Thèse

THESE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Kévin Jordao
ECOLE DOCTORALE : *MATISSE*
LABORATOIRE : *IRISA / INRIA Rennes*

Interactive design and animation of crowds for large environments

Thèse soutenue le 21.12.2015
devant le jury composé de :

Pr. Bruno Arnaldi
Professeur, INSA de Rennes / Président
Pr. Nuria Pelechano
Professeur Associée Université Polytechnique de Catalogne / Rapporteur
Dr. Ronan Boulic
Maître d'enseignement et de recherche Ecole Polytechnique Fédérale / Rapporteur
Pr. David Cazier
Professeur, Université de Strasbourg / Examinateur
Pr. Marie-Paule Cani
Professeur, Université de Grenoble INP-Ensimag / Directeur de thèse
Dr. Julien Pettré
Chargé de recherche HDR Inria Rennes / Directeur de thèse

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

ueb

THÈSE INSA Rennes

sous le sceau de l'Université Européenne de Bretagne

pour obtenir le grade de

DOCTEUR DE L'INSA DE RENNES

Spécialité : Informatique

présentée par

Kévin Jordao

ÉCOLE DOCTORALE : MATISSE

LABORATOIRE : Irisa / Inria Rennes

Interactive design and animation of crowds for large environments

Thèse soutenue le 21 Décembre 2015

devant le jury composé de :

Pr. Bruno Arnaldi
Professeur, INSA de Rennes, France
/ *Président*

Pr. Nuria Pelechano
Professeur Associée
Université Polytechnique de Catalogne, Barcelone, Espagne
/ *Rapporteuse*

Dr. Ronan Boulic
Maître d'enseignement et de recherche
École Polytechnique Fédérale, Lausanne, Suisse
/ *Rapporteur*

Pr. David Cazier
Professeur, Université de Strasbourg, France
/ *Examinateur*

Pr. Marie-Paule Cani
Professeur, Université de Grenoble INP-Ensimag, France
/ *Directrice de thèse*

Dr. Julien Pettré
Chargé de recherche HDR Inria, Rennes, France
/ *Co-directeur de thèse*

À Ma famille, mes amis..

Remerciements

Je tiens à remercier les membres de mon jury pour avoir examiné mon manuscrit, et pour la rapidité à la laquelle ils m'ont retourné leurs rapports. Je les remercie également de l'effort qu'ils ont fait pour le déplacement jusqu'à Rennes, et d'avoir accepté une date de soutenance qui généralement est une période de fête en famille.

Merci aussi à mes directeurs de thèse qui m'ont encadrés tout au long de ces 3 ans. Ils m'ont beaucoup aidé à progresser dans le milieu de la recherche, que ce soit en rédaction, sur la présentation de mes travaux, ou bien sur mes directions de recherche. Je tiens particulierement à souligner la disponibilité de Marie Paule Cani malgrés sont emploie du temps surchargé.

J'aimerais spécialement remercier les ingénieurs de recherche qui ont travaillé avec moi pendant ma thèse. A savoir, Orianne Siret et Tristan Lebouffant. Ils m'ont été d'une aide précieuse pour mettre en valeur mes resultats. Et d'un soutien sans équivoque lors des longues nuits à finaliser les projets.

Merci à mon collègue Panayiotis Charalambous, avec qui j'ai a beaucoup travaillé sur ma dernière contribution. Il m'a beucoup aidé concernant les répétitions de présentations d'artcles ou de ma thèse, sur l'anglais, la redaction et ma motivation en général.

Merci à mes collègues pour les délires et les discussions. Je pense notament à Orianne, Quentin, Camille, Kim, Tristan, Diane, Huyen, Guillaume, François, Julian, Ferran, Fabien, Andéol, Merwan, Adrien... Avec qui j'ai passé de très bon moments et qui a participé au fait que mes 3 ans de thèse ce soit déroulé dans de superbes conditions.

J'aimerais aussi remercier mon professeur de wushu, Bertrand Denolle, qui m'a

transmit une autre passion en dehors de l'informatique : les arts martiaux chinois. Dans les moment intenses de ma thèse les cours de Bertrand me permettaient de me vider completement la tête et de recharcger les batteries pour pouvoir repartir de plus belle.

J'aimerais aussi remercier mes amis en général. Dont certain qui ont fait l'aller retour depuis Paris pour venir m'encourager lors de ma soutenance. J'aimerais plus particulierement remercier Laurent Noel, qui a su me transmettre sa passion pour l'informatique. Sans lui je ne serai pas à ce niveau d'études aujourd'hui.

J'aimerais finir ces remerciements en mettant à l'honneur ma famille. Durant ces années d'études, elle n'a eu de cesse de me soutenir et de m'encourager. Je pense particulierement à ma mere et à ma grand mere, qui malgrés quelques périodes difficiles durant ma scolarité, n'ont jamais douté de moi et toujours su me donner l'élant necessaire pour aller plus loin, pour aller plus haut. Merci.

Contents

Chapter 1

Introduction

1.1 Context

Research in crowd animation became very active this last decades, because of the success the video games, movies, and of the exploration of 3D environments. Crowds play and important role in these fields.

In movies, background crowd animation, such as chatting or walking pedestrians, allows to dive main actors into a world full of life. The movie *Astérix et Obélix : au service de sa majesté* or the tv-show *Game of Thrones* use a crowd software *Golaem Crowd* [Gol, n.d.], to display crowds of rugby's fan or a population of panicked people trying to escape a dragon, as in Figure 1.1. Crowd animation is also very popular for battle scenes, for instance in the movie *The Lord of the Ring: The Return of the King*, when Minas Tirith is attacked by hordes of orcs. Background or battle scene crowds create a unique atmosphere that increases the immersion of the viewer. Consequently, the crowd should have appropriate behaviour according to the scenario, in terms of motion and local interaction, to not distract the audience.

In video games, it is now common to have entire cities populated by virtual characters. For instance in the video game *Assassin's Creed Unity* [Ass, n.d.], the player is able to explore the whole city of Paris in 1789, during the french revolution, with streets flooded of demonstrators (Figure 1.2). The crowd of this

Figure 1.1: Two crowd simulation computed using *Golaem* software. At the top, a crowd in the tribune for the movie *Astérix et Obélix : au service de sa majesté*. At the bottom, a crowd fleeing the dragon for the TV-show *Game of Thrones*.

Figure 1.2: A picture from the video game Assassin's Creed Unity. Here the main character observes a large crowd in a street of Paris in 1789.

game allows a great immersion in the french revolution. Additionally in some strategy games such as *Starcraft II* or *Age of Empire Online* [Sta, n.d.; Age, n.d.], the player can directly control armies to attack the opponents. A good control of crowd units is crucial for the game-play.

A possible evolution in the architecture and tourism fields could be to integrate crowds in virtual 3D cities. In the case of urban projects, mock-up can go with virtual characters to have a better feeling of the future district or city. In the case of a virtual tour of a city, crowds could allow to feel the atmosphere of the different districts, or to get which places are busy or calm. We can imagine a future extension of famous world explorers, such as *Google Earth* [Goo, n.d.], integrating crowds to populate cities over all continents. The kind of crowds we expect for these fields, are crowds animated during long periods of time, to be able to have a preview of the urban life at any moment of the day. Moreover, the size of the crowd is to be very large as well, in order to enable the user to explore whole cities if he or she wishes to do so.

Whether in computer games, movies, or 3D environment exploration, crowds

have a huge impact on the final media proposed by these fields. However, it remains hard to create crowds with specific behaviour and motion, with current techniques. The main way to create crowds is to use a crowd simulation algorithm, a black box taking into account user defined parameters to produce a crowd motion. The main issue of these methods is the lack of intuitive and direct control for the motion or the visual aspect of the crowd. Existing approaches consist of tuning the input parameters of the simulation to alter the crowd, instead of directly expressing what we need for the crowd. This implies that an expert designer is required because of the complexity of the methods and the time it takes to create the desire output. Producing large, good quality scenes for industrial applications is thus very costly.

