the **Node Editor** window toolbar or in the **Material** window, and rename the material `alienskin`.
10. Go into **Edit Mode**, select the eyes vertices, and click on the **+** icon on the right of the **Material** window to add a second material. Click on the **New** button and rename the material as `alieneyes`; then, click on the **Assign** button to assign it to the selected vertices. Go out of **Edit Mode**.
11. Switch the **Diffuse BSDF** shader with a **Mix Shader** node; in the first **Shader** slot, select a **Diffuse BSDF** shader, and in the second one, select a **Glossy BSDF** shader. Set the **Mix Shader** node's **Fac** value to `0.600` and the **Diffuse BSDF** node's **Color** values for **R** to `0.010`, **G** to `0.006`, and **B** o `0.010`; set the **Glossy BSDF** node's **Distribution** to **Beckmann**, change the **Color** values for **R** to `0.345`, **G** to `0.731`, and **B** to `0.800`, and change the **Roughness** value to `0.100`.

12. Select the **alienskin** material, and in the **Material** window, switch the **Diffuse BSDF** shader with an **Add Shader** node (label it as `Add Shader1`); in the first **Shader** slot, select a **Mix Shader** node, and in the second one, load the appended **SSS_group** node. In this node, set the **Contrast** value to `3.000` and change the **Color** values for **R** to `0.834`, **G** to `0.263`, and **B** to `0.223`.
13. Go to the **Mix Shader** node, and in the first **Shader** slot, select a **Diffuse BSDF** node, and in the second one, select a new **Add Shader** node (label it as `Add Shader2`). Set the **Diffuse BSDF** node's **Roughness** value to `0.800`.
14. In both the **Shader** slots of the **Add Shader2** node, select a **Glossy BSDF** shader node; label the first one as **Glossy BSDF 1** and the second as **Glossy BSDF 2**; set **Distribution** of both to **Beckmann**, and then set the **Roughness** value of the first one to `0.600` and that of the second one to `0.300`.
15. Add a **Fresnel** node (press *Shift* + *A* and navigate to **Input** | **Fresnel**) and connect its output to the **Fac** input socket of the **Mix Shader** node. Set the **IOR** value to `1.300`, as shown in the following screenshot:



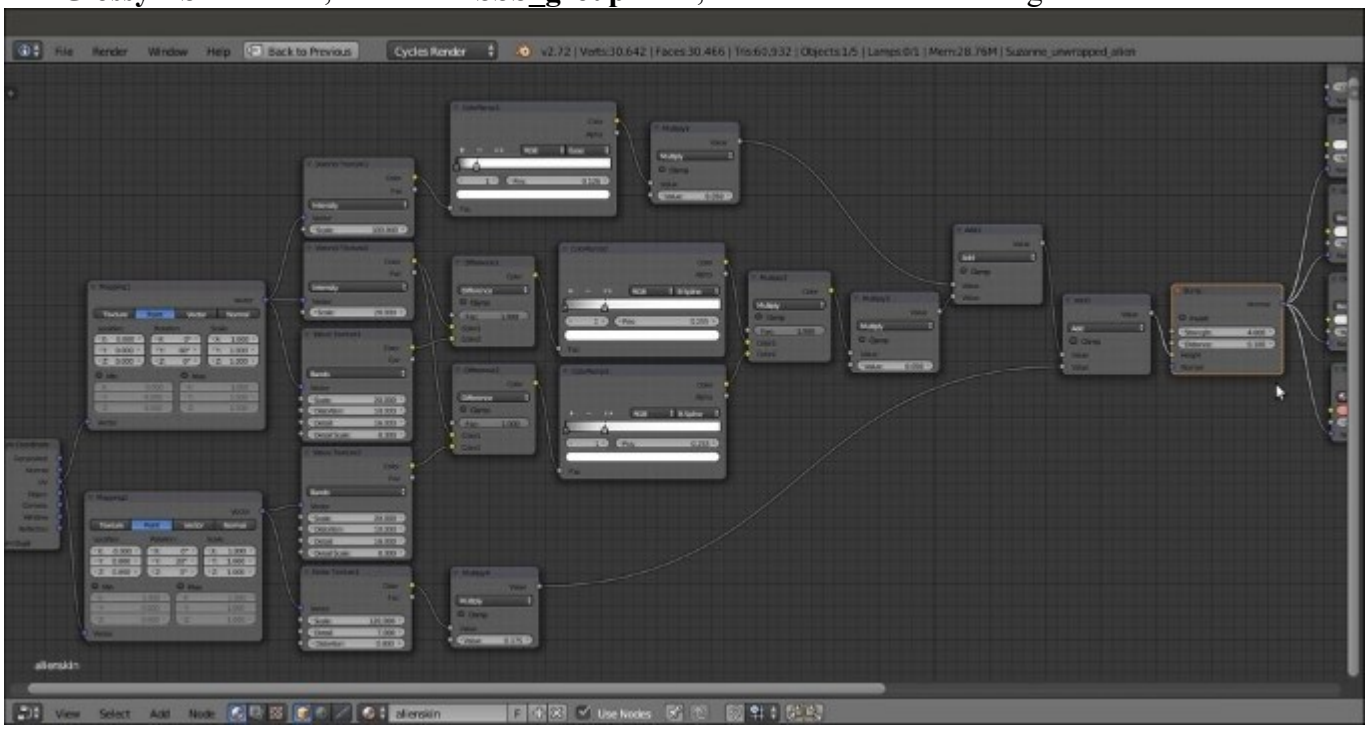
The basic shader network with the SSS provided by the appended node group

Now that we have set the basic shader for the alien skin, let's set an important component of the material, that is, the bump, by following these steps:

16. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) and two **Mapping** nodes (press *Shift* + *A* and navigate to **Vector** | **Mapping**); connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of both the **Mapping** nodes, labeled as `Mapping1` and `Mapping2` respectively.

17. Set the **Rotation** value of **Y** of the **Mapping1** node to `60°`; set the **Rotation** value of **Y** of the **Mapping2** node to `20°`.
18. Add two **Voronoi Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and label them as `Voronoi Texture1` and `Voronoi Texture2`. Connect the **Mapping1** node output to both their **Vector** input sockets. Set the **Scale** value of the **Voronoi Texture1** to `100.000` and the **Scale** value of the **Voronoi Texture2** to `20.000`.
19. Add two **Wave Texture** nodes (press *Shift* + *A* and navigate to **Texture** | **Wave Texture**) and label them as `Wave Texture1` and `Wave Texture2`. Connect the **Mapping1** node output to the **Vector** input socket of the **Wave Texture1** node. Set the texture **Scale** value to `20.000`, **Distortion** to `10.000`, **Detail** to `16.000`, and **Detail Scale** to `0.300`.
20. Connect the **Mapping2** node output to the **Wave Texture2** node's **Vector** input socket, and set all the texture values exactly as in the previously described one.
21. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and label it as `Noise Texture1`. Connect the **Mapping2** node output to its **Vector** input socket, and set the texture **Scale** value to `120.000` and the **Detail** value to `7.000`.
22. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp1`, and connect the **Voronoi Texture1** node's **Color** output to its **Fac** input socket. Set **Interpolation** to **Ease** and move the white color stop to the `0.126` position.
23. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**), change **Operation** to **Multiply**, and label it as `Multiply1`. Connect the **Color** output of the **ColorRamp1** node to the first **Value** input socket of the **Multiply1** node, and set the second **Value** to `0.050`.
24. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect the **Color** output of the **Voronoi Texture2** node to the **Color1** input socket, and the **Color** output of the **Wave Texture1** node to the **Color2** input socket. Set **Blend Type** to **Difference** and the **Fac** value to `1.000`, and then label it as `Difference1`.
25. Add a second **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect the **Color** output of the **Voronoi Texture2** node to the **Color1** input socket, and the **Color** output of the **Wave Texture2** node to the **Color2** input socket. Again, set the **Blend Type** to **Difference** and the **Fac** value to `1.000` and label it as `Difference2`.
26. Add two **ColorRamp** nodes (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), and label them as `ColorRamp2` and `ColorRamp3`; connect the **Difference1** node output to the **Fac** input socket of the **ColorRamp2** node, and the output of the **Difference2** node to the **Fac** input socket of the **ColorRamp3** node.
27. Set **Interpolation** to **B-Spline** for both of them, and for both of them, move the white color stop to the `0.255` position.
28. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect the **Color** output of the **ColorRamp2** node to the **Color1** input socket and the **Color** output of the **ColorRamp3** node to the **Color2** input socket. Set the **Blend Type** to **Multiply** and the **Fac** value to `1.000`. Label it as `Multiply2`.
29. Press *Shift* + *D* to duplicate the **Multiply1** node and label the duplicate as `Multiply3`. Connect the output of the **Multiply2** node to the first **Value** input socket of the **Multiply3** node. Set the second **Value** to `0.050`.
30. Press *Shift* + *D* to duplicate the **Multiply3** node and label the duplicate as `Multiply4`. Connect the **Color** output of the **Noise Texture** node to the first **Value** input socket of the **Multiply4** node, and set the second **Value** to `0.175`.

31. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and label it as `Add1`. Connect the output of the **Multiply1** node to the first **Value** input socket and the output of the **Multiply3** node to the second **Value** input socket.
32. Press *Shift* + *D* to duplicate the **Add1** node and label the duplicate as `Add2`; connect the output of the **Add1** node to the first **Value** input socket and the output of the **Multiply4** node to the second **Value** input socket.
33. Add a **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**) and connect the output of the **Add2** node to the **Height** input socket of the **Bump** node; set its **Strength** value to `4.000` and connect its **Normal** output to the **Normal** input sockets of the **Diffuse BSDF** node, of the two **Glossy BSDF** nodes, and of the **SSS_group** node, as shown in the following screenshot:

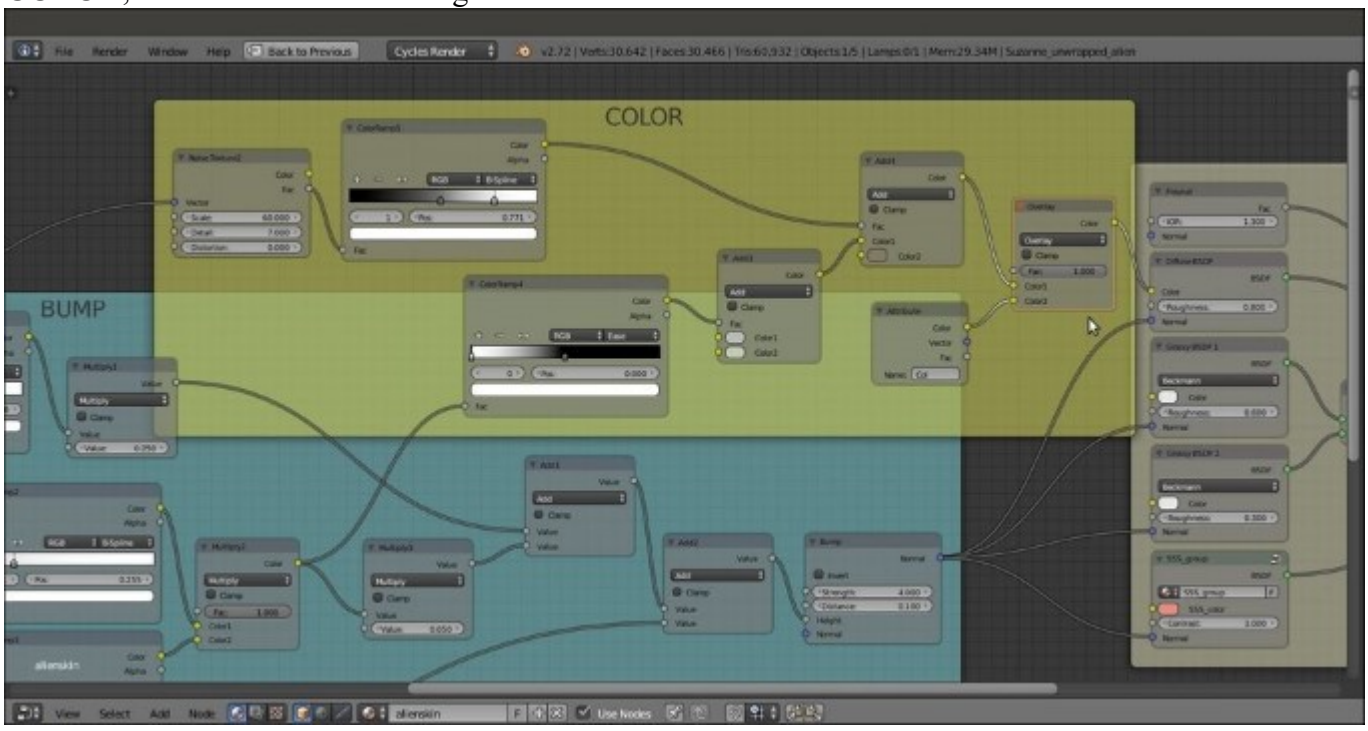


The apparently complex bump network to be connected as per the shader bump

We are done with the bump part, so now, let's set the color pattern using the following steps:

34. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp4`, and connect the **Color** output of the **Multiply2** node to its **Fac** input socket. Set **Interpolation** to **Ease** and move the black color stop to the `0.495` position, and the white color stop to the `0.000` position.
35. Add a **MixRGB** node (press *Shift* + *A* and navigate to **Color** | **MixRGB**), set the **Blend Type** to **Add**, and label it as `Add3`. Connect the **Color** output of the **ColorRamp4** node to the **Fac** input socket and set the **Color2** values for **R** to `0.553`, **G** to `0.599`, and **B** to `0.473`.
36. Add a **Noise Texture** node (press *Shift* + *A* and navigate to **Texture** | **Noise Texture**) and label it as `Noise Texture2`. Connect the **Mapping1** node output to its **Vector** input socket and set the texture's **Scale** value to `60.000` and the **Detail** value to `7.000`.

37. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**), label it as `ColorRamp5`, and connect the **Noise Texture2** color output to its **Fac** input socket. Set **Interpolation** to **B-Spline** and move the black color stop to the `0.486` position and the white color stop to the `0.771` position.
38. Press *Shift* + *D* to duplicate the **Add3** node, label the duplicate as `Add4`, and connect the **Color** output of the **ColorRamp5** node to its **Fac** input socket; connect the output of the **Add3** node to the **Color1** input socket and set the **Color2** values for **R** to `0.235`, **G** to `0.198`, and **B** to `0.132`.
39. Press *Shift* + *D* to duplicate the **Add4** node, label the duplicate as `Overlay`, and change **Blend Type** to **Overlay** as well. Set the **Fac** value to `1.000` and connect the output of the **Add4** node to the **Color1** input socket.
40. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**) and connect its **Color** output to the **Color2** input socket of the **Overlay** node; in the **Name** field, write `Col` (the **Vertex Colors** layer).
41. Connect the output of the **Overlay** node to the **Color** input socket of the **Diffuse BSDF** shader node.
42. To make them more easily readable, frame the three groups of nodes, **SHADER**, **BUMP**, and **COLOR**, as shown in the following screenshot:



The simpler color pattern to be connected only to the Diffuse shader node and the nodes grouped by frames

43. Save the file as `9931OS_08_alienskin_final.blend`.

How it works...

- From step 1 to 8, we built a simple and quick shader for the starry background and for the Plane in front of the **Suzanne_unwrapped_alien** mesh to be reflected in the eyes; note that the background Plane material works as a shadeless material. To see how to set a bright but not emitting light background, that is, a shader that behaves as the shadeless material you have in the Blender Internal engine, go to Chapter 9, *Special Materials*.
- From step 9 to 15, we built the basic shader for the alien Suzanne's skin; the diffuse component is mixed and modulated by the **Fresnel** output, with a specular component made by two summed **Glossy BSDF** nodes with different roughness values, so as to have a crisper specular effect on a more diffuse one, and with the additional help of the **SSS_group** node to simulate a reddish *Subsurface Scattering* effect.
- From step 16 to 33, we built the quite complex bump pattern for the skin by mixing the outputs of three different types of procedural textures with the help of both the **Math** and **MixRGB** nodes, and the variations provided by the **ColorRamp** nodes.
- Finally, from step 34 to 41, a simple grayish color pattern is modulated through the **Dirty Vertex Colors** layer data that comes from the mesh and that uses the first part of the bump pattern to add variation.

Chapter 9. Special Materials

In this chapter, we will cover the following recipes:

- Using Cycles volume materials
- Creating a cloud volumetric material
- Creating a "fire and smoke" shader
- Creating a shadeless material in Cycles
- Creating a fake immersion effect material
- Creating a fake volume light material

Introduction

In this final chapter, we are going to see some special materials, that is, materials that can be used for special effects or for situations where very realistic results are not required, for example, creation of volumetric effects (fire, smoke, mist, volumetric light, and so on) and special materials to obtain peculiar results (shadeless images, alpha backgrounds, and so on).

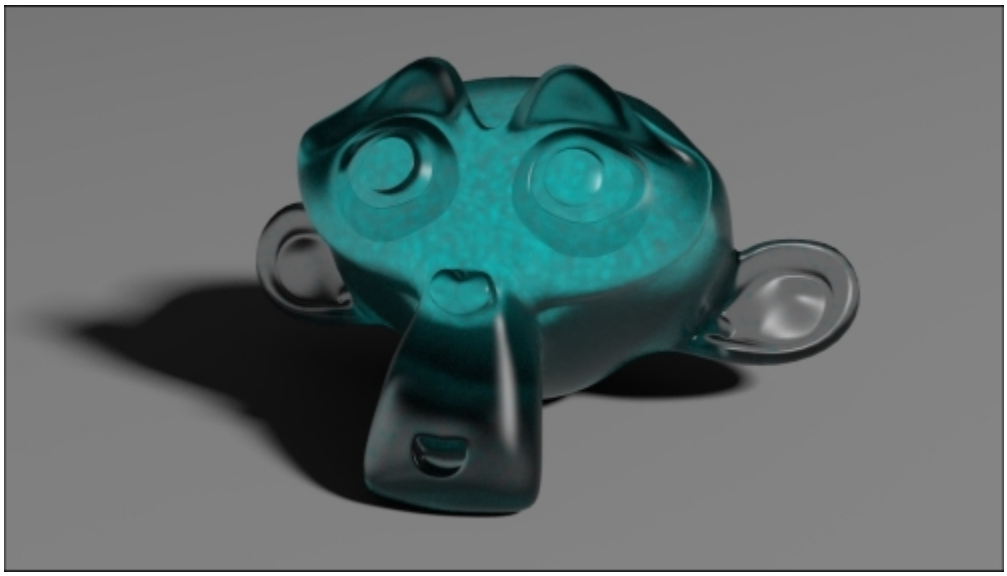
Using Cycles volume materials

In all the recipes we have seen so far, Cycles used the **Surface** input socket and (very rarely) the **Displacement** input socket for the bump effects of the **Material Output** node to make the renderings. Assigning colors or textures to the surface of an object clearly means that interaction between a ray of light and an object happens only at the surface level of the object, and until this surface doesn't show what should be inside, that's OK. The surface attribute is enough for a realistic rendering.