1.2 Problem statement

The process of interactively designing a virtual population, even for medium-scale environments, such as a block of buildings, is not an easy task and requires addressing two main challenges. Firstly, a crowd is an intrinsically complex system, containing characters that interact both at local and global scales, have individual or group behaviors and perform actions in close relation to their environment. The creation of realistic and complex crowds therefore requires the use of complex simulation or animation models for which the user needs to specify a large set of parameters through a series of, generally tedious, generate and test loops. Secondly, the required degree of control to easily and interactively populate such large environments requires the design of novel tools offering both spatial manipulation features (where to position the crowds, how to control flows or densities) and temporal manipulation features (how a crowd can change over time in terms of behavior, flow, and density).

Techniques based on crowd simulation only offer an indirect and generally global control on a crowd through the manipulation of agent parameters, and without immediate feedback [Ulicny *et al.*, 2004; Patil *et al.*, 2011]. These kind of techniques are not effective to define low-level subtleties in the crowd movement, such as the path of a character or its shape in time. Nevertheless, these kind

of control is essential for conveying individuality to the crowd or for allowing a player to achieve specific task in a video game.

Another approach would consist in directly manipulating agents trajectories [Kwon *et al.*, 2008; Kim *et al.*, 2009], i.e. bending, stretching and shortening them to fit both user requirements and the topological constraints of the environment to populate. However, large deformations may lead to unrealistic results. Additionally, when the crowd becomes large, many operations have to be done by the designer to reach the required output. Moreover, these methods become slower as the number of character increases, as collision between them should be avoided.

1.3 Contributions

The goal of this thesis is to develop methods enabling a user to design crowds while providing a direct control of the visual aspect and the local and global behaviour of characters, through interactive artistic-like applications. These techniques should be able to handle crowds of very large size with an endless animation. For this last point, we know that crowd patches [Yersin *et al.*, 2009], pre-computed pieces of crowd animation that can be connected together to shape a population, is an effective method to produce large crowd with an infinite animation. In contrast, this technique suffers from the lack of interactivity to locate and connect patches together, and from the lack of control to create a set of crowd patches with good properties.

In this thesis, we propose three methods to control large crowds through interactive applications or artistic-like interfaces.

- **Crowd sculpting**: a novel approach to interactively design complex animated crowds for virtual environments, with high level gestural control and immediate visual feedback. The generated animations can be endlessly looped in time, thanks to the crowd patches method we rely on.

- **Crowd art**: an optimization technique to compute crowd animations that satisfy localized density and directional flow constraints. The process is

able to avoid local loops in characters' trajectories by computing paths linking crowds patches with the biggest errors. The method provides an artist-driven tool for designing crowds. Designers can create crowds very quickly using an existing paint tool. User requirements are specified by combining image layers which specify dynamic and static density, direction and obstacles.

- **Temporally varying crowds**: an extension of crowd patches that improves motion variety, thanks to our new temporal permutation mechanism. Given that patches capture periodic animations, we propose a technique to permute different version of a crowd patch over periods of time. Temporally extended motions inside a patch therefore present a more diversified content.

1.4 Organization of this document

This thesis is organized as follow. Chapter 2 is dedicated to previous work on crowd simulation, crowd control, and 3D design through artistic metaphors. Advantage and drawbacks of each methods are discussed, and a conclusion is formulated at the end to focus on what has to be done on these fields.

All the methods developped during this thesis are based on the crowd patches paradigm. A full descritption of crowd patches is done in Chapter 3. Unpublished trajectories generation techniques, dedicated to crowd patches, are also described.

Then in Chapter 4, we tackle the spatial modeling of crowds. We present an interactive method to easily populate environments through intuitive gestures, able to shape a uniform crowd in a 3D environment.

In Chapter 5, we present a method able to create a huge crowd under stationnary density and flow constraints. Density and flow constraints are set through a paint-like interface to ease control.

Next, in Chapter 6, we present different methods to have motion variety over time. First, we describe how to avoid repetition of animations in a crowd patch. Then we present a possible extension where crowd patches become reactive to

user actions and to changes in the environment.

Finally, in Chapter 7, we formulate a conclusion of this work, and discuss limitations and possible future work driven by this research.

1.5 Publications

1.5.1 Technical papers

TP1. **K. Jordao**, P. Charalambous, J. Pettré, M. Christie, M. P. Cani. "Crowd Art: Density and Flow Based Crowd Motion Design". In Proceedings of *Motion In Games* (2015). ACM.

TP2. **K. Jordao**, J. Pettré, M. Christie, M. P. Cani. "Crowd Sculpting: A space-time sculpting method for populating virtual environments". In *Computer Graphics Forum* (2014), Volume 33, Number 2.

1.5.2 Posters

P1. **K. Jordao**, J. Pettré, M. P. Cani. "Density-controlled crowds". In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2014).

1.5.3 Short papers

SP1. **K. Jordao**, J. Pettré, M. P. Cani. "Interactive techniques for populating large virtual cities". In Proceedings of the *Eurographics Workshop on Urban Data Modelling and Visualisation* (2013) pp. 41-42. Eurographics Association.

Chapter 2

Related work on crowd animation design and expressive modeling

Contents

This chapter reviews the different methods for synthesizing the movement of crowds. In Section 2.1, we give an overview of simulation systems, followed by a discussion of advantages and drawbacks of these methods. Then, techniques able to design crowds through direct motion or intrinsic parameter control are exposed in Section 2.2. To inspire possible interactive interfaces to design crowds, work on expressive modeling is presented in Section 2.3. We end this state of the art by a conclusion in Section 2.4.

2.1 Crowd Simulation

Simulation can be divided in two main categories, microscopic simulation, macroscopic simulation. The first one, which focuses on individual interactions and local behaviour, is developed in Section 2.1.1. The second one, which sees the crowd as a whole and where individuals are implicitly represented through local density on a grid, is explained in Section 2.1.2. Then a summary and a discussion of the different approaches is given in Section 2.1.3.

2.1.1 Microscopic Simulation

Microscopic simulation sees the crowd as a set of autonomous agents, which are able to follow a goal or a leader, and have their own internal properties. This approach focuses on local interactions between individuals. Each member of the crowd takes into account other individuals' actions, and uses nearby information to move through the environment. The main challenge of this approach is to avoid agent/agent, and agent/environment collisions; while keeping realistic behaviour and good performance.

This section reviews the main contributions of microscopic simulation. The reader can find a description of this topic in the book of Thalmann & Musse [2007].

2.1.1.1 Rule-based and Path Planning Approaches

In rule-based approaches, agents are governed by rules, which define the right behaviour according to an individual's situation. The first example of this approach is [Reynolds, 1987]. Note that the concern of the author is group behaviour of animals. He defined a set of basic rules for each agent, such as separation, cohesion and alignment, to simulate flocks of birds, herds of land animals or schools of fish. He extended his model to steering behaviour in [Reynolds, 1999], with rules to simulate the motion of pedestrians, such as collision avoidance, leader following, seeking and fleeing, or group cohesion. Musse & Thalmann [1997] proposed a model where rules are based on concepts from sociology, such as polarization or domination.

Other techniques couple path planning algorithms with rule-based methods to create crowds with more plausible behaviours. For instance, [Bayazit *et al.*, 2003] combined the microscopic model of Reynolds [1987] with global information represented as roadmaps of the environment to enable more sophisticated flocking behaviours. In the work of Lamarche & Donikian [2004], authors use a Delaunay triangulation to define walkable areas, and detect bottlenecks inside the environment. This is used to allow a fast path finding and an efficient reactive navigation algorithm for crowds. Pettré *et al.* [2005] divide the environment into navigation corridors, giving rise to a navigation graph, which allows to find path for individuals in the crowds. Authors use steering behaviours [Reynolds, 1999] to drive simulated characters of the simulation. This method is extended [Pettré *et al.*, 2006] to achieve a scalable simulation enabling to display up to 35000 pedestrians. Shao & Terzopoulos [2007] using a hierarchical environmental modeling framework to efficiently synthesizes numerous self-animated pedestrians performing a rich variety of activities in a large-scale indoor urban environment.