Things get more complex when there is a need to show what's inside an object, for example, water inside a glass container, smoke and clouds in a thick atmosphere, and so on.

Usually, these are effects that require the use of the volume attribute more than the surface attribute to be effectively rendered.

So, in the first recipes of this chapter, we are going to see the use of the **Volume** input socket of the **Material Output** node. Rather than covering a specific material, this recipe is more of a "tour" to show the possibilities related to the **Volume** shader assigned to a mesh object. Have a look at the following screenshot:



A glass Suzanne containing some kind of liquid

Getting ready

Start Blender and open the `9931OS_09_start.blend` file, with the usual Suzanne object leaning on a Plane, a mesh-light Emitter, and the Camera.

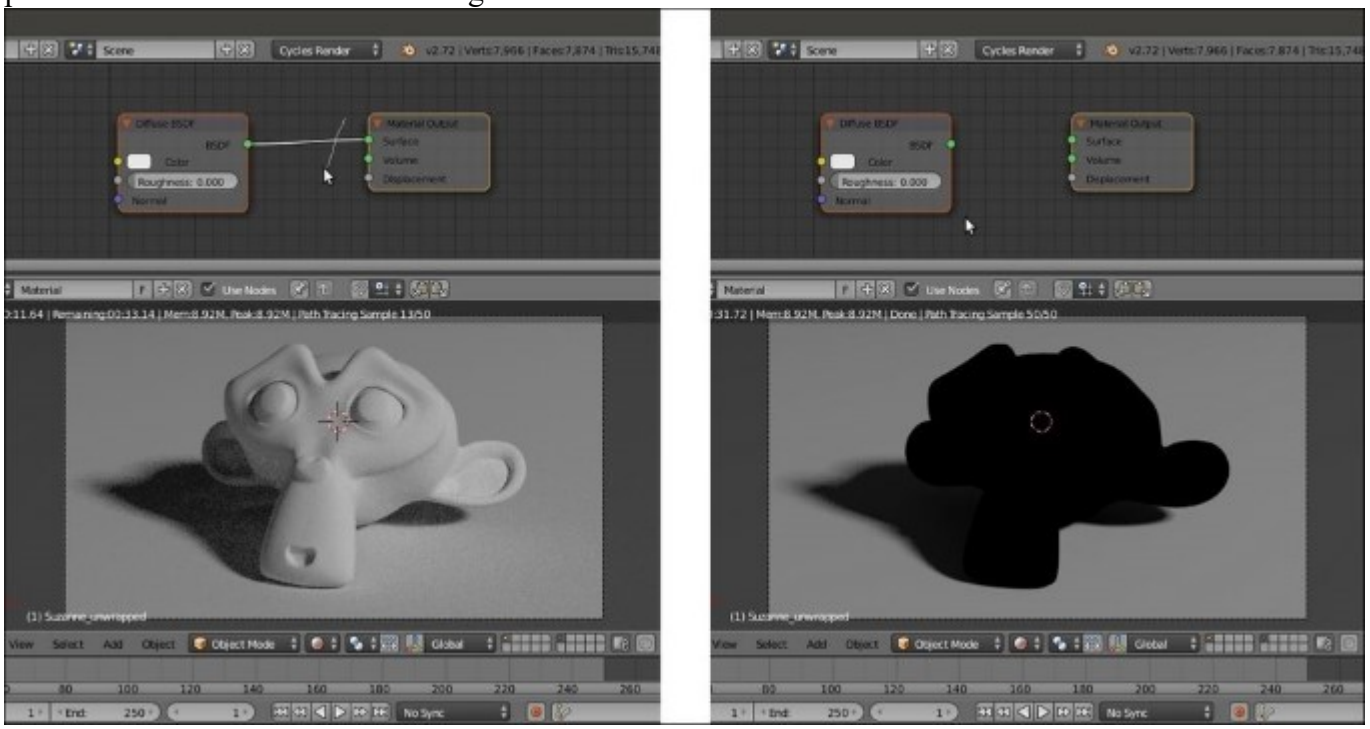
1. Go to the **Render** window, and under the **Sampling** subpanel, set **Samples** for **Preview** to `50` and for **Render** to `100`. Switch **Pattern** from **Sobol** to **Correlated Multy-Jitter**.

2. Still in the **Render** window, go to the **Volume Sampling** subpanel, and under the **Heterogeneous** item, set the **Step Size** value to `0.25`. The default value is `0.10`. Increasing this will make the rendering of volumes less accurate but faster, and lowering it will result in the opposite.

How to do it...

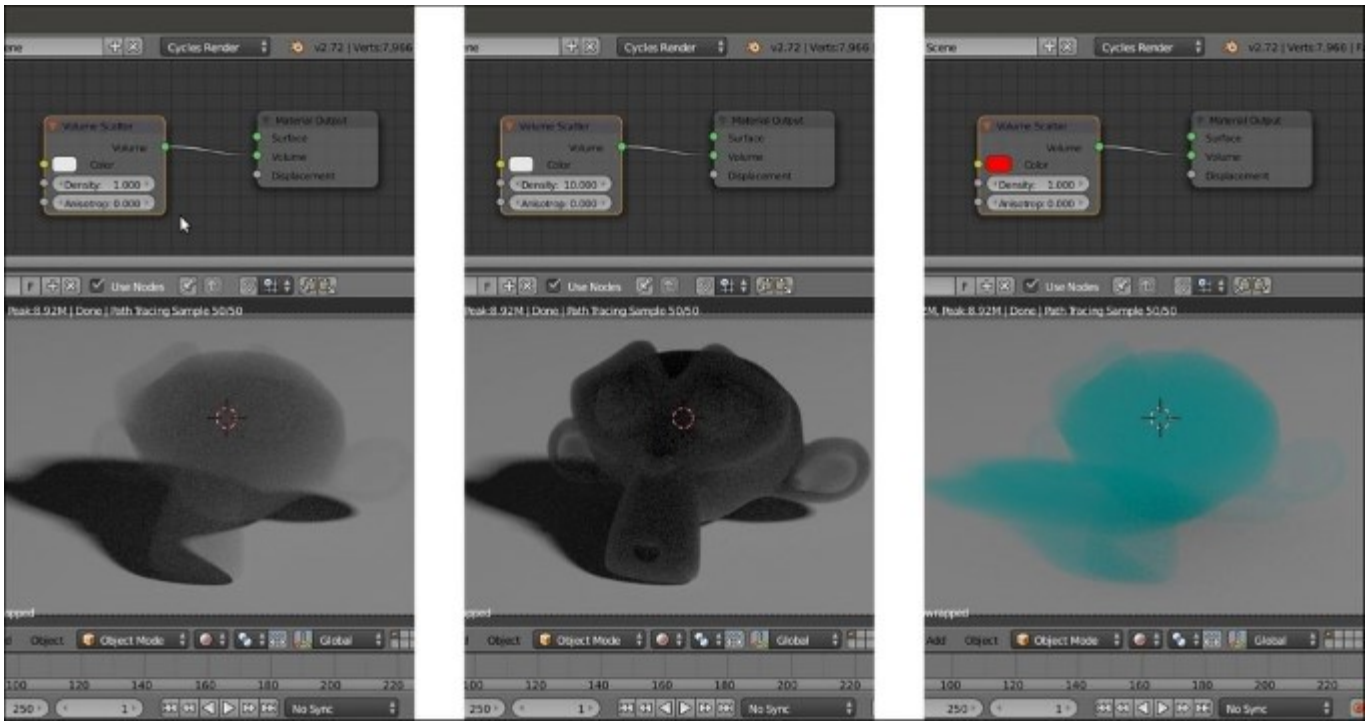
First, let's see the Volume applied to our usual Suzanne mesh primitive by performing the following steps:

1. Move the mouse to the **Camera** view and press *Shift* + *Z* to switch the **Viewport Shading** mode to **Rendered**.
2. Make sure that you have the **Suzanne_unwrapped** object selected, and click on the **New** button in the **Node Editor** toolbar, or in the **Material** window under the main **Properties** panel.
3. In the **Node Editor** window, press *Ctrl* and click and drag a line onto the link connecting the **Diffuse BSDF** shader to the **Material Output** node to cut it away. Because nothing is connected to the **Material Output** node sockets, in the **Camera** view, the **Suzanne** object turns pitch black as shown in the following screenshot:



The Diffuse shader connected and disconnected from the Material Output node

4. Select and delete the **Diffuse BSDF** shader node. Still in the **Node Editor** window, add a **Volume Scatter** node (press *Shift* + *A* and navigate to **Shader** | **Volume Scatter**) and connect its output to the **Volume** input socket of the **Material Output** node.

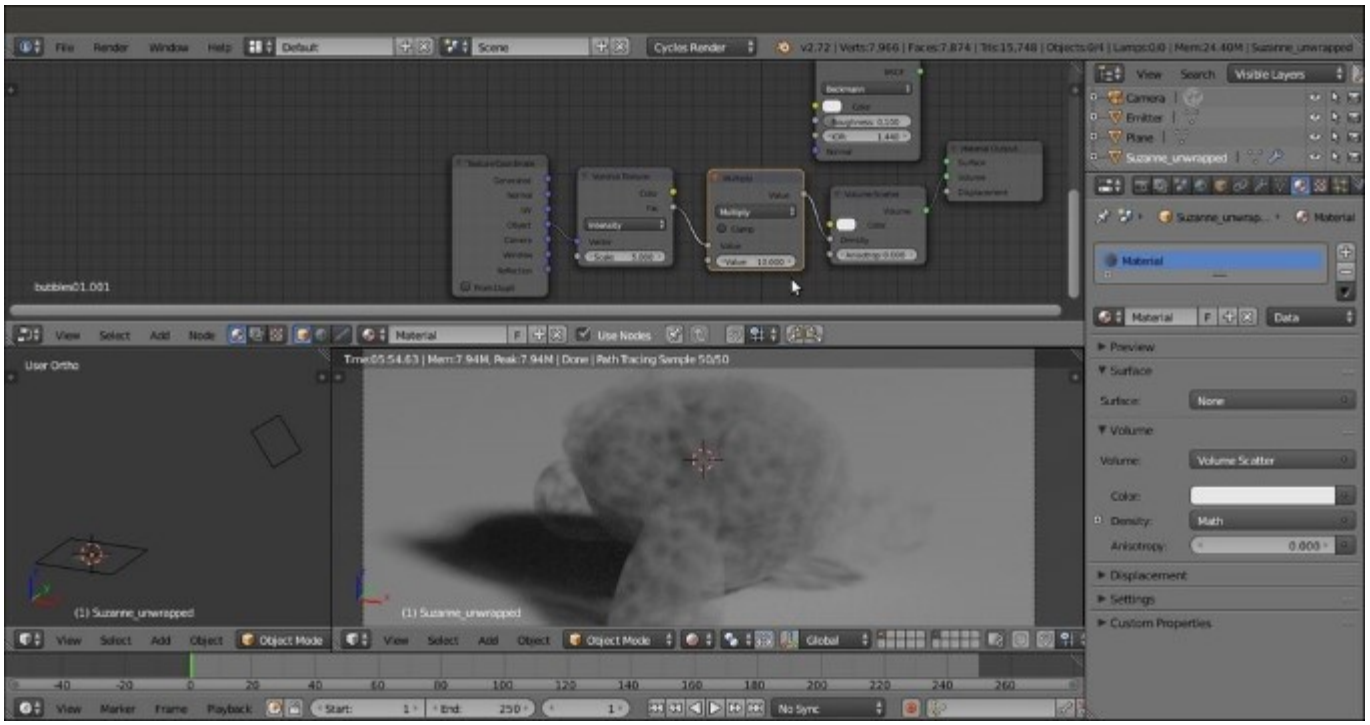


Different effects of the Volume Scatter node obtained by changing density and color

5. Try to increase the **Density** value to `10.000`, either in the node interface in the **Node Editor** window, or in the slot under the **Volume** subpanel in the main **Properties** panel. Suzanne's volume looks more solid, as shown in the middle of the preceding screenshot.
6. Change the **Density** value back to the default `1.000` and change the **Color** values of the **Volume Scatter** node for **R** to `1.000`, **G** to `0.000`, and **B** to `0.000` (a red color). The **Suzanne** object now appears as complementary colored smoke (on the right side of the preceding screenshot) because light is scattered (note that the shadow on the Plane gets the same color).
7. Add a **Glass BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glass BSDF**) and connect its output to the **Surface** input socket of the **Material Output** node. Set the **IOR** value to `1.440` and the **Roughness** value to `0.100`.

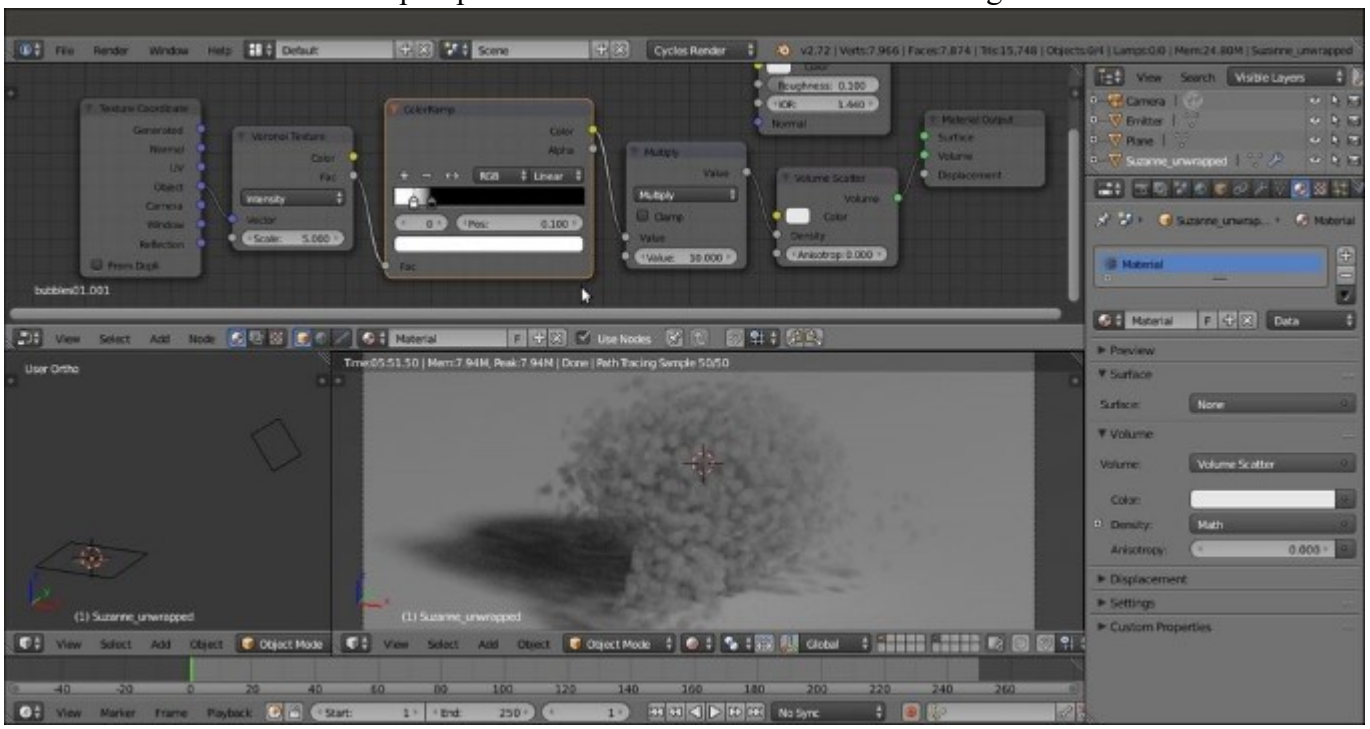
Adding a glassy envelope to the bluish, scattered glassy volume

8. Now you have to temporarily remove the connection of the **Glass BSDF** shader node to the **Surface** input socket of the **Material Output** node, take back the **RGB** value and set it to `0.800` for the **Color** of the **Volume Scatter** node.
9. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**), and a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**).
10. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Voronoi Texture** node, and the **Fac** output of this node to the first **Value** input socket of the **Math** node. Set the second **Value** to `10.000`, set **Operation** to **Multiply**, and connect its output to the **Density** input socket of the **Volume Scatter** node as shown in the following screenshot:



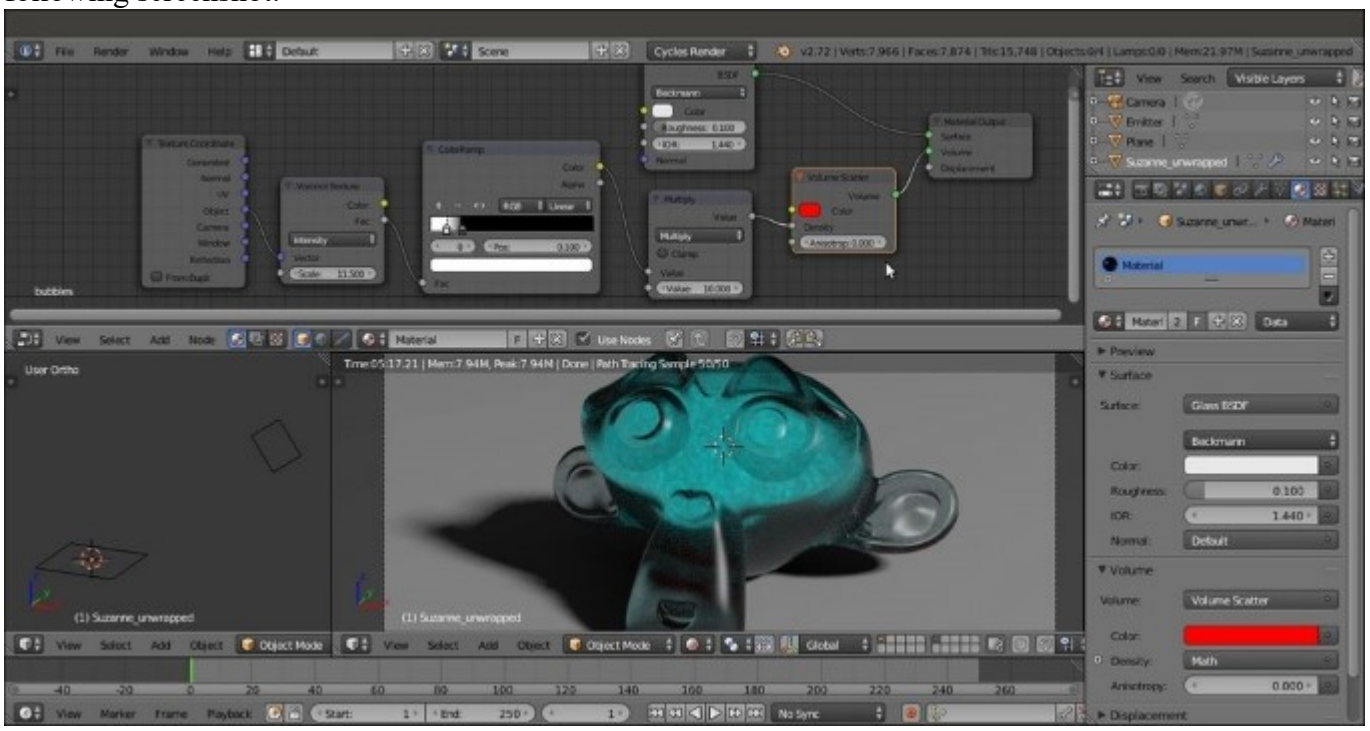
The Density value of the Volume Scatter node driven by a Voronoi texture output

11. Add a **ColorRamp** node (press *Shift* + *A* and go to | **Converter** | **ColorRamp**) and paste it between the **Voronoi Texture** and the **Math** nodes. Move the black color stop to position `0.195` and the white color stop to position `0.100` as shown in the following screenshot:



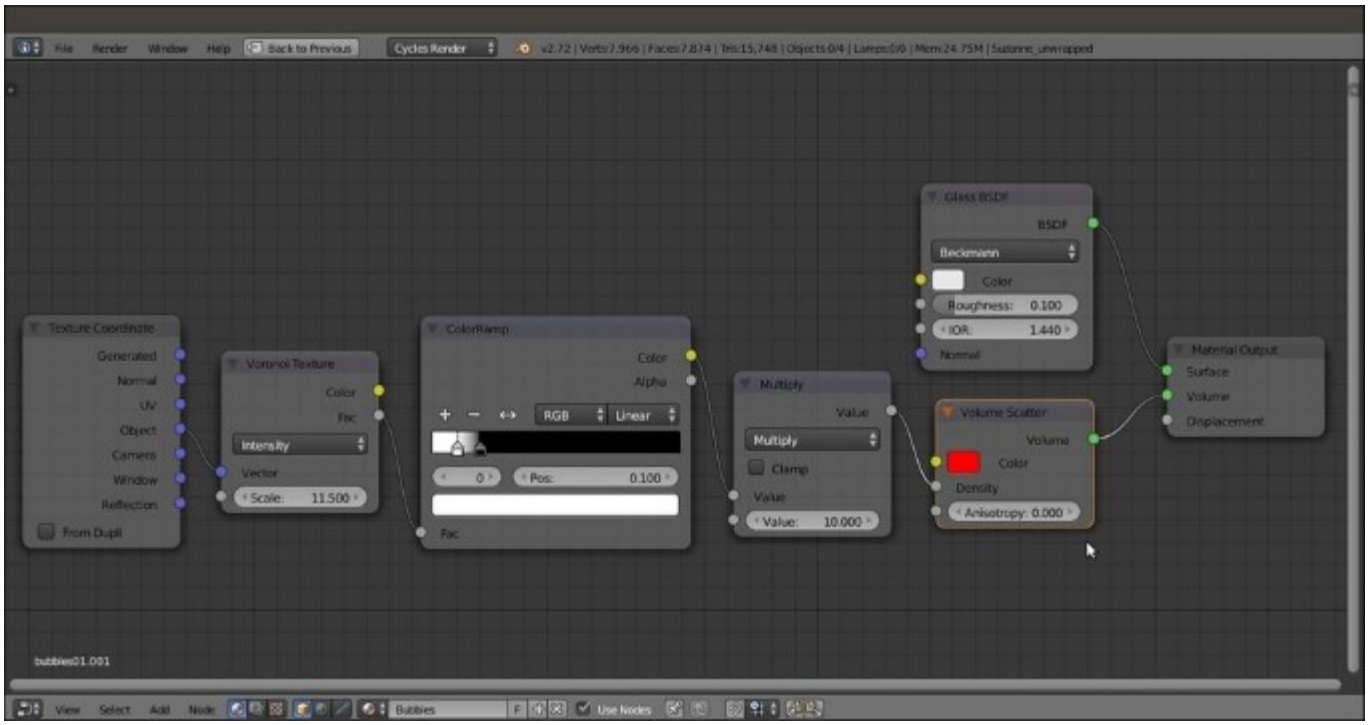
Enhancing the contrast of the texture output

12. Set the **Voronoi Texture** node's **Scale** value to `11.500`, and reconnect the **Glass BSDF** shader node output to the **Surface** input socket of the **Material Output** node. Change the **Color** values of the **Volume Scatter** node for **R** to `1.000`, **G** to `0.000`, and **B** to `0.000` as shown in the following screenshot:



The previously cloudy volume covered with a glass surface

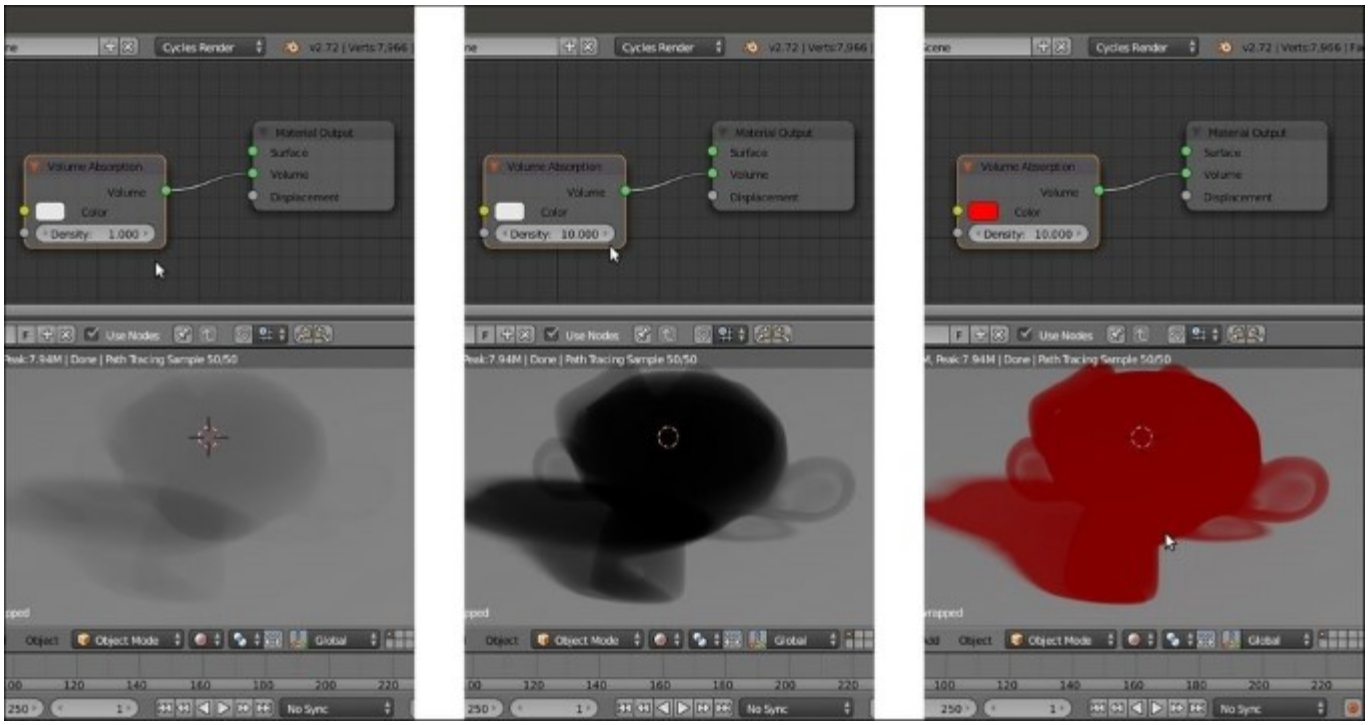
13. Rename the material as `Bubbles` and save the file by naming it `9931OS_09_volume.blend`. Have a look at the following screenshot:



The overall network for the combined surface and volume material

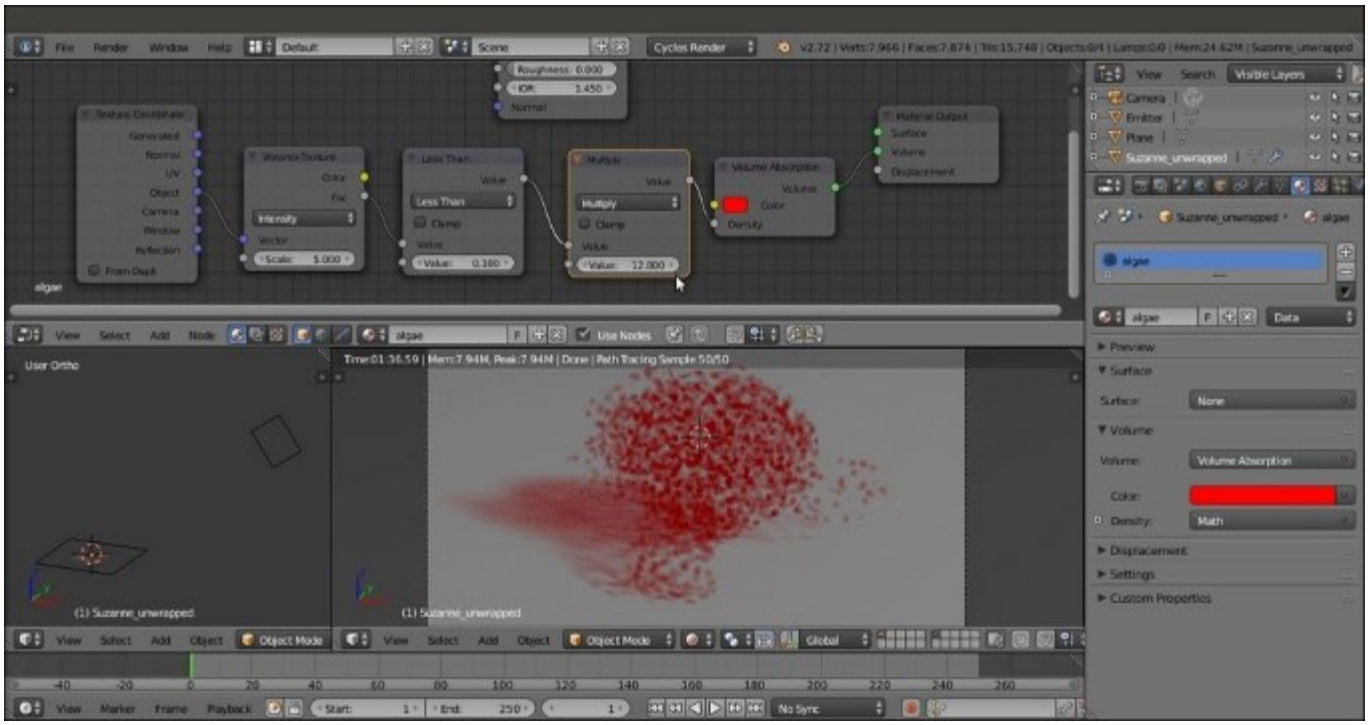
So, for the previous material named `Bubbles`, we used the **Volume Scatter** node. What about the **Volume Absorption** node?

14. In the **Node Editor** toolbar, enable *fake user* for the `Bubbles` material, and click on the **X** icon button to delink the datablock. Then click on the **New** button.
15. Delete the **Diffuse BSDF** shader node and add a **Volume Absorption** node (press *Shift* + *A* and navigate to **Shader** | **Volume Absorption**). Then connect it to the **Volume** input socket of the **Material Output** node.
16. To make a comparison with the **Volume Scatter** node, raise the **Density** value of the **Volume Absorption** node to `10.000` and set the **Color** values for **R** to `1.000`, **G** to `0.000`, and **B** to `0.000`. Have a look at the following screenshot:



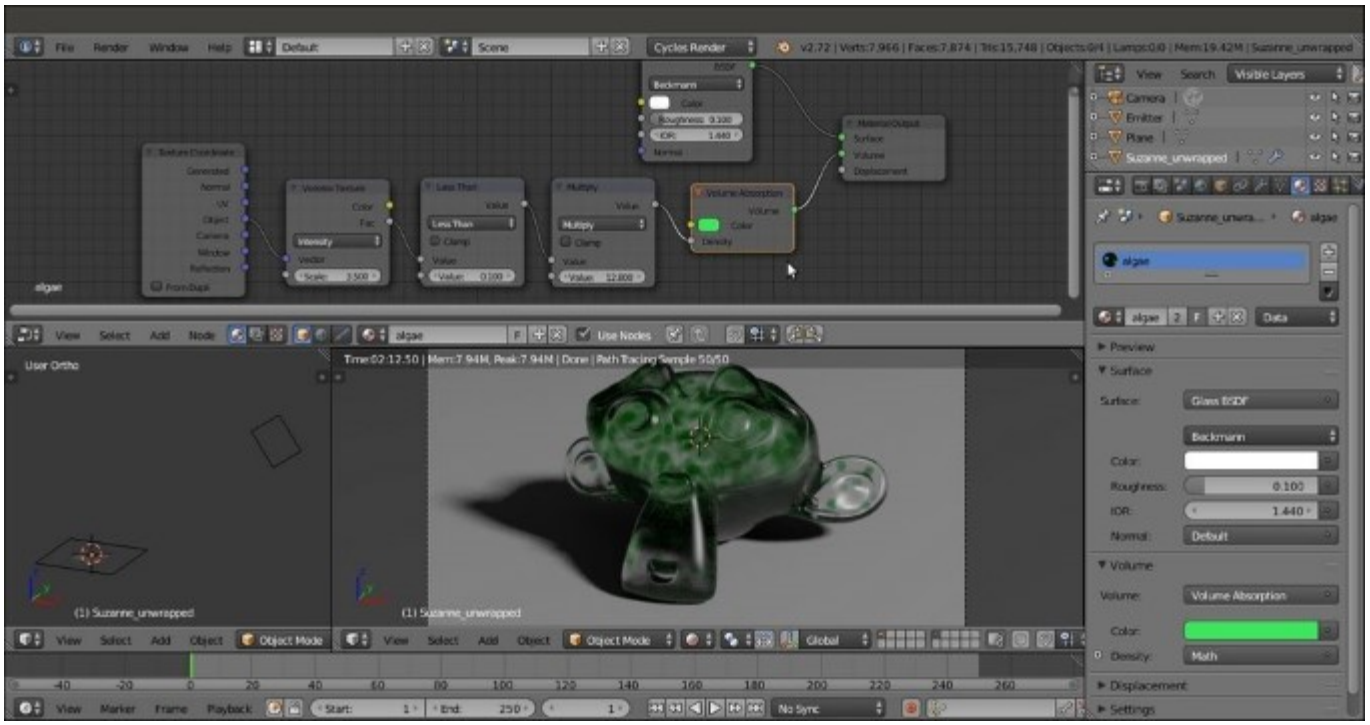
Different effects of the Volume Absorption node

17. Add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**), a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**), two **Math** nodes (press *Shift* + *A* and navigate to **Converter** | **Math**), and a **Glass BSDF** shader (press *Shift* + *A* and navigate to **Shader** | **Glass BSDF**).
18. Connect the **Object** output of the **Texture Coordinate** node to the **Vector** input socket of the **Voronoi Texture** node, and the **Fac** output of this node to the first **Value** input socket of the first **Math** node. Set the second **Value** to `0.100`, set **Operation** to **Less Than**, and connect its output to the first **Value** input socket of the second **Math** node. Set the second **Value** to `12.800` and the **Operation** to **Multiply**.
19. Connect the output of this **Multiply-Math** node to the **Density** input socket of the **Volume Absorption** node, and rename the material as `algae`. Here is a screenshot for your reference:



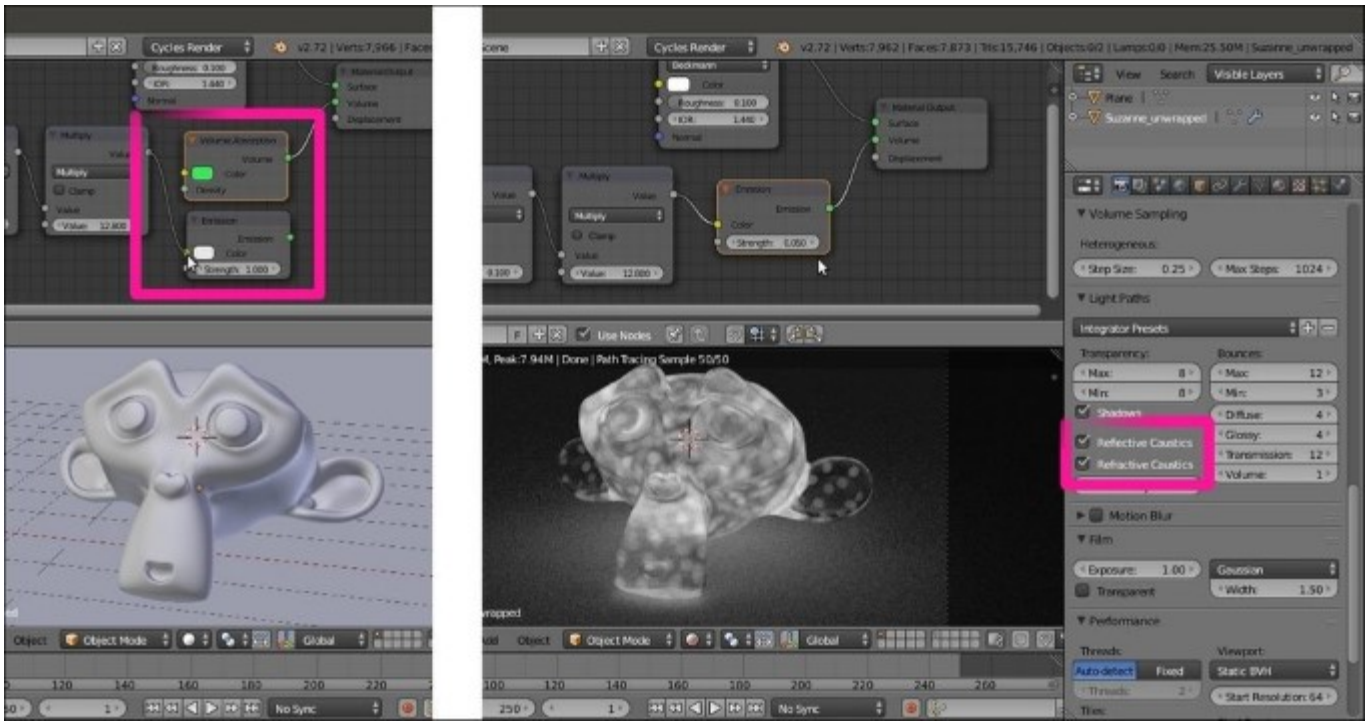
The density of the Volume Absorption node driven by the Voronoi Texture output

20. Set the **Scale** value of the **Voronoi Texture** node to `3.500`, change the **Color** of the **Volume Absorption** node for **R** `0.045`, **G** `0.800`, and **B** `0.113`, and connect the output of the **Glass BSDF** shader node to the **Surface** input socket of the **Material Output** node. Set the **IOR** value to `1.440` and the **Roughness** value to `0.100` as shown in the following screenshot:



Different colors and a glass cover for the absorption volumetric material

21. In the **Node Editor** toolbar, enable *fake user* for the `algae` material, and then click on the **2** icon (**Display number of users for this data**) to create a duplicate of the material, named `algae.001`.
22. Rename the material as `emitting_volume` and substitute the **Volume Absorption** node with an **Emission** node (press *Shift* + *A* and go to | **Shader** | **Emission**). Connect the output of the **Multiply-Math** node to the **Color** input socket, and set the **Strength** value to `0.050`.
23. Disable the visibility of the sixth scene layer to hide the **Emitter** mesh-light, and go to the **Render** window. In the **Light Paths** subpanel, enable both the **Reflective Caustics** and **Refractive Caustics** items. Here is a screenshot for your reference:



Substituting the Volume Absorption node with an Emission node as the volume material

24. Enable *fake user* for the `emitting_volume` material and save the file.

How it works...