A major issue with rule-based approaches is to find adapted rules able to generate the expected behaviour at the crowd level. This is particularly the case in constrained environments such as cities or indoor buildings. Moreover, these approaches require a large amount of parameters tuning to achieve the desired results.

2.1.1.2 Force-based Approaches

In contrast with steering behaviours that directly modify a character's instantaneous velocity, the set of rules and their weights, Force-based approaches apply external forces to crowd characters as if they were mass particles. In the manner of the second law of *Newton* $\sum_i \vec{F_i} = m\vec{a}$, the sum of these forces is equal to the acceleration, which make individuals move and avoid collisions. Helbing & Molnár [1995] propose the first example of a force-based approach. They explain how agents can undergo social forces, by receiving it from other agents, the environment, and goals. Mousaïd and colleagues have used the social force model to replicate observations from a series of controlled experiments [Moussaïd *et al.*, 2009]. Helbing *et al.* [2000] extended Mousaïd's model to simulate emergency panic situations. Braun use the social forces model to simulate the formation of groups in a crowd using additional attractive forces [Braun *et al.*, 2003]. Heigeas use an approach close to the force-based techniques, based on a mass-spring-damper system, to model collective crowd movement [Heigeas *et al.*, 2003].

A main issue of force-based methods is the lack of prediction, because the virtual characters only interact with other agents when they are sufficiently close. Techniques based on the geometrical information of the environment are introduced to increase individuals anticipation and consequently solve force-based approaches' main issue. These methods are discussed in the following section.

2.1.1.3 Geometric Approaches

Geometric approaches consider the local area occupied by each agent to select a future velocity to produce collision free motion. These approaches are inspired from robotics, where they are initially designed to handle incomplete or noisy information with regards to the robot's local surroundings.

Fiorini & Shiller [1998] developed a method based on the *Velocity Obstacles* (VO) for collision avoidance between multiple robots. In this work, objects and characters are seen as obstacles in velocity space. This velocity space represents for each agent a set of velocities that would cause a collision at some moment in time with another agent. The union of VOs from any number of obstacles

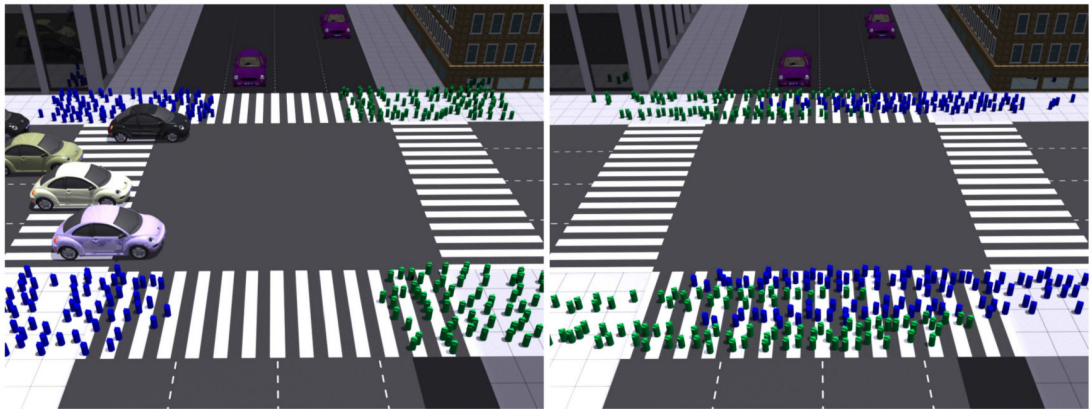

Figure 2.1: A pedestrian crosswalk modeled with RVO [van den Berg *et al.*, 2008b]. 400 agents distributed near the corners of the crossroads move to the other side of the street. The opposite flows of agents automatically form lanes, as they cross each other.

constitutes the set of unsafe velocities for an agent. The idea is to choose a velocity from the admissible component of the velocity space to avoid collisions between agents. Still based on VO, van den Berg *et al.* [2008a,b] introduced the Reciprocal Velocity Obstacles (RVO). An example of collision avoidance done by RVO is shown in Figure 2.1. The main contribution of this technique is to avoid oscillation issues of VO. Optimization on performance is introduced by Guy *et al.* [2009] thanks to a highly-parallel algorithm that uses a discrete optimization method. van der Berg *et al.* [2011] extend this method for n-body collision avoidance. This is later improved by Guy *et al.* [2010] to better match for simulation of virtual human. Other techniques consist in predicting the trajectories of the other agents to avoid collision, as proposed in [Paris *et al.*, 2007]. In a similar manner, Pettré *et al.* [2009], proposed a model elaborated from experimental interaction data involving multiple characters in crossing scenarios. Authors demonstrate that the minimum predicted distance is an adequate criterion to determine whether humans require to adapt their trajectory or not. In the work of Ondřej *et al.* [2010], collision avoidance behavior is handled as a visual-stimuli/motor-response control law. Lemercier *et al.* [2012] propose a following behaviour model based

on experimental studies.

Geometric approaches are a good trade-off between quality of individual trajectories and the number of supported agents in real-time, which now goes up to several thousands. However, the local scale resolution of these methods does not allow to solve bottleneck situations, as is often the case in dense crowds or highly constrained environments.

2.1.1.4 Data-driven Approaches

Instead of using previous simulation models to synthesize crowds, data-driven approaches use precomputed data of motion or behaviours, extracted from videos or from motion capture data [Metoyer & Hodgins, 2004], to govern the behaviour of individual characters in a crowd. Lerner *et al.* [2007] solve interactions between agents of the crowd by matching examples from the database with the current situation. A method proposed by Lee *et al.* [2007], creates an agent model which learns from real human trajectories. The model decides actions of agents based on the environment and on the motion of nearby agents in the crowd. Another work based on data [Ju *et al.*, 2010], proposes to blend existing crowd data to generate a new crowd animation with an arbitrary number of characters and duration. Ahn *et al.* [2012] propose the Trajectory Variant Shift (TVS) method, based on real pedestrian trajectories re-use. The mehod re-use and shift these trajectories to avoid collisions while retaining the liveliness of captured data. A more recent technique have been developed by Charalambous & Chrysanthou [2014]. This work is based on a data structure: the perception-action graph, which stores actions and states from the database. The graph deduce characters actions from what they perceive. Compared to [Lerner *et al.*, 2007], this method is twenty times faster.

Data-driven approaches are able to handle more varied and realistic situations than rule-based and force-based approaches. However, to produce good quality simulations, a huge database of various motions is required. Such a database remains very long and expensive to create. Moreover, this large amount of data raises two issues. Firstly, the analysis and processing of this data cannot be

performed at runtime because computations are too costly. Secondly, the access to such amounts of resources is too slow for real-time applications. therefore, these methods are usually used for crowds of less than one hundred characters.

2.1.2 Macroscopic Simulation

Macroscopic simulation focuses on the global behaviour of a crowd, in opposition to microscopic simulation which focuses on individual behaviours. Computation of the crowd behavior and motion is done as if the crowd were made of some continuous material, using for instance fluid simulation techniques. Characters are then rendered as particles obeying the computed velocity field. Consequently, macroscopic models are very efficient and are able to handle hundreds of thousands of virtual characters. This can be used to simulate large group phenomena such as evacuation of a building or a battlefield scene.

Many methods are based on fluid mechanics. The first work introducing this approach is from Henderson [1971, 1974]. The method consists in simulating a flow of people along a channel, based on gas dynamic equations. However, this method only works in low density situations. Helbing [1998] proposes a gas-kinetic model for crowds as an extension to the work by Henderson [1974], able to handle other denser crowd situation such as, walking lanes and pedestrian jams, the propagation of waves, and the behavior of people on a dance floor.