In this tour recipe, we saw the three shaders used for the volumetric attribute of a material in Cycles, that is, the **Volume Scatter**, **Volume Absorption**, and **Emission** shaders (we have already seen the **Emission** shader the previous chapters, and it is commonly used in Lamps and mesh-lights).

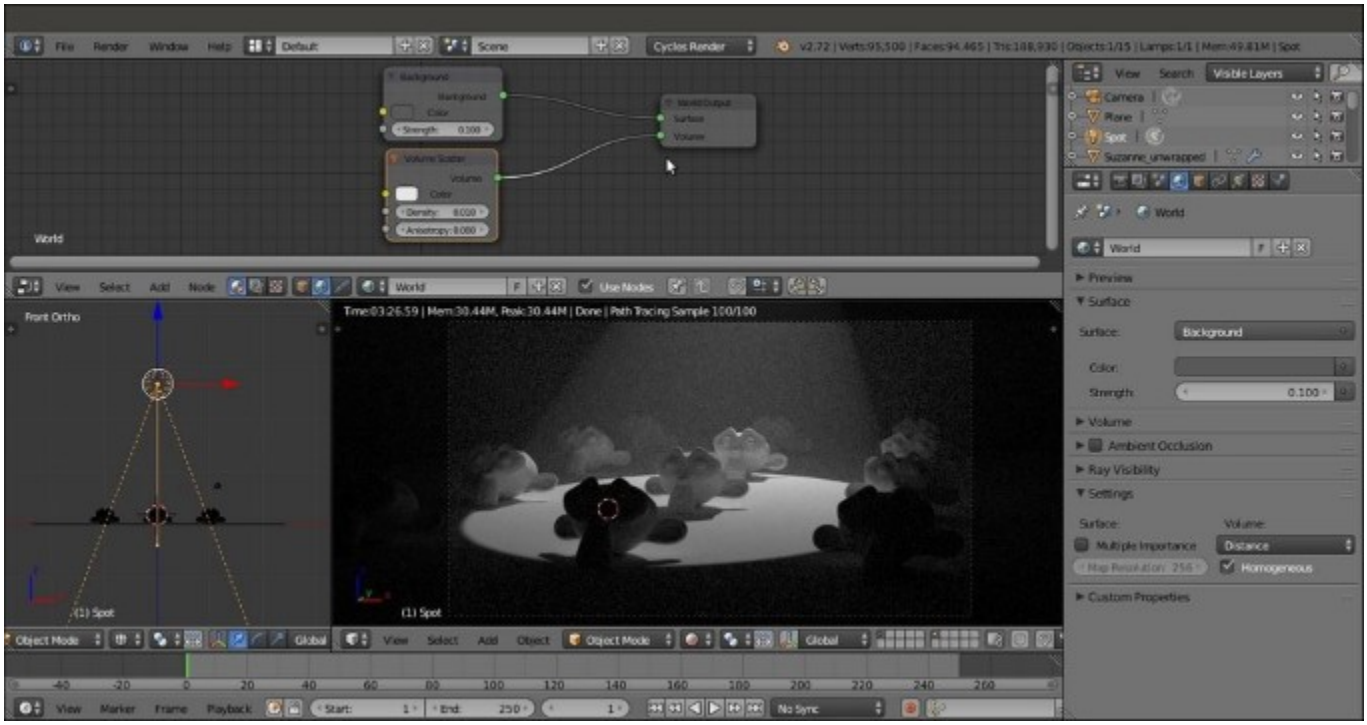
The **Volume Scatter** and **Absorption** shaders do exactly what their names say, as we saw in the examples. If we give them a color other than black, gray, or white, the **Volume Scatter** shader returns a complementary hue, while the **Volume Absorption** shader returns the same hue we set up.

About the **Density** value, remember that the higher the value, the more particles inside the volume. This allows for simulation of very light and rarefied vapors or very dense clouds of smoke, where the material looks almost solid.

There's more...

A Volume can be associated not only with objects but also with the World. This allows for several effects, for example, mist, or the famous God's rays. They are obtained by simply scattering light in the air of a Spot lamp.

The setup is really simple and intuitive: a **Volume Scatter** node connected to the **Volume** input socket of the **World Output** node. Have a look at the following screenshot:



The cone of a Spot lamp visible through the ambient volume material

The **Density** value of the **Volume Scatter** node in this case is set very low (0.010) to allow the light of the Spot lamp to shine through.

Open the `9931OS_09_volume_ambient.blend` file to have a look.

See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Materials/Volume

Creating a cloud volumetric material

The natural consequence of a volumetric material is (quite obviously) clouds.

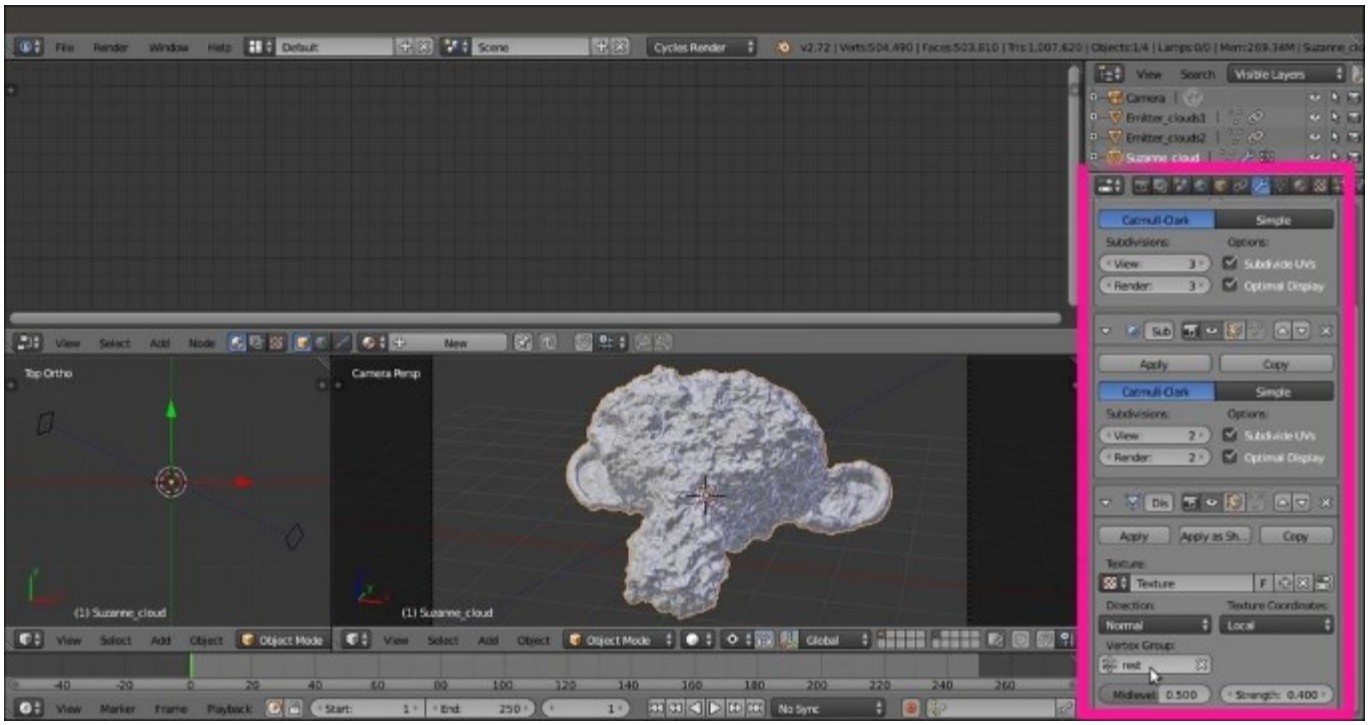
A simple way to create clouds in Cycles is by modeling the desired shape and then assigning an appropriate volumetric material. In the following screenshot, you can see this method applied to the usual Suzanne mesh:

The volumetric Suzanne cloud as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_09_cloud_start.blend` file, where there is the **Suzanne_cloud** object with two **Emitter** mesh-lights and a bright **World** set with a **Sky Texture**.

1. Go to the **Object modifiers** window and raise the **Subdivisions** level of the **Subdivision Surface** modifier from `2` to `3` for both **Preview** and **Render**.
2. Assign a new **Subdivision Surface** modifier. Set the **Subdivisions** level to `2`.
3. Assign a **Displace** modifier. Click on the **Show texture in texture tab** button to the right of the **New** button to switch to the **Texture** window.
4. Click on the **New** button under the **Displace** item, then click on the **Type** slot to switch from **Image or Movie** to **Clouds**. Set the **Size** to `0.35` and the **Depth** to `3`.
5. Go back to the **Object modifiers** window and click on the **Vertex Group** slot to select the **rest** item. Then set the **Strength** value to `0.400`.



The displaced Suzanne cloud seen in the Solid Viewport Shading mode and the assigned modifiers in the main properties panel to the right

How to do it...

After creating the cloud shape, let's make a start on the material:

1. Ensure that the **Suzanne_cloud** object is still selected, and click on the **New** button in the **Node Editor** toolbar or in the **Material** window under the main **Properties** panel.
2. In the **Node Editor** window, select and delete the **Diffuse BSDF** shader node.
3. Add a **Volume Scatter** node (press *Shift* + *A* and navigate to **Shader** | **Volume Scatter**), a **Volume Absorption** node (press *Shift* + *A* and navigate to **Shader** | **Volume Absorption**), and an **Add Shader** node (press *Shift* + *A* and navigate to **Shader** | **Add Shader**).
4. Connect the output of the **Volume Absorption** node to the first **Shader** input socket of the **Add Shader** node, and the **Volume Scatter** output to the second **Shader** input socket. Connect the output of the **Add Shader** node to the **Volume** input socket of the **Material Output** node.
5. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Add Shader** node and the **Material Output** node. Connect the output of the **Volume Scatter** node to the second **Shader** input socket of the **Mix Shader** node. Set the **Fac** value to `0.700`.
6. Add a **Value** node (press *Shift* + *A* and navigate to **Input** | **Value**) and an **RGB** node (press *Shift* + *A* and navigate to **Input** | **RGB**). Connect the **Value** output to both the **Density** input sockets of the **Volume Absorption** and **Volume Scatter** nodes. Set the input value to `5.000`.

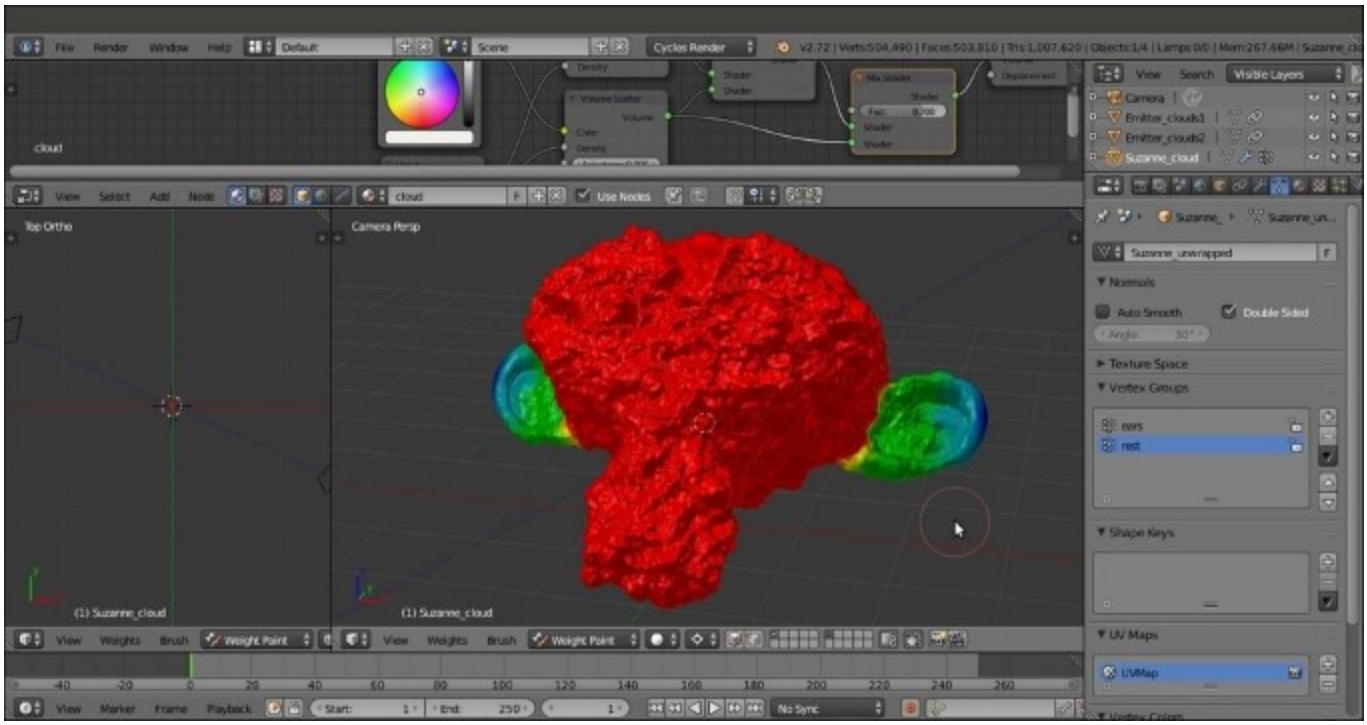
7. Connect the output of the **RGB** node to the **Color** input sockets of the **Volume Absorption** and **Volume Scatter** nodes. Set the color values for **R** to `0.890`, **G** to `0.866`, and **B** to `0.832`.
8. Under the **Material** windows, go to the **Settings** subpanel and enable the **Homogeneous** item.
9. Go to the **Scene** window, and under the **Color Management** subpanel, enable the **Use Curves** item. Then click on the curves window to create a new control point. Set its coordinates as **X** to `0.67206` and **Y** to `0.88125`. Click again to create a new control point, and set the coordinates as **X** to `0.31579` and **Y** to `0.46875`. Have a look at the following screenshot:

The Rendered preview of the Suzanne cloud and the material network in the Node Editor window

How it works...

Most of the effect of this material is because of the **Displace** modifier deforming the Suzanne mesh in order to resemble the shape of a Suzanne cloud. The material itself is simply a combination of the **Volume Scatter** and the **Volume Absorption** shader, first added by the **Add Shader** node and then with a **Mix Shader** node to further control the mixing of the scattering in the whole shader.

The **Vertex Group**, **rest**, selected in the **Displace** modifier slot is simply a group with lower weight going towards the ears, because they are quite thin, and the displacement we're using can easily cause bad mesh intersections.



The Rest Vertex Group visible in the Weight Paint mode

In the last step, we enabled curves for the Color Management to obtain a brighter and more contrasted rendering of the cloud against the sky.

Creating a fire and smoke shader

In this recipe, we are going to see one of the most exciting effects we can obtain in Cycles—a fire and smoke simulation effect:

The fire and smoke shader as it appears in the final rendering when assigned to the Suzanne mesh

Getting ready

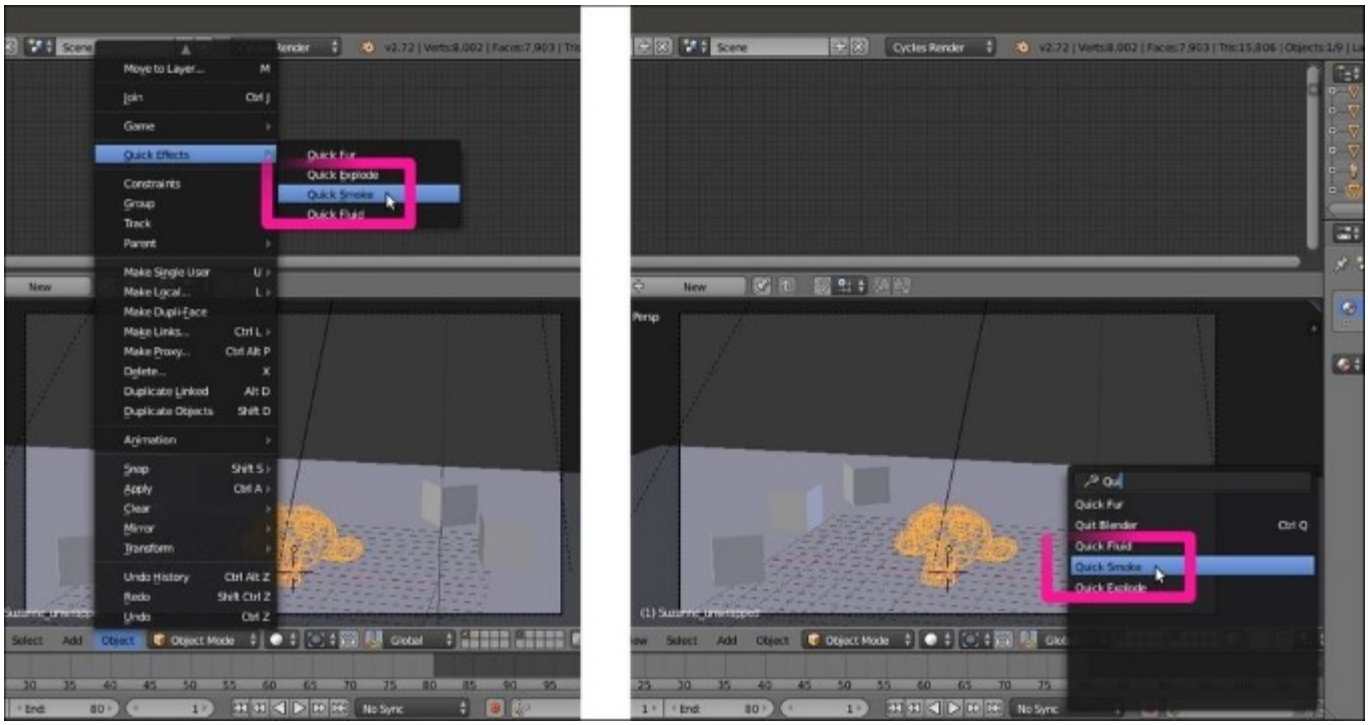
Start Blender and open the `9931OS_09_fire_smoke_start.blend` file, where there is the **Suzanne_unwrapped** object leaning on a Plane and surrounded by five small Cubes, a Spot lamp, the Camera, and a medium-intensity World with a **Sky Texture**.

1. Go to the **Render** window, and under the **Sampling** subpanel, set the **Preview** samples to `50` and the **Render** samples to `100`. Then go to the **Dimensions** subpanel and set the **End Frame** to `80`. Under the **Light Paths** subpanel, enable the **Reflective** and **Refractive Caustics** items.

How to do it...

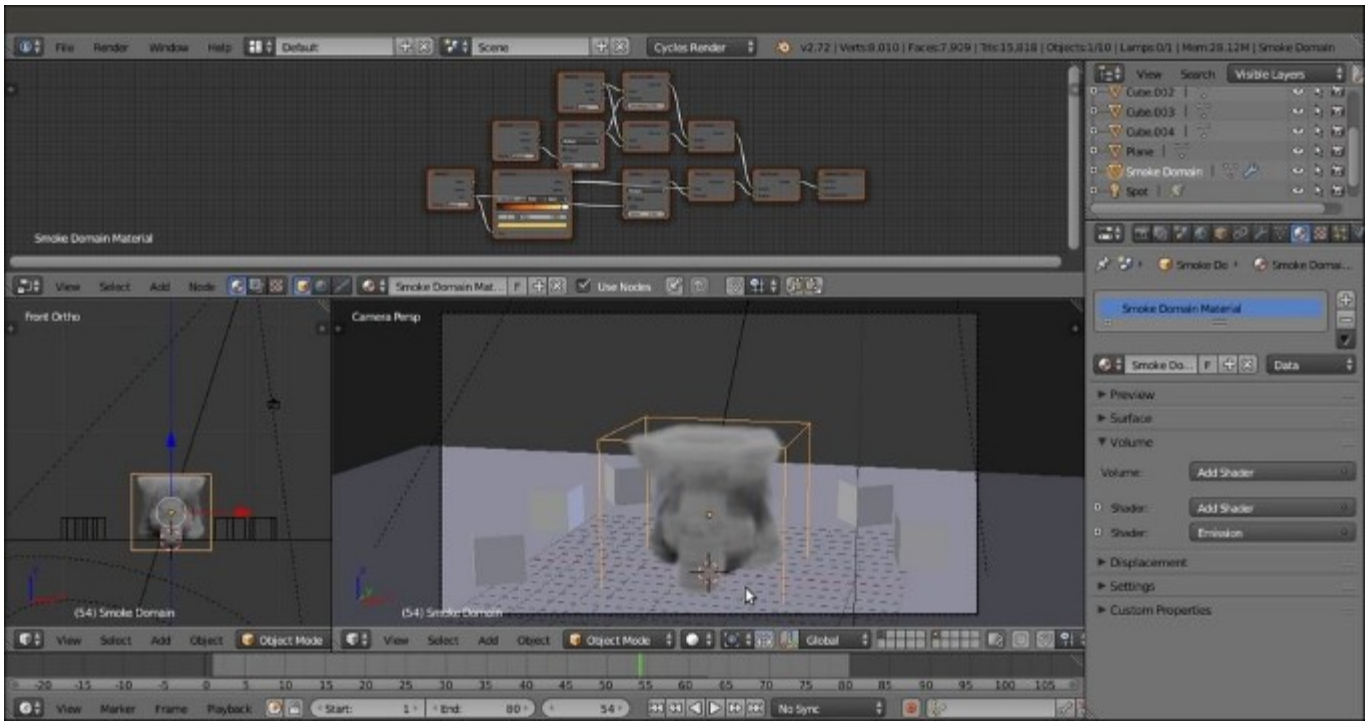
Let's start by creating the smoke simulation by a shortcut:

1. Select the **Suzanne_unwrapped** object, click on the **Object** item in the 3D viewport toolbar to go to **Quick effects**, and select the **Quick Smoke** item. Alternatively, press the spacebar, and in the search window, start typing `Quick`. Then select the **Quick Smoke** item from the menu as shown in the following screenshot:



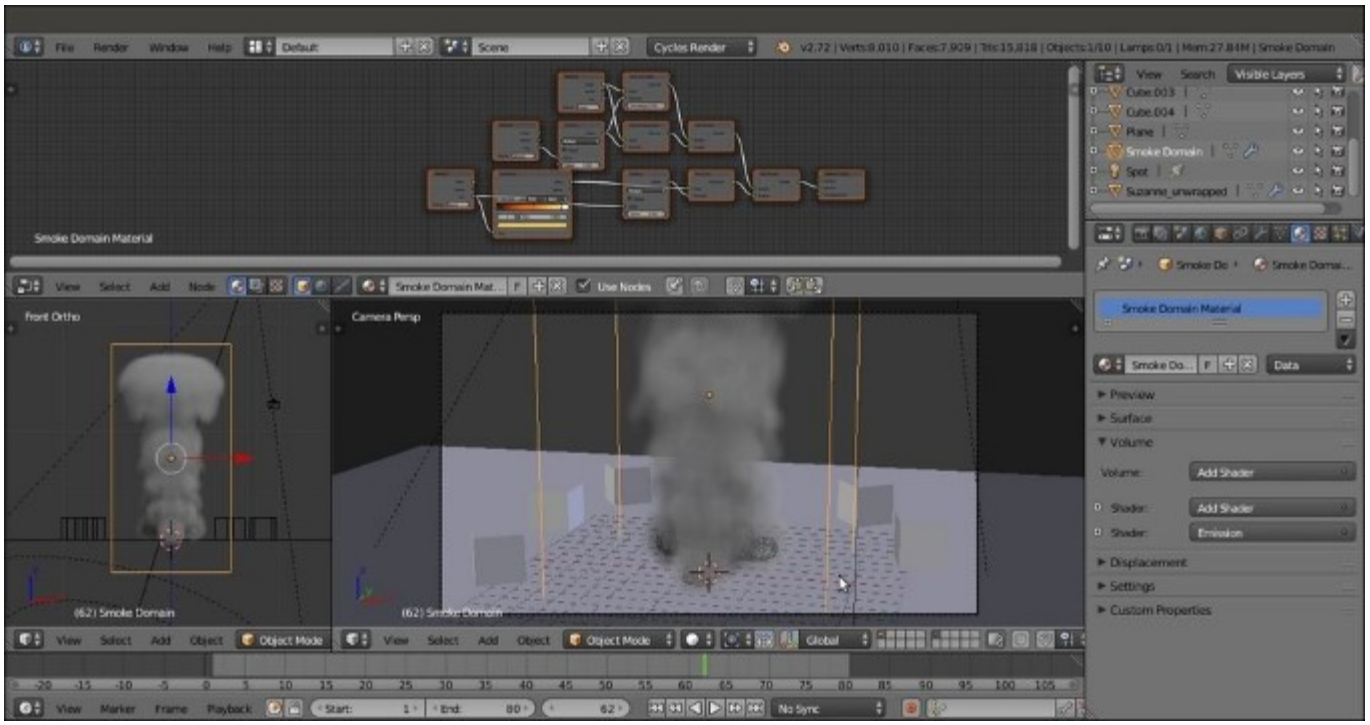
The Quick Effects menu

2. A **Smoke Domain** (the selected wire box around the **Suzanne** object) with a prepared fire/ smoke material (`Smoke Domain Material`) is automatically set up on the selected object, and when you press the **Play** button in the **Player Control** on the **Timeline** toolbar, the smoke simulation starts. Have a look at the following screenshot:



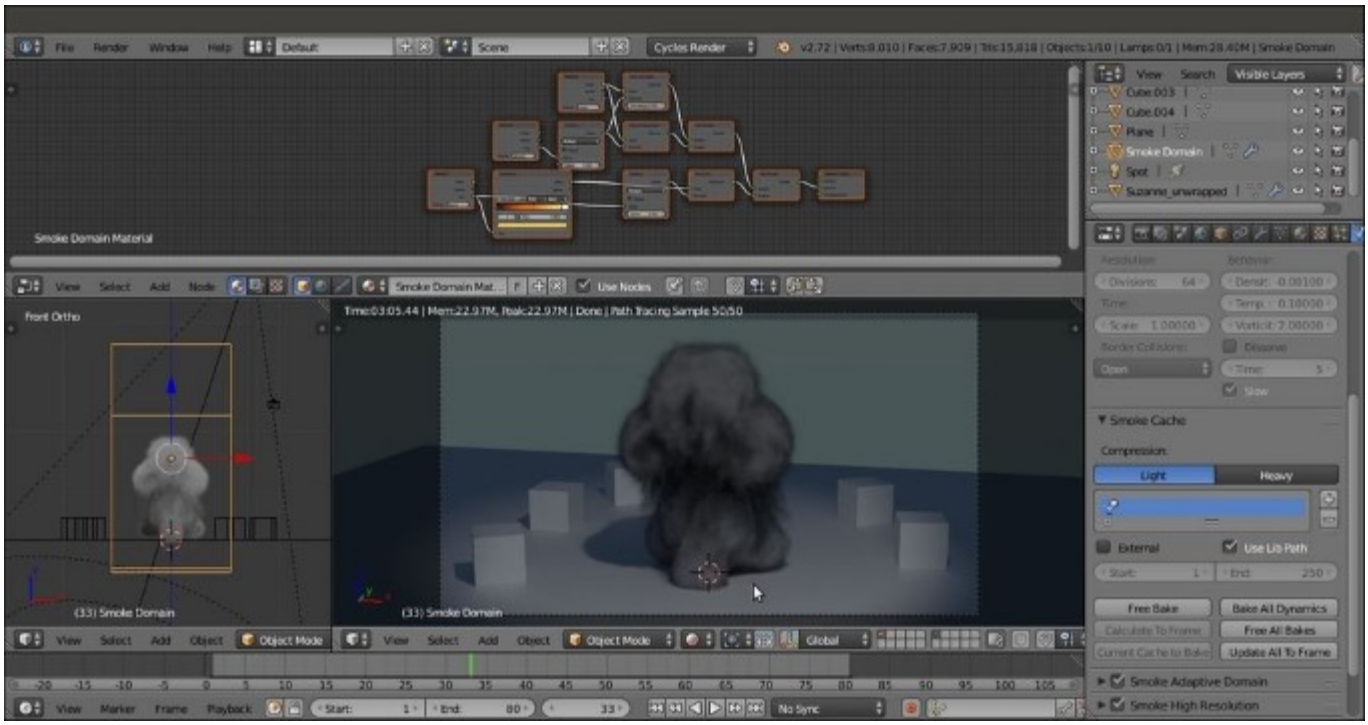
The smoke simulation: in the upper Node Editor window, the material being automatically created by the Quick Effects tool

3. Go back to frame **1** and scale the **Smoke Domain** to a larger size on the global *z* axis such that its top goes out of the Camera frame boundary. Scale it by `3.000` (press *S*, enter digit `3.000`, then press *Enter*). Then move it 5 units upwards.
4. Go to the **Physics** window, and under the **Smoke** subpanel, set the **Divisions** value to `64`. Then go to the **Smoke Flames** subpanel and set the **Speed** value to `2.00000`, **Smoke** to `0.50000`, and **Vorticity** to `1.00000`. Restart the **Play** button to recalculate the cache. Have a look at the following screenshot:



The smoke simulation inside a bigger domain box

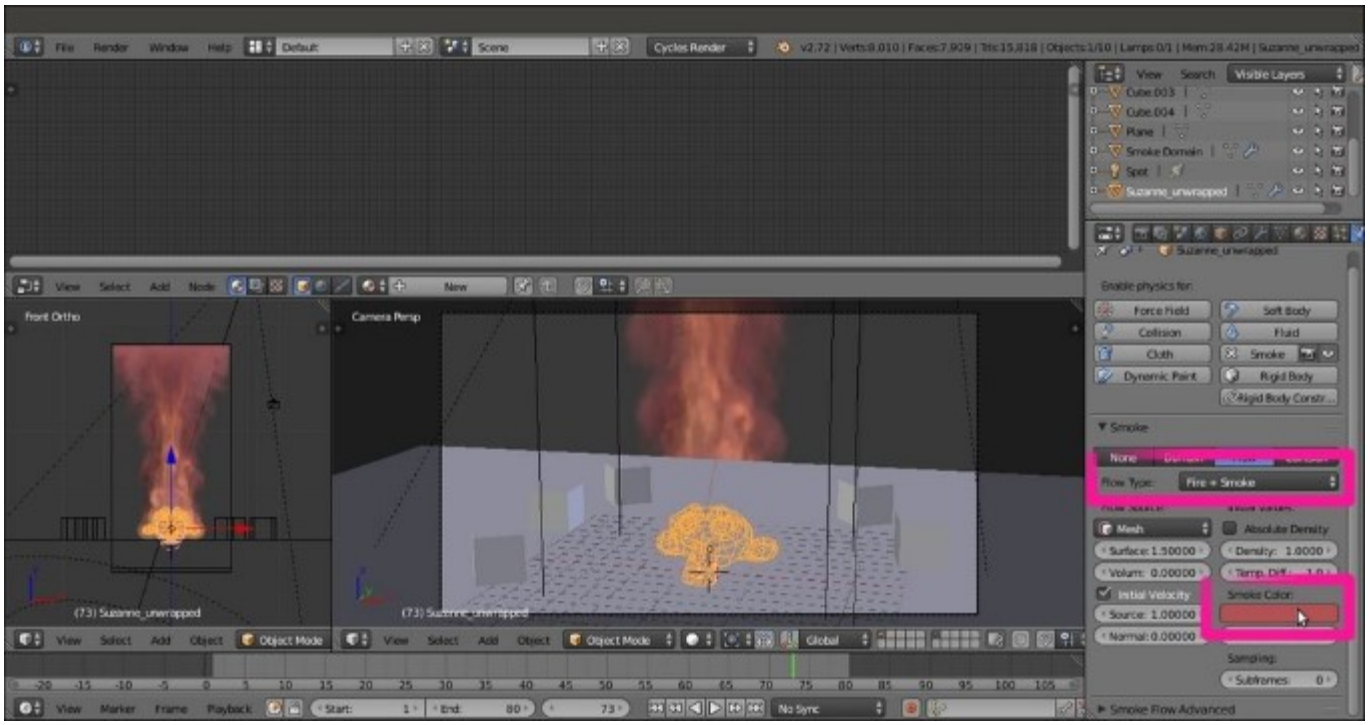
5. As all the 80 frames have been cached, go to the **Physics** window, and under the **Smoke Cache** subpanel, click on the Current Cache to Bake button.
6. Enable the **Smoke Adaptive Domain** and the **Smoke High Resolution** subpanels. Leave the default settings as they are.
7. With the mouse arrow in the **Camera** view, press *Shift* + *Z* to switch to the **Rendered** viewport shading mode. Have a look at the following screenshot:



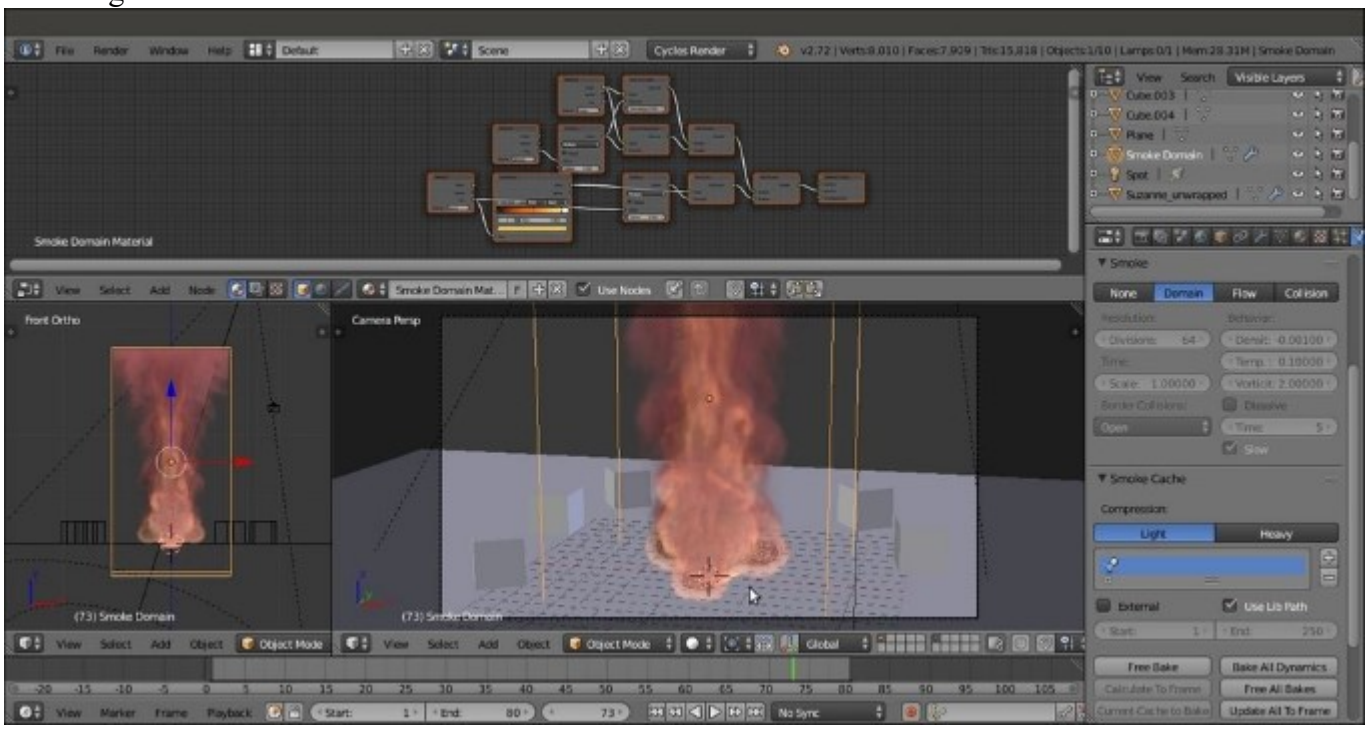
The smoke simulation seen in the Rendered preview

In the **Rendered** preview (be careful that the smoke is not supported by GPU yet), we can see the dark grey smoke, but what about the fire?