Other techniques are based on a particle representation of agents to describe the global motion of the crowd. Chenney [2004] proposes a flow tiles method that shapes a continuum velocity field in the environment. These flow tiles drive agent-particles in the environment. Treuille *et al.* [2006] propose to create a continuum potential fields in the environment to drive individuals. The potential field is obtained by combining several parameters from areas in the environment, such as maximum speed, density, direction, obstacles, etc... Crowds are animated by using the potential field to drive individuals. Narain *et al.* [2009] introduce a variational constraint called unilateral incompressibility. This allows crowds to be compressible or incompressible, depending on their density. Moreover, the maximum volume of the crowd is limited. The way the crowd motion is computed

(a) (b) (c)

Figure 2.2: Aggregate Dynamics for Dense Crowd Simulation [Narain *et al.*, 2009]. Examples of large, dense crowds simulated with Narain's technique. (a) 100,000 pilgrims moving through a campsite. (b) 80,000 people on a trade show floor. (c) 25,000 pilgrims with heterogeneous goals in a mosque.

depend on the aggregation of individuals. Very large and dense crowds can be simulated with this technique, as illustrated in Figure 2.2.

Macroscopic approaches are well adapted to simulate crowds of thousand of characters in real time, thanks to their low consumption in term of computational resources. However, these approaches do not allow to obtained rich individual behaviours, because the simulation is only handled at a global scale. These methods are preferred for large crowd scenes where individual behaviour of the agents is less important than the overall motion of the crowd.

2.1.3 Summary

Simulation offers many ways to synthesize crowd motion. These techniques can be classified into two main approaches:

- **Microscopic simulation**, which takes into account the behaviour of individuals to make the crowd motion emerge. These methods allow to simulate a large variety of situations, using simple rules based on collision avoidance, behavioural factors, and example-based local decision making. However these techniques suffer from their parameter-tuning phase to achieve the expected crowd motions. Moreover, results rarely exceed crowds composed of more than several thousand characters, because of the per-agent process-

ing.

- **Macroscopic simulation**, which gives good global behaviours for large and dense crowds composed of thousand of characters, even for real time applications. However, these techniques do not capture any individuality in the behaviour of the agents.

Simulation approaches allow to create realistic crowds. If an artist wants to design a large battlefield, he or she will prefer macroscopic models. In contrast, microscopic models will be preferred if he or she wants to create a train station populated with moving characters. However, in the context of animation for movies or games, crowd designers have a clear picture of the final output they want. Using a simulation system to obtain the desired behaviour is then time consuming and non intuitive. The user only has an indirect control on the final animation, by manipulating parameters of the simulation instead of the content of the animation. Moreover, simulation techniques are hard to understand for a beginner, because the underlying models are based on complex mathematical or physical properties or complex data structures. Consequently, the user has to try a first set of parameters to see if the resulted animation is satisfying, and repeat the process many times until the result eventually matches the desired goals.

The next section describes methods for improving user control over crowd animations.

2.2 Crowd Animation Design and Re-use

The purpose of crowd animation design methods is to give some more direct control on the crowd animation to the user, and prevent trial and errors process inherent to simulation methods. Crowd design is performed, by using sketch-based interfaces, or manipulation of velocity fields etc. . . , to set in an intuitive way parameters of the underlying simulation or to manipulate the crowd motion. Other techniques such as *Crowd Patches* [Yersin *et al.*, 2009], allow to define repeatable local animations in space-time that can be connected to each other. The problem of populating a scene with virtual humans changes from tuning

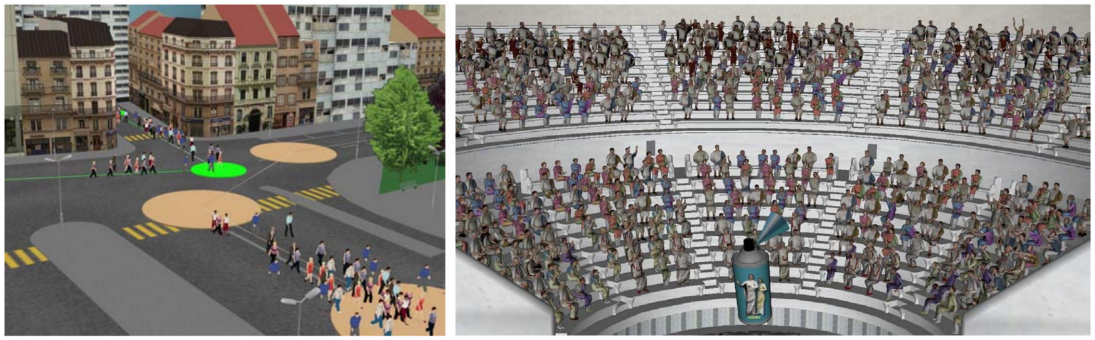

Figure 2.3: Crowd Brush [Ulicny *et al.*, 2004]. **Left**: Crowd following paths defined by a "path brush". **Right**: A crowd in a stadium with behaviour defined by the brush in the center of the picture.

parameters of a simulator system to assemble a puzzle. These different approaches are described in the following section.

2.2.1 High-Level Control on Simulation System

In Section 2.1, we listed the drawbacks of indirect control of simulation systems. The main issue of these models is the parameters tuning phase. Some techniques simplify this phase thanks to a user interface able the set parameters for the desired simulation. For instance, Ulicny *et al.* [2004] introduce *CrowdBrush*, a paint interface for designing crowds. The user is able to use different brushes to create agents, to apply different behaviour such as running or walking, or color of characters. Then an underlying simulation system is used to follow the inputs of the user. *CrowdBrush* allows the creation of a populated scene in few minutes (See Figure 2.3). Chenney [2004] provides an interface to design global crowd motion by editing the velocity field of the scene. An other method consist in sketching shapes to control the group formations ans transition between differents shapes Gu & Deng [2013]. However, these techniques do not allow to define goals for the agents. In contrast, Jin *et al.* [2008] propose an intuitive way for authoring crowd scenes, by allowing the user to drive the flow of crowds by sketching desired velocities out of anchor points in the scene. In a similar manner, the work of [Patil

et al., 2011] consists of drawing strokes in the environment to define navigation fields for agents. They are driven by user defined directions to avoid bottleneck situations or achieve certain desired behaviours.

In all these methods, parameter setting of the simulation is performed through a user interface. Consequently, the user specification stage is much easier. Despite the control on parameters on simulation, there is no direct control on the animation results. As simulation methods, it is still time consuming and non intuitive to obtain the desired crowd behaviour.

2.2.2 Motion Editing

In contrast with the simulation approach, the motion editing approach try to ease the design and the creation of a crowd. Based on existing data of crowd, obtained from real data or simulation, users are able to create new content by editing existing motions or by assembling different motions together.

In the group motion editing technique [Kwon *et al.*, 2008], the user is able to edit precomputed motion of group of people, while preserving neighborhood formation and individual moving trajectories. Several operations are possible: bending, stretching, shrinking and merging. It becomes easy to populate an environment by deforming and moving the motion in the scene, as shown in Figure 2.4. However large deformation of a motion leads to unnatural speed of characters. Moreover, this technique cannot handle thousands characters at interactive rate, because of the expensive time of computation for deforming a crowd motion.

This method can only deform walking behaviour. The problem is more complex if the motion is linked with interactions between agents, such as carrying boxes from point A to point B. Interactions have to be preserved while deformations are applied. This is what the paper [Kim *et al.*, 2009] solved. But again this method is not suited for very large crowds.