8. In the **Physics** window, under the **Smoke Cache** subpanel, click on the **Free All Bakes** button.
9. Go to the **Outliner** to select the **Suzanne_unwrapped** item. Then go to the **Physics** window again. Under the **Smoke** subpanel, click on the **Flow Type** slot and switch from **Smoke** to **Fire + Smoke**. Then click on the **Smoke Color** slot and change the color values for **R** to `0.700`, **G** to `0.317`, and **B** to `0.335`. Have a look at the following screenshot:



10. Reselect the **Smoke Domain** object and click on the **Play** button to cache the smoke simulation again, this time with reddish smoke and also the fire. Then click on the **Current Cache to Bake** button again.

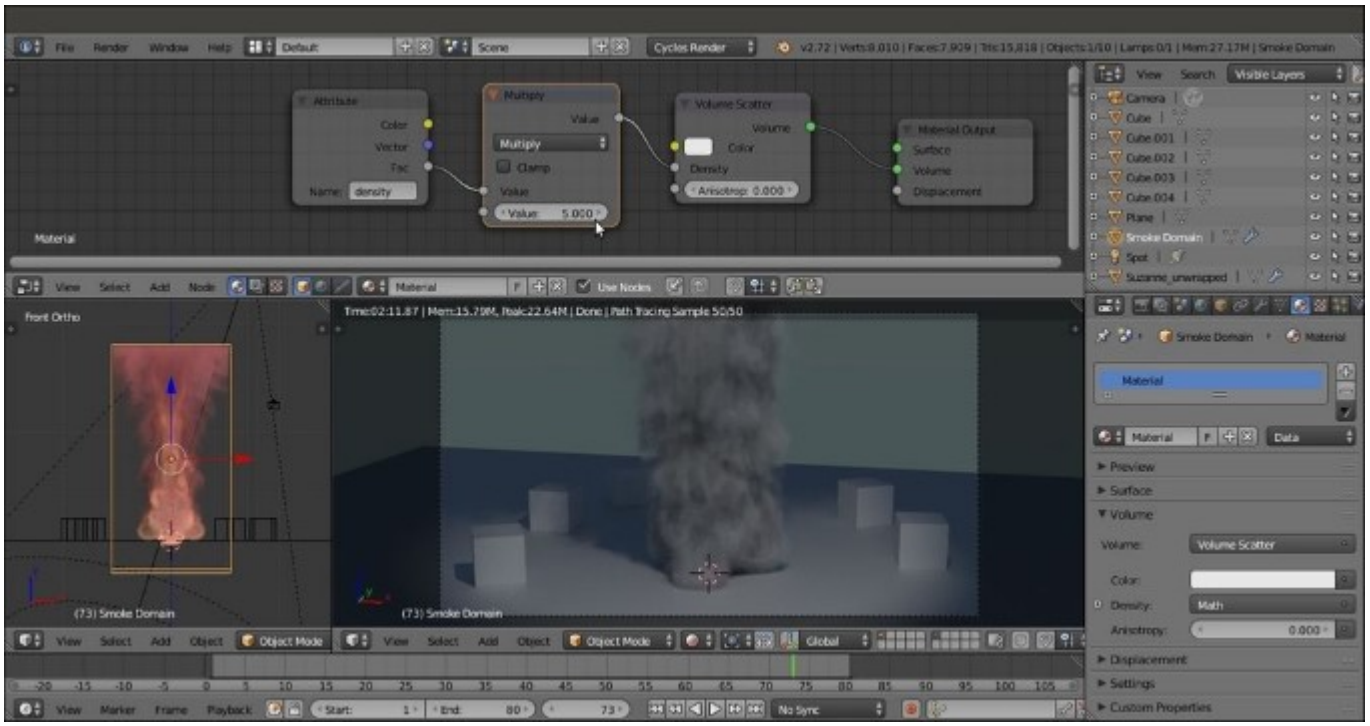


The new smoke (and fire) simulation

But this is a Cookbook about materials, so let's put aside the smoke simulation settings and concentrate on the material. To better understand how this works, let's delete the ready-made material and create a new material from scratch.

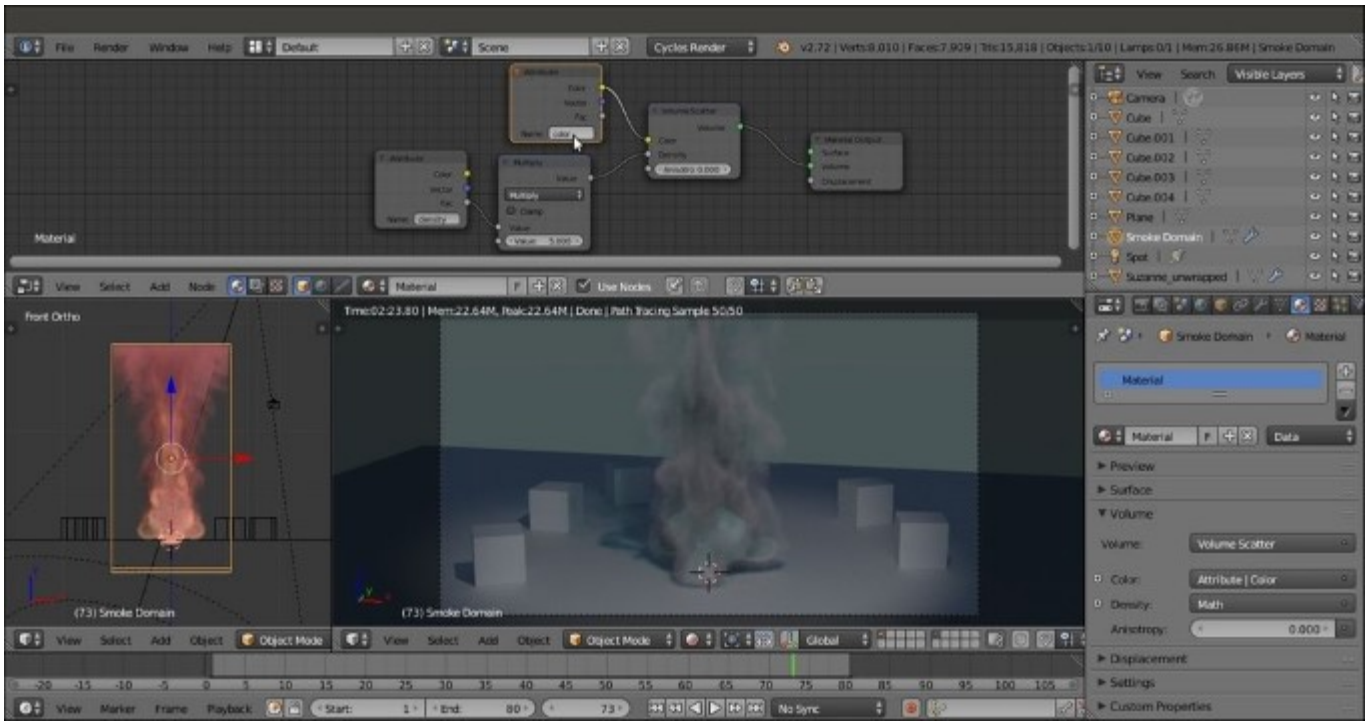
11. In the **Node Editor** toolbar, press *Shift* and click on the **X** button to unlink the `Smoke Domain Material`. Set the users to zero.

    Now, setting the **Camera** view shading mode to **Rendered** shows only the Smoke Domain box as a solid object because no material is assigned to the simulation.
12. Click on the **New** button in the **Node Editor** window toolbar. Delete the **Diffuse BSDF** shader node and add a **Volume Scatter** node (press *Shift* + *A* and navigate to **Shader** | **Volume Scatter**). Connect it to the **Volume** input socket of the **Material Output** node.
13. Add an **Attribute** node (press *Shift* + *A* and navigate to **Input** | **Attribute**) and connect its **Fac** output to the **Density** input socket of the **Volume Scatter** node. In the **Name** field of the **Attribute** node, write `density`.
14. Add a **Math** node (*Shift* + *A* | **Converter** | **Math**), set the **Operation** to **Multiply**, and paste it between the **Attribute** and the **Volume Scatter** nodes. Set the second **Value** to `5.000`.



Building the smoke density after deleting the default Quick Effects material

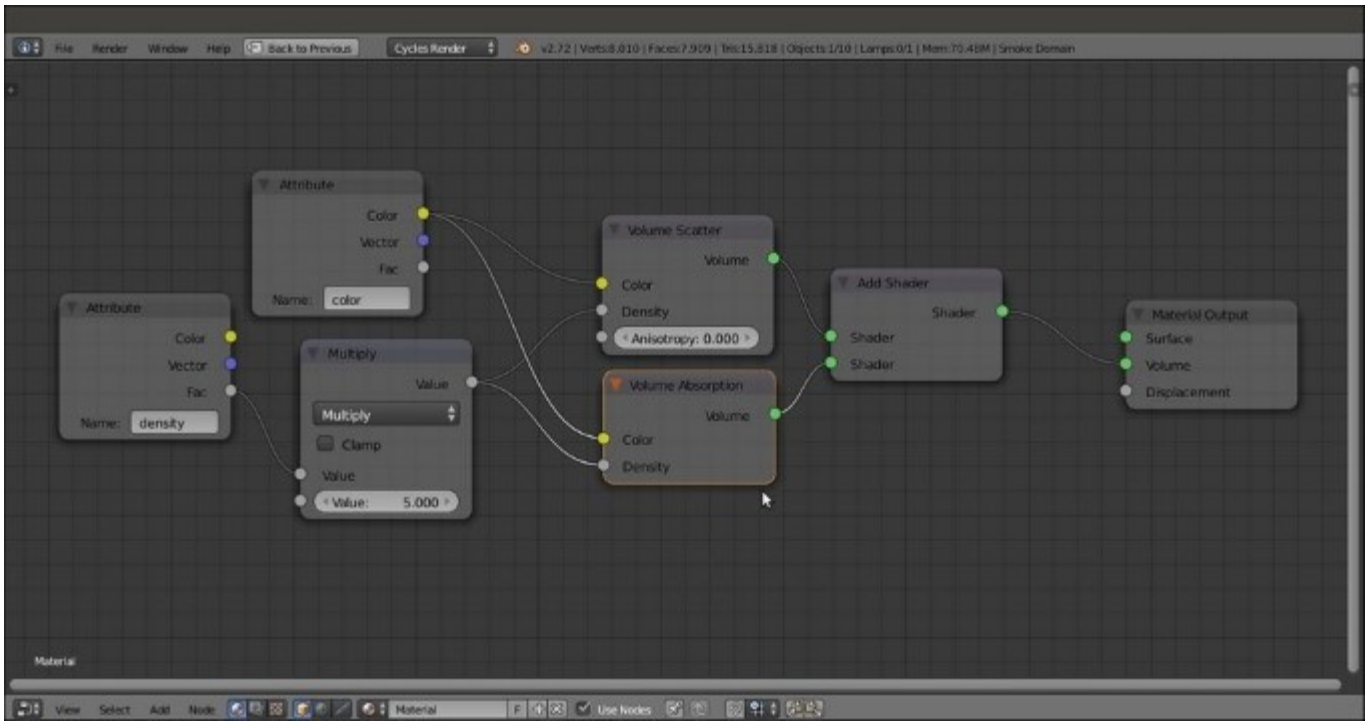
15. Press *Shift* + *D* to duplicate the **Attribute** node, connect the duplicated node's output to the **Color** input socket of the **Volume Scatter** node. In the **Name** field, write `color`.



The smoke color

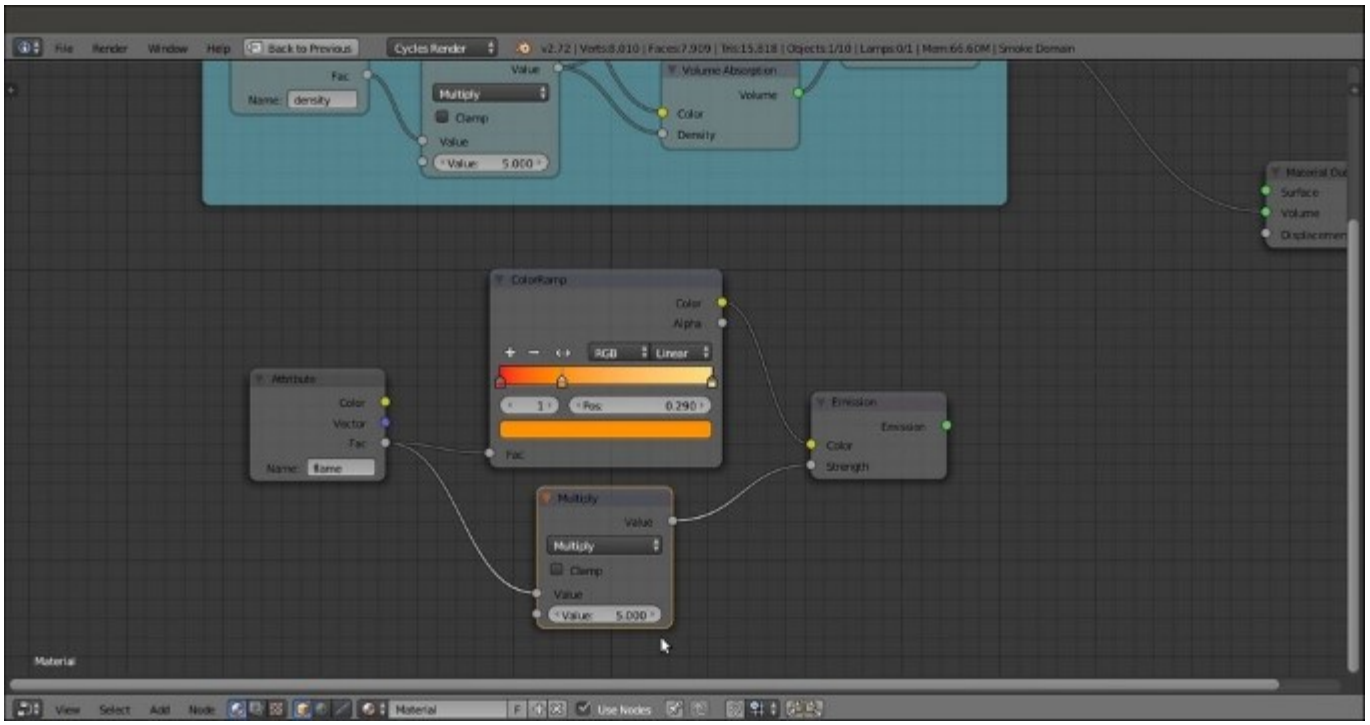
As you can see in the rendered preview, the smoke gets a bluish coloration, with the complementary color (orange) getting scattered.

1. Add a **Volume Absorption** node (*Shift* + *A* | **Shader** | **Volume Absorption**) and an **Add Shader** node (*Shift* + *A* | **Shader** | **Add Shader**). Paste the **Add Shader** node between the **Volume Scatter** and the **Material Output** nodes. Then connect the **Volume Absorption** output to the second **Shader** input socket of the **Add Shader** node.
2. Connect the **Color** output of the **Color-Attribute** node to the **Color** input socket of the **Volume Absorption** node, and the output of the **Multiply** node to the **Density** input socket of the **Volume Absorption** node as shown in the following screenshot:



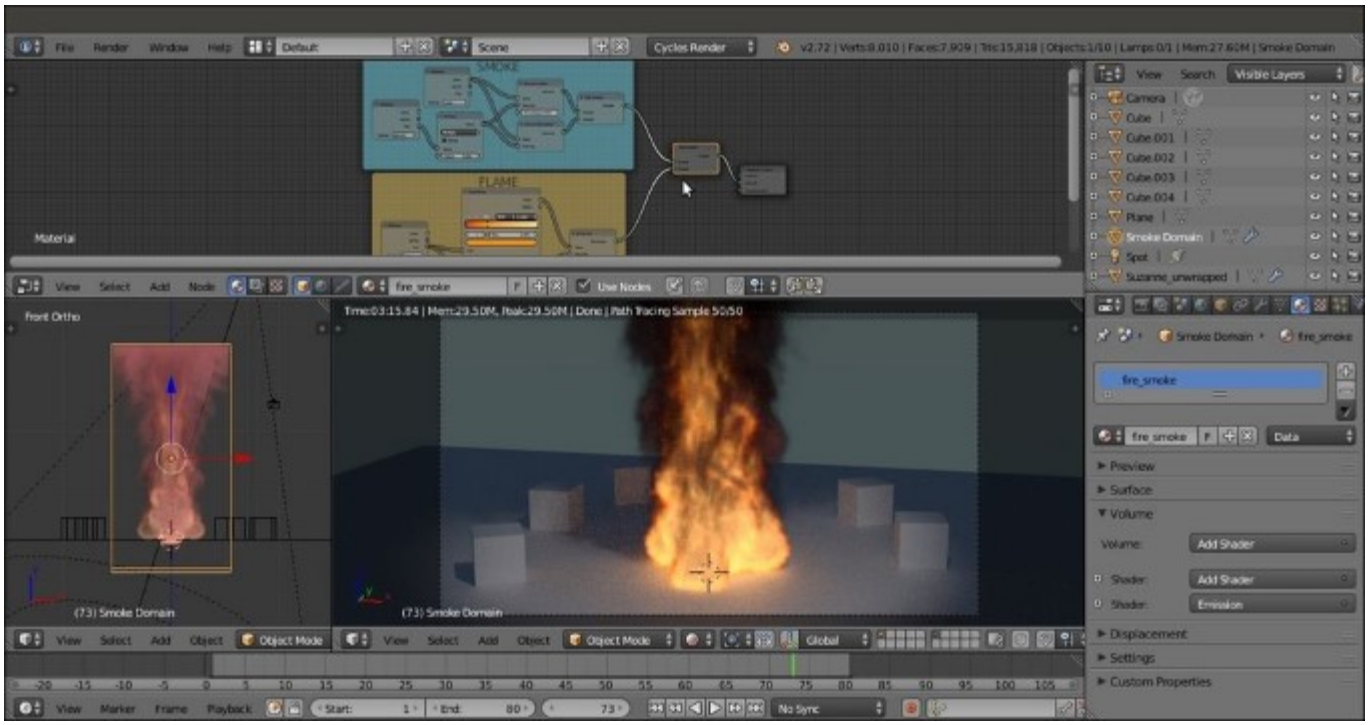
The complete smoke network

3. Parent all of these nodes, except the **Material Output** node, to a **Frame**. Label it as `SMOKE`.
4. Duplicate or add a new **Attribute** node, and in the **Name** field, write **flame**. Add an **Emission** shader (press *Shift* + *A* and navigate to **Shader** | **Emission**) and connect the **Fac** output to both the **Strength** and the **Color** input sockets of the **Emission** shader node.
5. Add a **ColorRamp** (*Shift* + *A* | **Converter** | **ColorRamp**) and paste it between the **Attribute** output and the **Color** input socket of the **Emission** node. Set the black color stop values for **R** to `1.000`, **G** to `0.000`, and **B** to `0.010`. Then set the white color stop values for **R** to `1.000`, **G** to `0.724`, and **B** to `0.224`. Add a new color stop and move it to the `0.290` position. Set the color values for **R** to `1.000`, **G** to `0.280`, and **B** to `0.000`.
6. Add a **Math** node and paste it between the **Attribute** output and the **Strength** input socket of the **Emission** node. Set the **Operation** to **Multiply** and the second **Value** to `5.000`.



Starting to build the fire shader

7. Parent these four nodes to a new **Frame** labeled as `FLAME`.
8. Add an **Add Shader** node (press *Shift* + *A* and navigate to **Shader** | **Add Shader**) and paste it between the output of the **Add Shader** inside the **SMOKE** frame and the **Material Output**. Connect the output of the **Emission** shader inside the **FLAME** frame to the second **Shader** input socket of the last **Add Shader** node.



Smoke and fire shaders added

9. Rename the material as `fire_smoke` and save the file as `9931OS_09_fire_smoke_final.blend`.

How it works...

Considering that we basically used the default settings for the fire and smoke simulation, the result was pretty good. To find out more about the different settings and types of smoke, take a look at the links provided in the *See also* section of this chapter.

We used the Quick Effects menu to let Blender automatically set up the smoke simulation for us. The steps usually involved are as follows:

- For the simulation domain, set an object that defines the bounds of the simulation volume. In our case, it was the **Smoke Domain** box in the wireframe viewport shading mode. It could be scaled, moved, or rotated if necessary.
- For the flow, set an object that determines where the smoke will be produced from, that is, the **Suzanne_unwrapped** object.
- Assign a material to the smoke. In our case, it was the automatically created `Smoke Domain Material`, which we then remade as the `fire_smoke` material.
- Bake the simulation by computing the cache for the required frames and then clicking on the **Current Cache to Bake** button.
- Save the blend file.

The `fire_smoke` material we set in the **Node Editor** is made exactly as shown in the previous volume recipes of this chapter. It consists of all the three shader nodes that can be used for a Volume: the **Volume Scatter**, the **Volume Absorption** and the **Emission** shaders, driven by the **flame**, **density** and **color** attributes we wrote in the **Name** field of the **Attribute** nodes and coming from the smoke simulation.

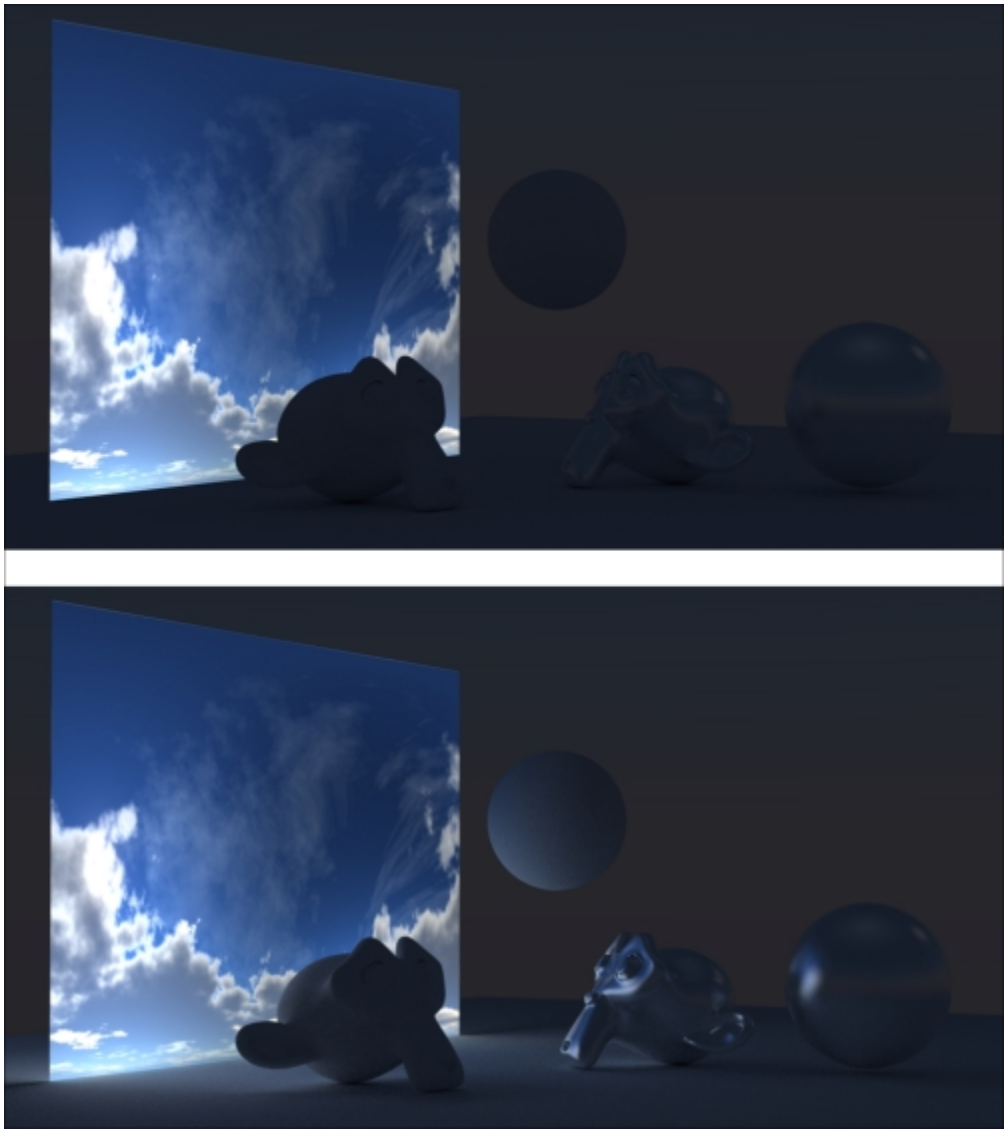
See also

- http://wiki.blender.org/index.php/Doc:2.6/Manual/Physics/Smoke
- https://cgcookie.com/blender/lessons/02-cycles-fire-and-smoke/
- http://www.blendernation.com/2014/04/14/rendering-smoke-and-fire-in-cycles/

Creating a shadeless material in Cycles

In this recipe, we will create a shadeless material, which is a material that behaves as self-illuminated but does not actually emit any light on the nearby objects.

In the following screenshot, we can see the difference between a Plane with a shadeless material and a Plane with an emitting material:



A shadeless Plane and an emitting Plane in comparison

At the top, the cloudy sky image is perfectly self-illuminated and visible, but it's neither affecting the Spheres or the Suzannes nor the floor Plane (nevertheless slightly visible because of a low intensity World).

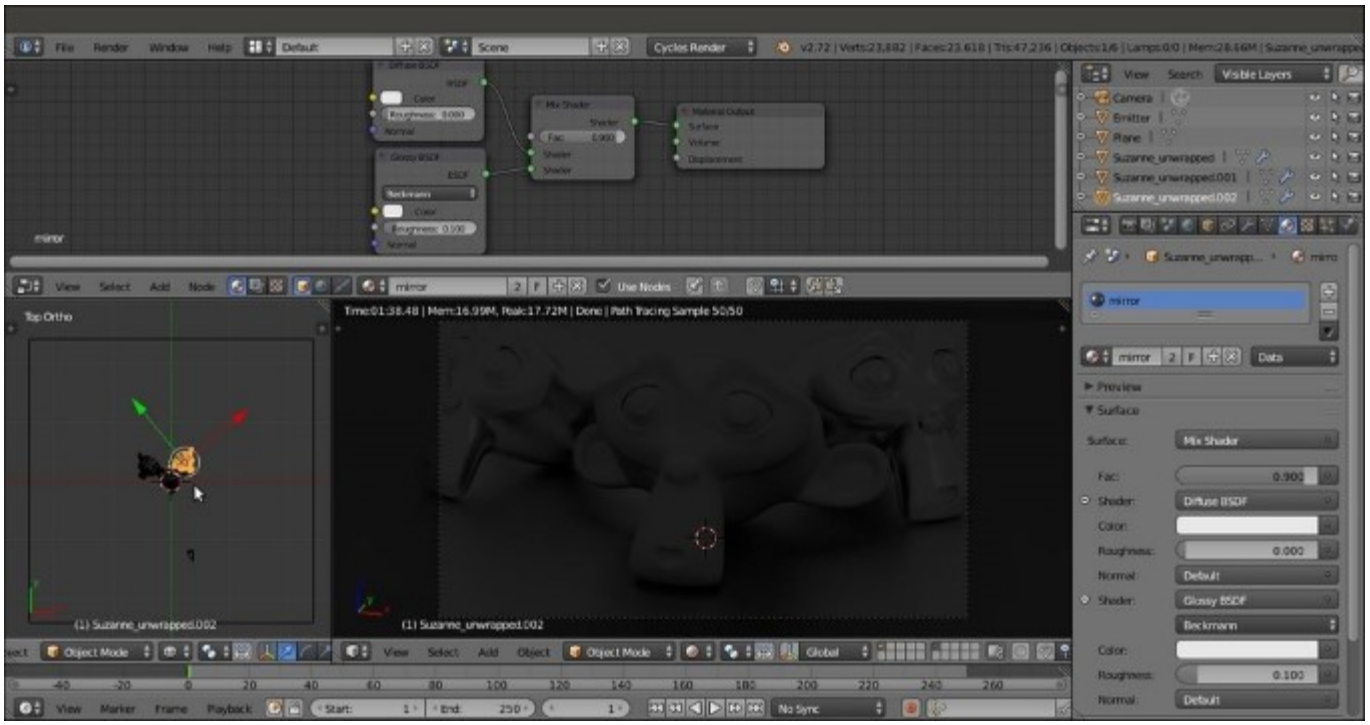
This is the reason a shadeless material is perfect for backdrop elements mapped on Planes (or more often on unwrapped half-spheres called domes) to simulate skies, clouds, and even distant trees. It can also simulate forests and mountains in the background of a scene.

In Blender Internal, obtaining a shadeless material is very simple. Enabling the appropriate item in the material panel is enough. In Cycles, there are two methods to obtain this effect: one based on the material and the other based on the **Ray Visibility** subpanel in the **Object** window, under the main **Properties** panel.

Getting ready

Start Blender and open the `9931OS_09_start.blend` file. Then follow these steps:

1. Go to the **Render** window, and under the **Sampling** subpanel, set the **Samples** to `50` for **Preview** and `100` for **Render**.
2. Set the **Camera** view to the **Rendered** shading mode by pressing *Shift* + *Z* with the mouse arrow in the viewport.
3. Go to the **World** window and set the **Background** strength to `0.200`.
4. Go to **Outliner** and select the **Emitter** object. In the **Node Editor** window, set the **Strength** of the **Emission** shader node to `0.100`.
5. Select the **Plane** object, and in the **Material** window under the main **Properties** panel to the right, replace the **Diffuse BSDF** shader with a **Glossy BSDF** shader node. Switch the **Distribution** of this node from **GGX** to **Beckmann** and set the **Roughness** value to `0.200`.
6. Select the **Suzanne_unwrapped** object and click on the **New** button in the **Node Editor** window toolbar, or in the **Material** window under the main **Properties** panel to the right.
7. With the mouse arrow in the bottom-left corner of the 3D window, press the *7* key in the numeric keypad to switch to the **Top Ortho** view. Press *Shift* + *D* to duplicate the Suzanne mesh, and move it to the left of the scene. Rotate it to accommodate it close to the original mesh. Click on the **2** button to the side of the **Material** datablock in the **Node Editor** window toolbar to make it single-user.
8. In the **Material** window, switch the **Diffuse BSDF** node with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** node, and in the second **Shader** slot, select a **Glossy BSDF** shader node. Set the **Glossy** distribution to **Beckmann**, the **Roughness** value to `0.100`, and the **Fac** value of the **Mix Shader** node to `0.900`. Rename the material as `mirror`.
9. Press *Shift* + *D* to duplicate the `mirror` Suzanne mesh, and move it to the right of the scene, close to the original mesh.

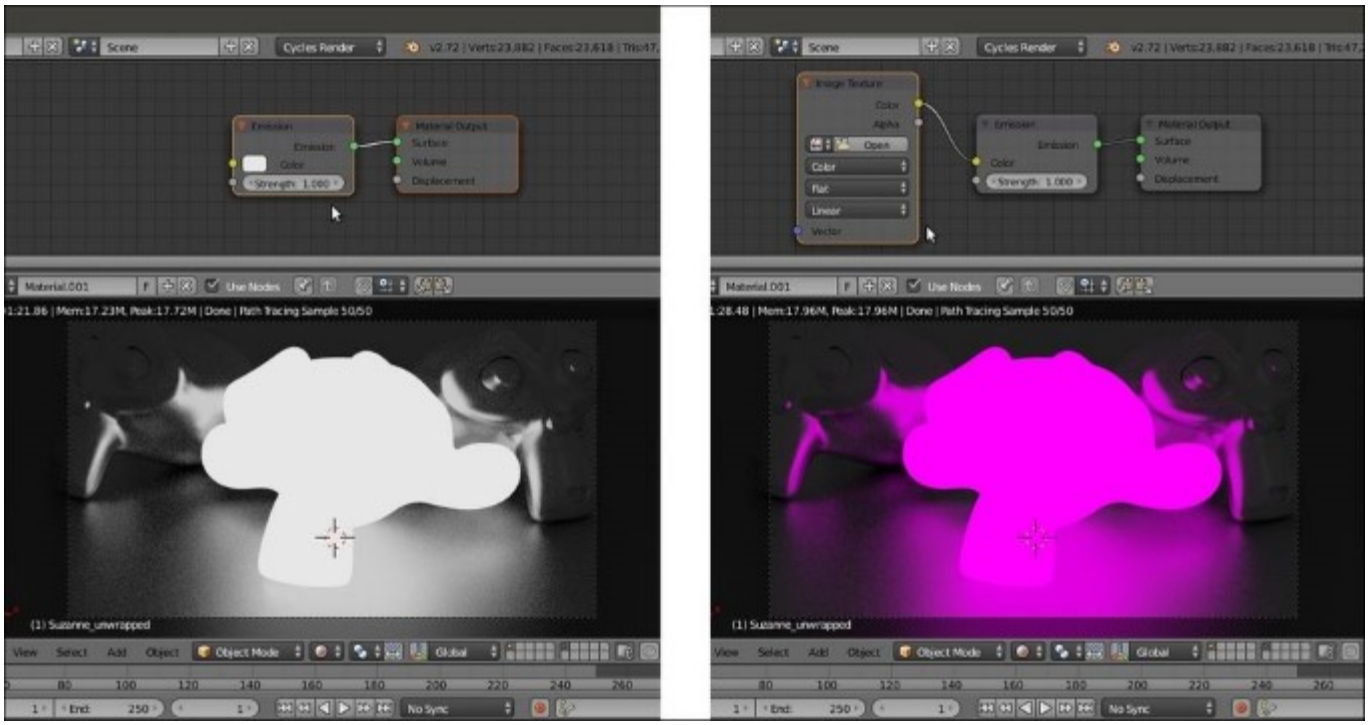


The three Suzannes on the floor Plane in the dark

How to do it...

Let's start with the first method, based on the material:

1. Select the original **Suzanne_unwrapped** mesh, which is in the middle, and click on the **New** button in the **Node Editor** window toolbar. In the **Material** window, switch the **Diffuse BSDF** shader with an **Emission** shader. Add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and connect its **Color** output to the **Color** input socket of the **Emission** node.
2. In the **Rendered Camera** view, the original Suzanne mesh is emitting pink light on the scene. Pink is the default color that Blender uses to tell us that we haven't loaded an image texture yet.

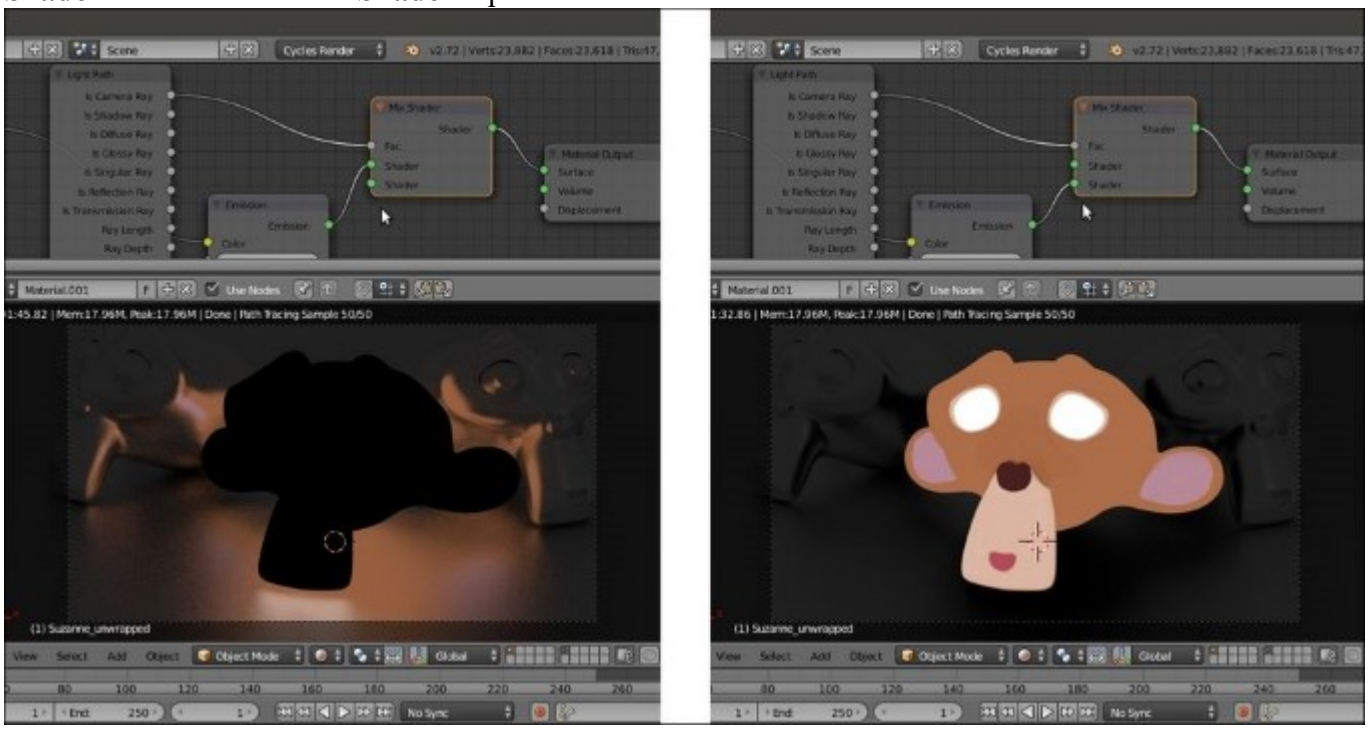


Suzanne emitting light and the texture being missing

3. Click on the **Open** button of the **Image Texture** node, browse to the `textures` folder, and load the `teddybear.png` image (which was used in Chapter 8, *Creating Organic Materials*).