This works is extended in [Kim *et al.*, 2012]. The system is able to connect piece of animations together to cover the entire environment. Deformations can be applied on it while preserving interactions between agents. The covering process

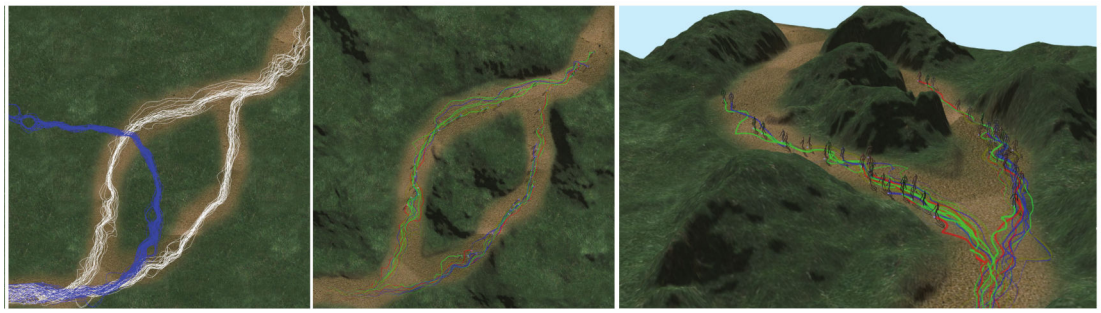

Figure 2.4: Group motion editing [Kwon *et al.*, 2008]. **Left**: In blue the original group motion, in white the edited group motion based on the blue one. **Middle**: A top view of the final output. **Right**: A back view of the final output.

is automatic, so user have less control on the final scene. However, this approach can handle much more characters than other techniques [Kwon *et al.*, 2008; Kim *et al.*, 2009].

A final method [Kim *et al.*, 2014], combine both huge crowd manipulation and interaction preservation. Thanks to a smart selection by cage, user can modify part of motion in space and or in time to fit with the constraint environment. The manipulation is able in real time.

All these techniques allow to manipulate group of people more or less large. However these systems are based on motion limited in time, so creating an endless animation of a crowd is impossible with these approaches.

2.2.3 Texture based approach

Other approaches for populating environments consist in covering the environment by animations encapsulated in patches. These patches are precomputed and used to represent short segments of motion in small areas that can be copied and concatenated at run time to produce larger scenes. Populated an environment becomes equivalent to applying 4D texture on it. Yersin *et al.* [2009] introduced *Crowd Patch*, a method able to create and connect pieces of periodic animation of several moving characters. A *Crowd Patches* is defined by the following items.

Figure 2.5: Two urban environments populated using the *Crowd Patches* technique [Yersin *et al.*, 2009]. For each environment a picture with and without displaying trajectories and borders of crowd patches are provided.

- Patterns, which are a set of spatio-temporal way-points on a boundary of a *Crowd Patch*. They play the role of gates where characters can enter and exist from the *Crowd Patch*. Two crowd patches can be connected together if one of the two patch share a symmetric pattern (called *mirror pattern*) of the second.

- Obstacles, which characters have to avoid. Only small objects of the scene are represented in the *Crowd Patch*, such as benches or trashes.

- Characters, which are divided into two groups.

  - *endogenous*, which stay inside the patch.
  - *exogenous*, which enter and exit the patch and are constrainted by the spatio-temporal way points of the patterns.

  Note that to ensure the continuity of the animation at the end of the patch period, characters' trajectory have to end where another character's trajectory begins. These specific constraints are defined as inner spatio-temporal way points.

Populating an environment is just a matter of correctly tiling the environment into a set of *Crowd Patches*, as demonstrate in Figure 2.5. Moreover crowd patches have several interesting properties. Because they are periodic, the resulting crowd animation can be play endlessly. Also, trajectories of characters are

precomputed, so there is no extra computational cost at runtime, contrary to the simulation approaches. A crowd of thousands characters can be easily modeled using a 3D rendering and animation engine.

Note that this technique is inspired from the work of Lee *et al.* [2006]. Authors introduced the *Motion Patches*, which are animations limited by a single character interacting with objects or environment and encapsulated into square area.

Creating periodic motions for crowd patches remains challenging. Characters of a crowd patch are constrained by spatio-temporal way-points which they have to meet to ensure the periodicity of the motion. A first method [Yersin *et al.*, 2009] consists in using the social forces approach [Helbing & Molnár, 1995] constrained by spatio-temporal way-points to create these periodic motion. However, this method leads to unexpected behaviour when a high density of characters is requested. Indeed, to satisfy spatio-temporal constraints in a dense context the speed of characters could be abnormally high. A second approach consists in applying an optimization based approach to avoid collision in a patch [Ramirez *et al.*, 2014]. This method is able to handle denser situations than the Yersin's approach. However unrealistic behaviours can be observed in very dense scenario. An example of a patch generated by this method is shown in Figure 2.6. A last method consists in modifying quasi-periodic data of human trajectories to make it periodic [Li *et al.*, 2012], and then create crowd patches from it. With this approach, realistic motions are obtained. However it is hard to create a varied and connectable database of crowd patches with this technique.

A lot of work has been done with crowd patches. Yet, no method exits to easily design complex scenes with crowd patches, this is the problem tackled in this thesis.

2.2.4 Summary

Different approaches were introduced to improve the design of crowd animations. Methods based on intuitive interface to easily set parameters of a simulation, offer the opportunity for a beginner to design a population in a virtual environment. However, these methods are still based on simulation, and consequently suffer

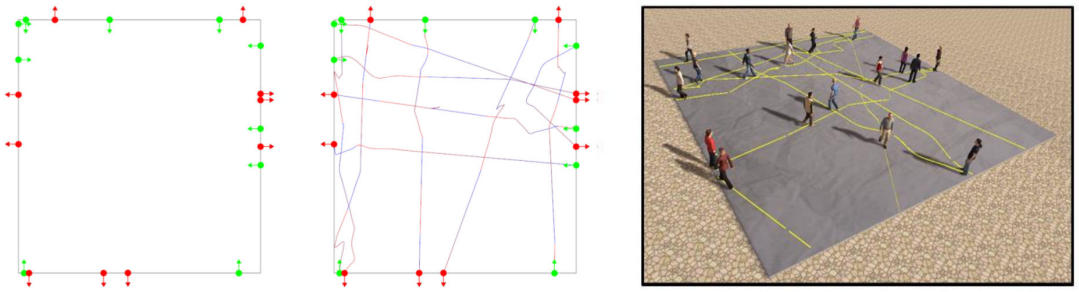

Figure 2.6: Optimization-based Computation of Locomotion Trajectories for Crowd Patches [Ramirez *et al.*, 2014]. Left: a patch with only entry and exit points. Middle: Generated trajectories based on the optimization algorithm. Right: The 3D rendering of the patch.

from limitations in the number of individuals and in possible unpredictability.

Techniques based on edition of existing data offer a direct an precise control over individual or group motions. However these methods require a huge database of crowd motion to have a rich variety of situations. Moreover, these techniques could be long and fastidious for an user in charge of populating a very large environment.

Unlike methods based on simulation, these based on patches allow crowd animation composed of hundred of thousand individuals for and infinite period of time. Although Patches concept eases the process of creation of crowd animation, they still lack control for placing them, inter-connecting them, or defining their general properties such as density or flow of characters. Moreover, these methods limit the variation in time of the crowd, because of the periodicity of patches.

2.3 Expressive Modeling

Expressive modeling techniques aim to interactively design 3D contents through intuitive gestures such as sketches, painting or sculpting. Such expressive methods offer high level of creation and edition for hiding the complexity of the task. Moreover, the underlying models are smart enough to react as expected to the

user's gestures, by automatically maintaining constraints, or by taking into account knowledge on the content to be modeled.

Intuitivity is a very important criteria for expressive tools, as opposed to interactive but not intuitive tools, such as The Gimp [Harford, 2000] to paint and edit your photos or Maya [**?**] to model and animate 3D contents, which require great expertise to be mastered.

The interactive design of crowds is still limited, as described in Section 2.2. The purpose of this section is to review other more general expressive modeling techniques to open novel perspectives of crowd design. Because important aspects of a crowd are the characters which compose it, their motion, and the way they are distributed, this section will focus on methods which are related to these problems. Firstly, we review techniques based on sketching metaphors in Section 2.3.1. Then those on painting metaphors, in Section 2.3.2. Next, methods based on interactive deformation, i.e. sculpting metaphors, in Section 2.3.3. Finally a summary on the different techniques is given, followed by a discussion on possible applications to crowds, in Section 2.3.4.

2.3.1 Sketching metaphors

For artists, sketches are the very first step to make his or her idea something concrete. In many cases, the final output is just some technical processes done by hand but inspired by the sketch. In computer graphics, we try to automatise these processes to get from the sketch an output as close as possible to the artist's intent.

? and **?** proposed two of the first gestural interfaces enabling to create and edit 3D content from sketches, *SKETCH* and *Teddy* respectively. The first one enables the creation of 3D objects and scenes by creating and assembling primitive shapes. The second one use 2D free-form strokes sketched by the user to automatically construct plausible 3D polygonal models.

? proposed a sketch-based method for designing trees. The technique incorporates botanical knowledge to infer branches that have a correct distribution in 3D. In addition, the statistical distribution of sub-branches extracted from the

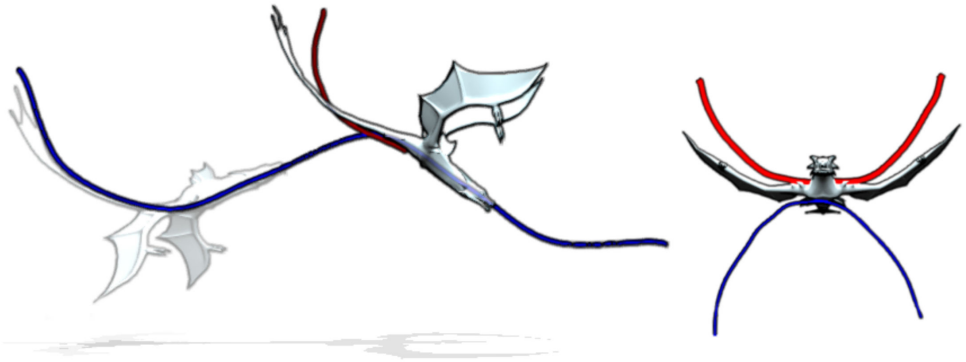

Figure 2.7: Space-time sketching of character animation [Guay *et al.*, 2015b]. **Left**: The user sketches a line of action stroke on top of a path-following dynamic line of action (DLOA) to alter the motion of the tail over a time interval. The path-following motion (a DLOA) blends with another DLOA, the static key frame sketched for the tail, over a time interval. **Right**: The user edits secondary lines onto a separate plane and view.

user's local sketch is automatically transformed to neighboring branches. Other methods allow to create and animate characters. For instance, Martin Guay uses lines of actions in [Guay *et al.*, 2013], to easily create stylized positions for characters by using a distance between shapes based on tangent vectors. This work is extended in [Guay *et al.*, 2015a], to enable the animation of characters based on several postures at different time-step. The user can freely edit the dynamic line between the different postures like an elastic band, to make parts of the animation faster or slower. Lastly in [Guay *et al.*, 2015b] the authors proposed to sketch a space-time curve in order to initialize the dynamic line of action, in order to easily drive the motion of the character through projective constraints. Moreover, this method is also able to add secondary motion to the same character, as it is illustrated with a dragon in Figure 2.7.

These sketching metaphors could be sources of inspirations to develop new expressive crowd designing tools. For instance, the method for sketching distributions of branches of trees is limited by static objects. It could be interesting to extend this method to animated content such as crowds. The method [Guay

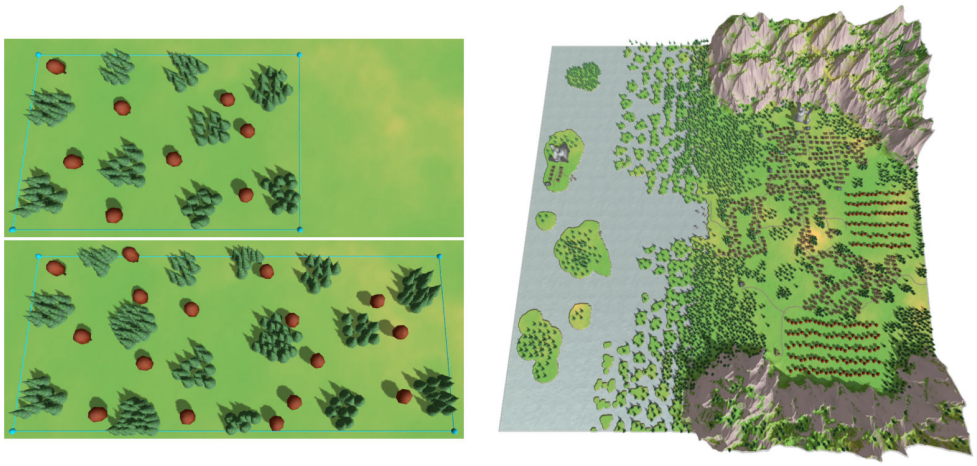

Figure 2.8: World Brush [Emilien *et al.*, 2015]. **Top-Left**: Initial arrangement. **Bottom-Left**: New trees have been seamlessly inserted in the empty region, preserving the visual appearance of the distributions. **Right**: A view of a complex scene edited with World Brush system.

et al., 2015b] controls animation over time, but is limited to a single character. Although, extension to multiple characters seems possible, it would be difficult to extend this kind of technique at the scale of a crowd.

2.3.2 Painting metaphors

Paint-like expressive applications use a brush abstraction to design content. Brushes can be seen as a property or a constraint to be applied everywhere you paint with. In some application you can paint with different brushes, so the properties blend together as color in a painting. The underlying system is automatically adapted to the user's gesture. For instance, Milliez use a motion brush to create an animation in [Milliez *et al.*, 2014]. With this method, artists are able to create animations as if they were painting motion. Motion brushes, thank to their hierarchical brushes, combine simple motions to create complex ones. Emilien *et al.* [2015] introduced world brush, a method able to design a virtual world based on statistical example-based synthesis to create and deform world content. The

user is able to select regions of the world, that have as effect to store statistical parameters of the region in a brush. Then the user is able to paint with this brush and blend several brush strokes together, to edit the world. The Figure 2.8 gives a picture of this tool.

In Section 2.2, we discussed about [Ulicny *et al.*, 2004], a method to paint crowd behaviours and motion, in a similar manner of the motion brush of Milliez *et al.* [2014]. However, it could be interesting to extend the work of Emilien *et al.* [2015] to animated content, to enable the interactive creation of populated world. In our work presented in Chapter 5, we also use a paint interface to draw crowds based on their density and main directions.

2.3.3 Sculpting metaphors

In 3D graphics, even simple manipulations as deforming a 3D-object can be a challenging task, but this task is essential to personalise them. **?** is the first to introduce the free-form deformation of a 3D model while preserving his volume. This enable to manipulate object as manipulating clay. Another major contribution on mesh deformation is the work of Sorkine & Alexa [2007]. The authors apply the principle of as-rigid-as-possible deformation to intuitively deform and twist 3D objects, while preserving the details on the object. A dedicated interface allows a user to select and move nodes of the mesh to deform it, as illustrated in Figure 2.9. The deformation of the new mesh is computed based on the minimization of an energy functions (Equation 2.2) with two unknown parameters: the difference between triangle positions and orientations of the new and original meshes.

$$E(C_i, C_i') = \sum_{j \in N(i)} w_{ij}\ dist((p_i' - p_j') - R_i(p_i - p_j)) \tag{2.1}$$

$$E(S') = \sum_{i \in V} E(C_i, C_i') \tag{2.2}$$

Where S is a triangle mesh; V is the set of nodes of the mesh; i a node; C_i and C_i' represent the original cell and the deformed cell at node i; $N(i)$) represent the neighborhood of the node i; p and p' represent respectively the original position and the new position of a node; and R_i the rotation of the node i.