Suzanne with the UV-mapped texture

4. In the **Node Editor** window, add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and paste it between the **Emission** node and the **Material Output** node.
5. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**) and connect its **Is Camera Ray** output to the **Fac** input socket of the **Mix Shader** node. The Suzanne mesh turns totally black (no material) but still lights the scene based on the `teddybear.png` image's values.
6. Switch the connection of the **Emission** node from the first **Shader** input socket of the **Mix Shader** node to the second **Shader** input socket.



Inverting the order of the connections in the Mix Shader node

At this point, the shadeless Suzanne is still affecting the surrounding objects. It's reflected as a black object by the floor and by the `mirror` Suzannes. In fact, the output of the first empty input socket of the **Mix Shader** node is a black color because there is no material at all.

To prevent the shadeless Suzanne from getting reflected, follow these steps:

1. Add a **Transparent** shader node (press *Shift* + *A* and navigate to **Shader** | **Transparent BSDF**) and connect it to the first **Shader** input socket of the **Mix Shader** node.

The totally shadeless Suzanne

2. Rename the material as `shadeless` and save the file as `9931OS_09_shadeless.blend`.

How it works...

Thanks to the **Is Camera Ray** output of the **Light Path** node, all the light rays from the Camera that directly hit the Suzanne mesh are rendered with the **Emission** material brightness (because this material has a value equal to 1, which is due to its connection to the second **Shader** socket of the **Mix Shader** node). For the other kind of rays (reflected, transmitted, and so on, first socket = 0 value) there is no emitting material coming from the Suzanne mesh. Actually, there is initially no material at all, and this gives a black, reflected Suzanne. Following this, to avoid the black reflections, a **Transparent BSDF** shader has been connected to the first socket of the **Mix Shader** node.

There's more...

The second method to obtain a shadeless object is as follows:

1. Starting from the preceding file, select the **Suzanne** mesh, and in the toolbar of the **Node Editor** window, click on the **F** button on the right side of the material name data block to assign a *fake user* (this is to keep the material saved in the blend file even if not assigned to anything). Then click on the **X** icon (to unlink the datablock).
2. Now click on the **New** button to create a new material. In the **Material** window under the **Properties** panel, switch the **Diffuse BSDF** with an **Emission** shader node.

3. In the **Node Editor** window, add an **Image Texture** node (press *Shift* + *A* and navigate to **Texture** | **Image Texture**) and connect its **Color** output to the **Color** input socket of the **Emission** shader.
4. Click on the **Open** button on the **Image Texture** node to load the `teddybear.png` image texture.

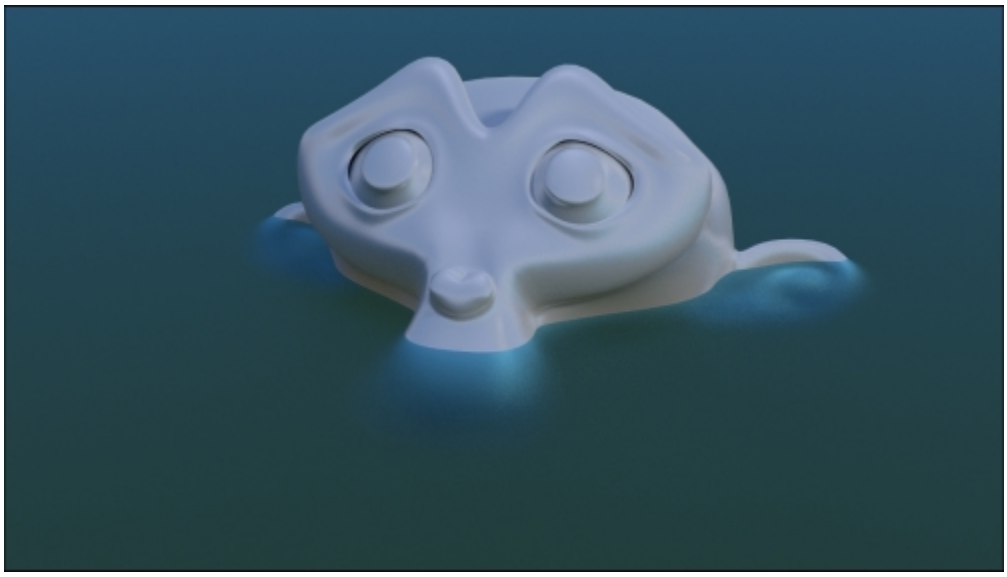
   At this point, we are at the same stage as step 2 of the first method. We have created a light emission material based on the image texture mapped on the Suzanne mesh.
5. Now go to the **Object** window under the main **Properties** panel to the right of the screen. In the **Ray Visibility** subpanel (usually the last at the bottom) uncheck the **Diffuse**, **Glossy**, **Transmission**, **Volume Scatter** and **Shadow** items.

The shadeless Suzanne obtained through the Ray Visibility subpanel

So basically, only the **Camera** item is active now. Simple, quick, and effective!

Creating a fake immersion effect material

In this recipe, we will create a material to give the effect of an object immersed in a substance becoming more and more opaque as the depth increases, for example, murky water.



The murky water effect as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_09_start.blend` file. Then follow these steps:

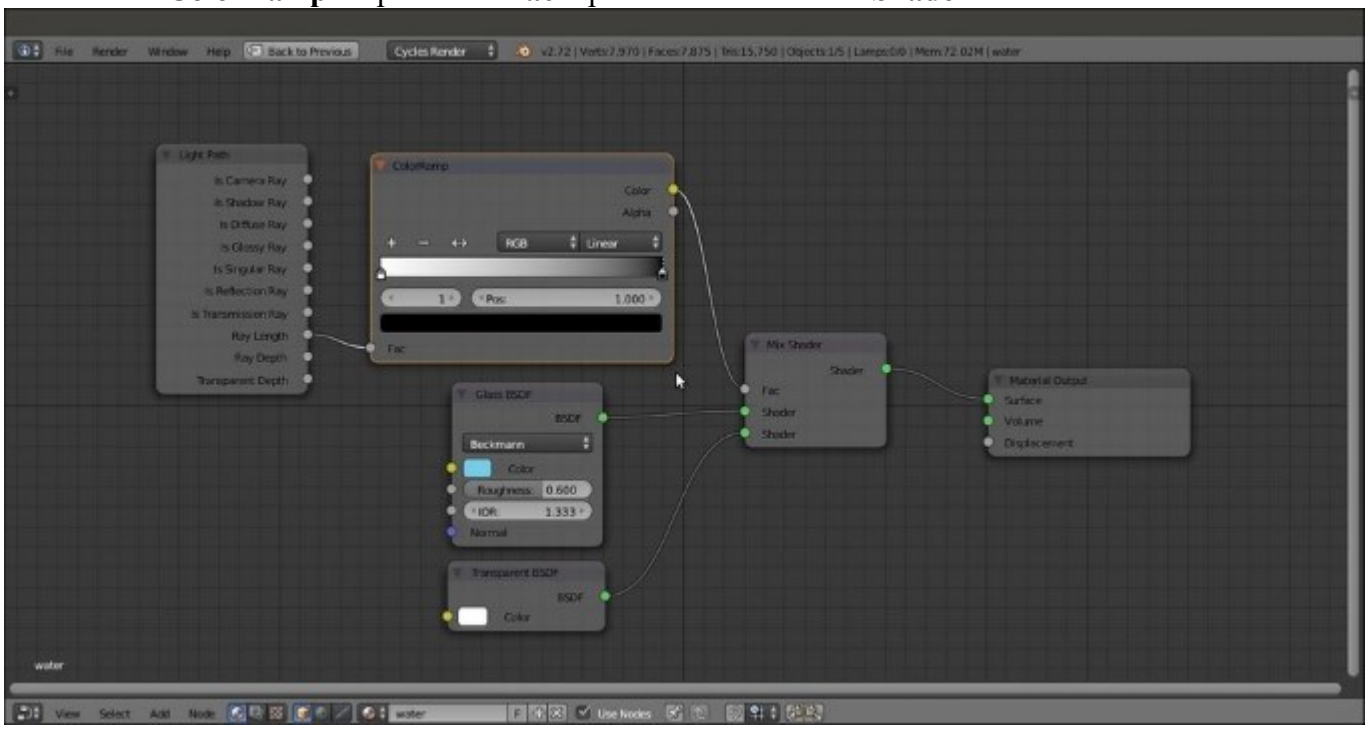
1. Go to the **World** window and click on the little dotted square on the right side of the color slot. From the menu, select **Sky Texture**. Then set the **Strength** value to `0.500`.
2. Select the Plane, rename it as **water**, and move the **Location** value of **Z** to `1.17000`. Then press *Shift* + *D* to duplicate it, rename it as `bed`, and move the **Location** value of **Z** to `-2.00000`.

How to do it...

Let's go ahead and create the different materials:

1. Go to the **Material** window and select the **Suzanne_unwrapped** mesh. In the **Node Editor** window toolbar, click on the **New** button, and rename the material as (simply) `Suzanne`. In the **Material** window under the main **Properties** panel, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** shader node, and in the second **Shader** slot, select a **Glossy BSDF** shader node.
2. Set the **Glossy** distribution to **Beckmann** and the **Roughness** value to `0.100`. Then set the **Fac** value of the **Mix Shader** node to `0.600`.

3. Select the **bed** Plane and click on the **New button** in the **Node Editor** window toolbar. Rename the material as `bed`.
4. In the **Material** window, switch the **Diffuse BSDF** shader with an **Emission** shader node. Set the **Color** values for **R** to `0.800`, **G** to `0.659`, and **B** to `0.264`. Then set the **Strength** value to `0.100`.
5. Select the **water** Plane and click on the **New** button in the **Node Editor** window toolbar. Rename the material as `water`.
6. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Glass BSDF** shader node, and in the second **Shader** slot, select a **Transparent BSDF** node.
7. Set the **Glass BSDF** node's **Roughness** value to `0.600` and the **IOR** to value `1.333`. Then set the **Color** values for **R** to `0.185`, **G** to `0.611`, and **B** to `0.800`.
8. Add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**) and a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**). Connect the **Ray Length** output to the **Fac** input socket of the **ColorRamp** and invert the position of the black-and-white color stops (that is, move the black color stop to the extreme right and the white color stop to the extreme left of the slider).
9. Connect the **ColorRamp** output to the **Fac** input socket of the **Mix Shader** node.



The murky water material network

How it works...

The effect happening on the `water` material is due to the **Ray Length** output, which returns the length of the light rays passing through an object, thus giving the thickness of that object. In our case, the

distance from the water mesh surface to the far distance (from the Camera point of view, because the light rays originate from the Camera).

The gradient of the **ColorRamp** is mapped on the length of this **Ray Length** output (also clamped and inverted by the same **ColorRamp** node), connected to the **Fac** input socket of the **Mix Shader** node in order to work as a stencil map to smoothly blend the amount of the **Glass** shader with the amount of the **Transparent BSDF** shader node.

Thus, the transition from the **Transparent** shader to the **Glass** shader returns the impression of a volume of water becoming murkier as the distance of the object from the surface increases.

Creating a fake volume light material

In this recipe, we will create a material to fake the typical effect of a cone of light visible when passing through the dust suspended in the air, or falling from the sky on a cloudy day (the so-called God's rays) different from the real volumetric effect described in the *There's more* section of the *Using Cycles volume materials* recipe. This is a fake—just a mesh and not a real light to be used for the scene. Therefore, a matching Lamp must be set for the real lighting, as shown in the already made blend file.

The fake volume cone of light as it appears in the final rendering

Getting ready

Start Blender and open the `9931OS_09_volumelight_start.blend` file, where there is a scene set with a ground Plane, a Cube, the volumetric cone mesh (**volume_light**), a spot mesh object (**spot_mesh**), and an effective Spot lamp parented to the **volume_light**. The Spot lamp cone follows the shape of the **volume_light**, and its purpose is to light the Cube leaning on the ground Plane.

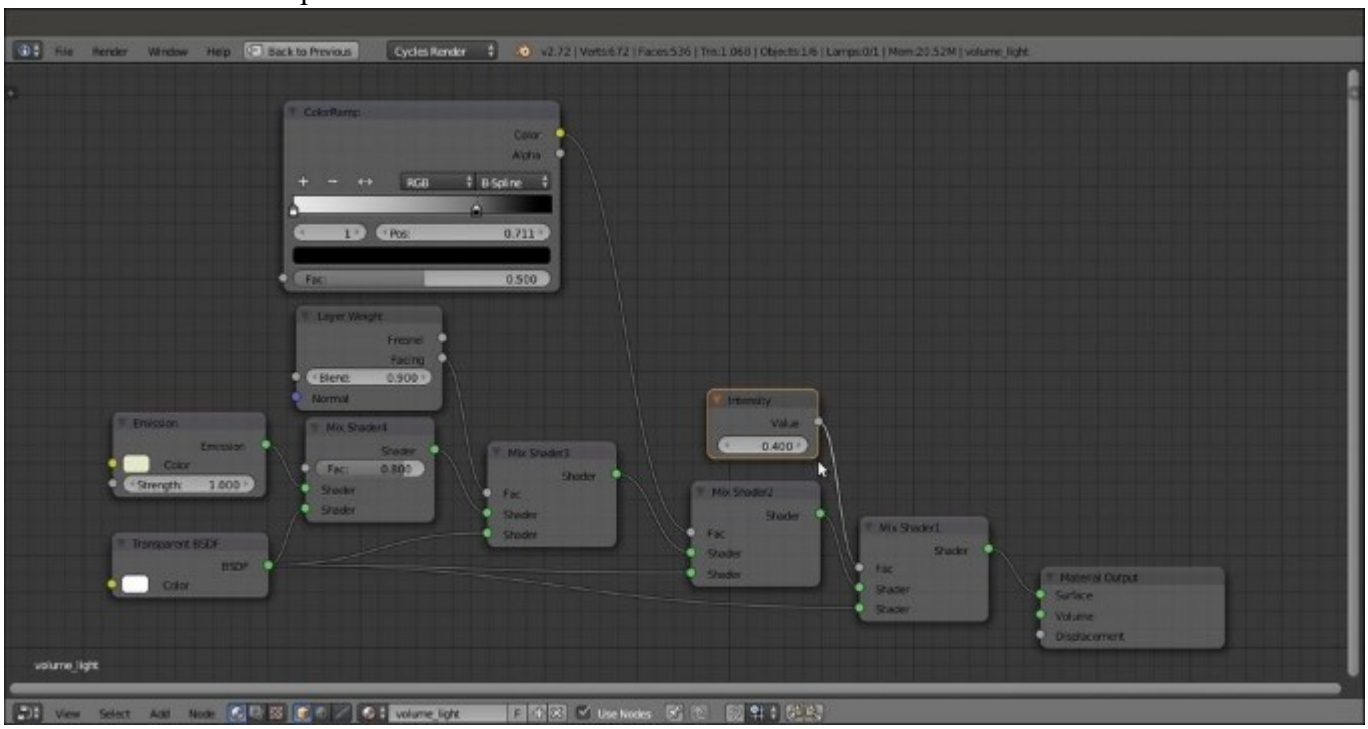
The volume light objects also have a brief animation of 98 frames. To see it, move the Time Cursor inside the **Timeline** window; to point the cone of light to the Cube, go to frame **81**.

How to do it...

Let's go ahead with creating the light cone material:

1. Select the **volume_light** object and click on the **New** button in the **Node Editor** window toolbar, or in the **Material** window under the main **Properties** panel to the right. Rename the material as `volume_light`.

2. In the **Material** window, switch the **Diffuse BSDF** shader node with a **Mix Shader** node. Label it as `Mix Shader1`.
3. In the first **Shader** slot of the **Mix Shader1** node, select a new **Mix Shader** node. Label it as **Mix Shader2**. In the second **Shader** slot, select a **Transparent BSDF** shader node.
4. Go to the **Mix Shader2** node, and in the first **Shader** slot, select a **Mix Shader** node again (**Mix Shader3**). Connect the output of the **Transparent BSDF** node to the second **Shader** slot of the **Mix Shader2** node.
5. Go to the **Mix Shader3** node, and in the first **Shader** slot, select one more **Mix Shader** node (label it `Mix Shader4`). Connect the output of the **Transparent BSDF** node to the second **Shader** slot.
6. Go to the **Mix Shader4** node, and in the first **Shader** slot, select one **Emission** node. Connect the output of the **Transparent BSDF** node to the second **Shader** slot of the **Mix Shader4** node.
7. Set the color values of the **Emission** node for **R** to `0.769`, **G** to `0.800`, and **B** to `0.592`. Then set the **Fac** value of **Mix Shader4** to `0.800`.
8. Add a **Layer Weight** node (press *Shift* + *A* and navigate to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader3** node. Set the **Blend** value to `0.900`.
9. Add a **ColorRamp** node (press *Shift* + *A* and navigate to **Converter** | **ColorRamp**) and connect its color output to the **Fac** input socket of the **Mix Shader2** node. Set the **Interpolation** to **B-Spline** and move the black color stop to position `0`. Then move the white color stop to the extreme left.
10. Add a **Value** node (press *Shift* + *A* and navigate to **Input** | **Value**), label it as `Intensity`, and connect it to the **Fac** input socket of the **Mix Shader1** node. Set the value to `0.400`.



The entire material network

11. Save the file as `9931OS_09_volumelight_final.blend`.

How it works...

The effect of light blending with the night is obtained by the various factors of blending of the **Mix Shader** nodes that cause the mixing of the **Emission** shader with the **Transparent** shader. The purpose of connecting the **Value** node to the **Fac** input of the **Mix Shader1** node is to establish the intensity of the fake volumetric light. A value of `1.000` turns it off completely (be careful not to go beyond `1.000`, otherwise the cone mesh will show up as a dark silhouette). On the contrary, values towards `0.000` (or even negative values) make it appear more and more intense.

Be careful with this simulation because being a mesh emitting light, it can produce strange and unrealistic effects if not carefully planned. Suppose you go to frame **62** and start the rendering. Then you will see that the volumetric cone mesh is intersecting the Cube even in those areas where a real light would create shadows.

See also

Since Cycles has been added to Blender, many artists have posted screenshots and tests for almost every possible kind of material. Especially on the Blender Artists forum, you will find a plethora of data and will discover different (and often better) ways to create the same materials that you have seen in this Cookbook.

Now it's up to you to create new, amazing materials and renderings that no one can avoid staring at. Blend on!

Bibliography

This Learning Path is a blend of content, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Blender 3D By Example By Romain Caudron and Pierre-Armand Nicq*
- *Blender 3D Cookbook By Enrico Valenza*
- *Blender Cycles: Materials and Textures Cookbook - Third Edition By Enrico Valenza*

Index

A

B

C

D

E

H

I

J

K

L

M

N

O

P

S

U