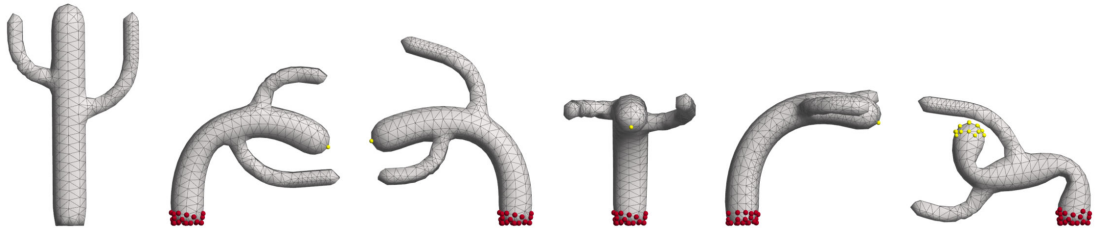

Figure 2.9: As rigid as possible deformation [Sorkine & Alexa, 2007]. **(a)** is the original model; yellow handles are translated to yield the results **(b-f)**, red handles are fix. **(d, e)** show side and front views of forward bending, respectively.

To solve for the next local minimum energy state, author use a alternating minimization strategy. Starting from a given initial vector of positions and rotations, the authors find positions p that minimize $E(S')$. Then, they find the rigid transformations R_i that minimize $E(S')$ for the given set of positions p'. finally, they continue these interleaved iterations until the local energy minimum is reached.

Bokeloh *et al.* [2011] present a deformation method able to detect and preserve patterns of the mesh. Firstly, the mesh is analyzed to find symmetries and repetitions. Then, the deformation is computed taking into account the structure of the mesh. This structure analysis approach is then extended with a new algebraic model [Bokeloh *et al.*, 2012] that enables the edition of complex structured meshes, such as chairs, stairs, or castles.

Milliez *et al.* [2013] introduce mutable elastic models (MEM). More precisely, MEM were introduced for static shapes able to take a number of different rest-states, all preserving a meaningful structure. Let us take the example of a castle wall. Rest states for each section of the wall can either be "straight" or "corner". When the designer bends a straight wall beyond a certain point, one of the sections will turn into a corner, because this new rest position will best minimize the associated deformation energy. Interestingly, the structure of shapes is captured in this work by a bidirectional graph representation $\mathcal{G}$, which deforms following the as-rigid-as-possible model [Sorkine & Alexa, 2007]: deformation is computed

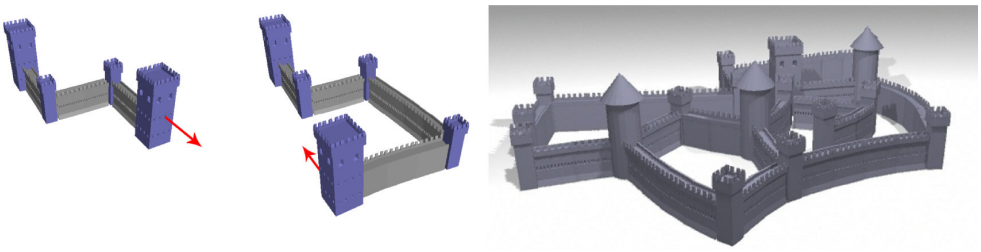

Figure 2.10: Sculpting structured shapes using a "smart clay" metaphor [Milliez *et al.*, 2013]: as the object deforms due to user constraints (blue), local piece s adapt their shape, and new pieces are inserted, deleted, and merged adaptively. The basis is mutable elasticity, permitting multiple local rest states for parts.

so that the graph parts globally remain as close as possible to their rest states. The method therefore plays on the mapping between each graph node and rest states using a state space representation. Changes of the graph topology are also allowed, through node insertion and removals. Both changes of state and topology changes are based on a shape grammar, to ensure that only valid transformations are applied. This system builds a 3D sculpting system which allows to elongate, compress, bent, cut, and merge the structured shape, by simple user gestures. As results a user can create and edit objects such as castle, roads, or chenille, by manipulating and deforming it as mutable clay, as illustrated in Figure 2.10.

As we have seen in Section 2.2, sculpture metaphor have also been used to create and manipulate crowds [Kwon *et al.*, 2008; Kim *et al.*, 2014]. However, these techniques are based on existing data and can't handle large deformation. Inspired by [Milliez *et al.*, 2013], we developed Crowd Sculpting a method able to manipulate crowd as mutable clay. In opposition with [Kwon *et al.*, 2008; Kim *et al.*, 2014], crowd sculpting have no limitation in deformation. All the details of this method are given in Chapter 4.

2.3.4 Summary

Expressive modeling methods are definitely an efficient way to create and edit 3D content. However, these techniques are often limited to static objects. Adding an-

imation constraint can be an good opportunity to extend these models to crowds. It will allow to hide the complexity of crowd simulation models while offering an interface based on intuitive gesture to ease the design process design and to enable a direct control on crowd motion and behaviour.

2.4 Conclusion

A crowd is a complex system with hardly predictable behaviour. Consequently, simulation approaches, which offer an non-direct control on crowd creation, are not adapted to crowd design. Some existing methods provide interfaces to get a closer control of crowd motion, but these techniques are still based on crowd simulation so they are limited by same limit of crowd simulation. Others techniques propose motion data editing through expressive interfaces for populating environments. However, too few of these techniques exist and present limitations due to data acquisition, a fastidious user manipulations, and a short time of animation. At least texture based approaches allow to populate very large environments without limit on time animation. However, these techniques lack of expressive modeling to easily cover the scene and define intrinsic parameters of the crowds, such as the repartition of characters and their main directions. You can refer to the Table 2.1 to have a clear summarize of the related work.

Knowledge on interactive methods for modeling demonstrate that many manners exist to control complex mathematics models through intuitive interfaces. These techniques are sources of inspiration to improve the crowd design. This thesis will take advantage of crowd patches and knowledge on expressive modeling, to develop new methods able to ease the process of crowd design.

Methods		Direct control	Large crowds	Endless animation
Simulation	[Reynolds, 1999]	- -	++	+
	[Helbing *et al.*, 2000]	- -	+	+
	[van den Berg *et al.*, 2008b]	- -	+	+
	[Narain *et al.*, 2009]	- -	++	+
Interface-based simulation	[Ulicny *et al.*, 2004]	+	+	+
	[Patil *et al.*, 2011]	+	+	+
Motion Editing	[Kwon *et al.*, 2008]	++	+	-
	[Kim *et al.*, 2014]	++	++	-
Texture based method	[Yersin *et al.*, 2009]	-	++	++

Table 2.1: Summary of the related work.

Chapter 3

Crowd Patches Framework

Contents

As motivated in the related work, crowd patches have good properties for populating large environments. This whole thesis is based on the crowd patches paradigm. This platform and related methods is used all over the thesis. In this chapter, we detail the principles of crowd patches and present our implementation, within a dedicated platform. Moreover, unpublished methods for creating space-time constraint trajectories for crowd patches are exposed.

3.1 Crowd Patches Principle

A *crowd patch* (Figure 3.1) is a pre-calculated animation of a virtual crowd. It correspond to a small area traversed by moving characters. Characters animation is periodic, so it can be played endlessly in time. Boundary conditions for animation trajectories are well controlled to enable connections between patches. In analogy to puzzle, crowd patches is able to cover large environments. More formally, a *crowd patch* is a tuple $\{\mathbf{A}, \pi, \mathbf{D}, \mathbf{S}\}$ where $\mathbf{A} \subset \mathbb{R}^2$ is the convex 2D geometrical area where the animation takes place, π is the period of the animation and $\mathbf{D}$ and $\mathbf{S}$ are the sets of dynamic and static objects, respectively. *Static objects* are simple obstacles whose geometry is fully contained inside the patch, whereas *dynamic objects* are animated ones; i.e., they are moving in time according to a set of constrained spatio-temporal trajectories. There are two categories of dynamic objects: *endogenous* and *exogenous* characters ($\mathbf{D}_{en}$ and $\mathbf{D}_{ex}$ respectively). Endogenous characters remain inside $\mathbf{A}$ for the entire duration π of the patch whereas exogenous leave $\mathbf{A}$ and enter other patches.

3.2 Synthesizing Crowds using Crowd Patches

A patch can be considered as a spatio-temporal right prism with base area $\mathbf{A}$ and height π. By defining spatio-temporal control points on each of the lateral faces of the prism (called *patterns*), input and output points (I/O points) can be defined. These points define portals between patches where characters can respectively enter or leave the patch (Figure 3.1). Two patches can be connected

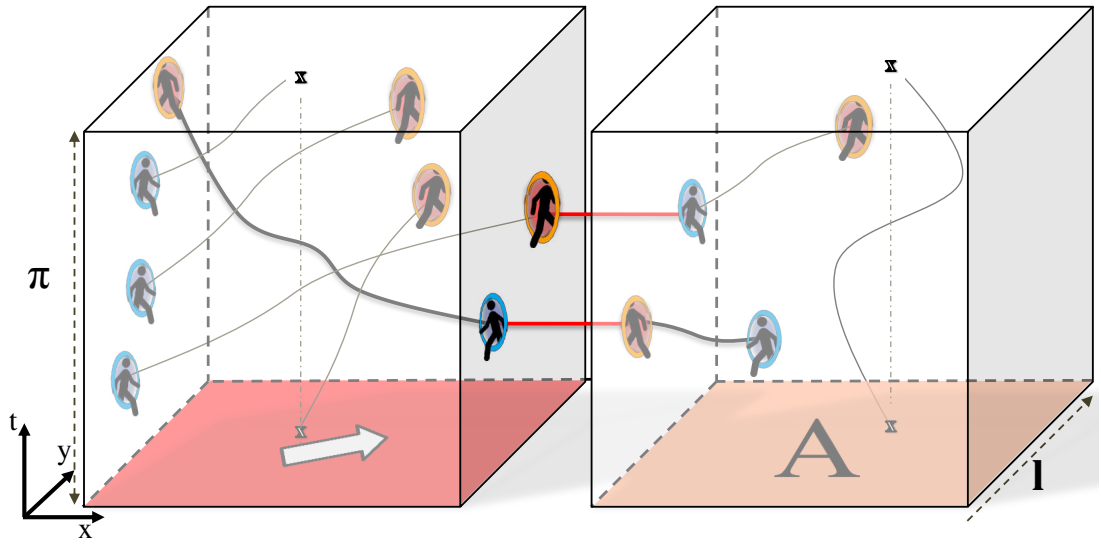

Figure 3.1: **Patches and Patterns** Adjacent patches can be connected if they have matching mirror patterns. Shading on the base of the patch indicates density and arrows represent flow direction.

if they have matching patterns but with points of opposite purpose; i.e., input and output points switch roles. Therefore large π-periodic crowd animations can be created by assembling patches.

Here there is one important constraint; the total number of input and output points of a patch must be equal; i.e., exogenous characters entering a patch must leave it at some time.

Crowd animation systems that use crowd patches as building blocks for crowds are typically decomposed into 4 stages:

1. *Patch decomposition.* The scene to be populated is divided into smaller convex areas where periodic crowd animations will be computed.

2. *Patches definition.* Patches parameters are then computed/defined so that boundary constraints are not violated.

3. *In-Patch trajectories generation.* Boundary points in single patches are connected and internal collision free trajectories are generated.

4. *Animation.* Finally, characters are placed on the trajectories like trains on rails and the animation can be played.

Existing methods [Yersin *et al.*, 2009; Li *et al.*, 2012; Ramirez *et al.*, 2014] focus on stage 3, and manually or automatically set stages 1 and 2. We also designed techniques taking place into stage 3. These methods are described in the following section. Mainly, our methods focus on stages 1 and 2. The methods [Jordao *et al.*, 2014] focus on stage 1, and [**?**] focus on stage 2. A description of these two methods is done Chapter 4 and 5. Stage 4 is also a challenging problem, discussed in Section 3.4.

3.3 In-Patch trajectories generation

As seen in Section 2.2.3, several methods exist to generate trajectories in crowd patches fitting with spatio-temporal constraints at their borders. Still, these methods present several limitations. The method exposed in [Yersin *et al.*, 2009] can't handle dense scenario and often lead to unrealistic results even in non-dense scenario. The Ramirez's method [Ramirez *et al.*, 2014] is better compared to those of Yersin, but still provoke oscillation in dense scenario. The Li's method [Li *et al.*, 2012] generate realistic trajectories, but the acquisition process is based on motion capture, or real motion data of pedestrians, which makes the method difficult to use.

The following subsection will present two new manner for creating trajectories in crowd patches. These work was done together with interns and engineers of my team. For now these techniques are unpublished.

3.3.1 RVO-based algorithm

Generating trajectories in a crowd patch demand to (i) define entry and exit point on patterns of the crowd patch; which can be done in an automatic or manual way, (ii) match the different entry and exit to define where characters will enter and exit from the crowd patch; which can be handle by the stable matching

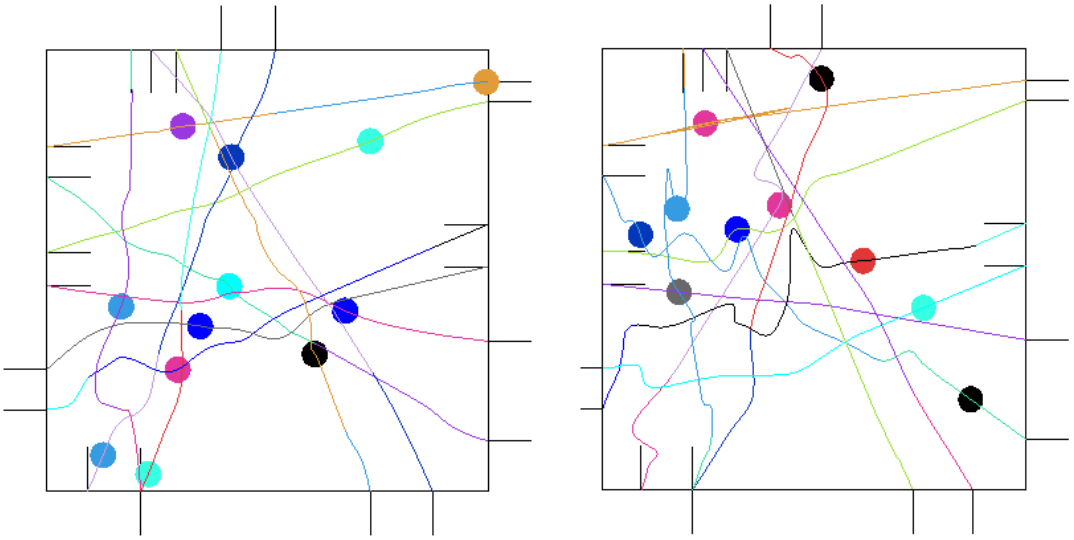

Figure 3.2: The same Crowd Patch, generated with two different algorithm. Left: Trajectories generated with the RVO-based algorithm. Right: Trajectories generated with the Ramirez's algorithm [Ramirez *et al.*, 2014].

pairing of Ramirez *et al.* [2014], and (iii) use a dedicated algorithm to generate collision free trajectories. In this part, the RVO-based algorithm is explained.

3.3.1.1 Principle

As seen in Chapter 2, Section 2.1.1.3, the Reciprocal Velocity Obstacles (RVO) model [Guy *et al.*, 2010] is an efficient simulation system enabling characters to anticipate possible collisions with other characters or obstacles.

The idea of the method is to take benefits from the RVO model while ensuring entry and exit of crowd patch characters in times defined by patterns. More accurately, the algorithm proceeds in two steps. In a preliminary step, input points from patterns are linked to output points, using for instance the matching pairing of Ramirez *et al.* [2014]. For each pairs, we compute the theoretical speed ν of the character. Where (p_i, p_o) and (t_i, t_o) are respectively positions and times of the input and the output.

$$\nu = \frac{\|p_o - p_i\|}{t_o - t_i} \tag{3.1}$$

If ν is more or less fifty percent of the preferred speed, intermediate way-point is added to slow down correctly the speed of the character. More way-points can be added if one does not slow down enough the character speed. Moreover, these way-points are set to keep a uniform speed of the character, and to be outside obstacles or too close from other way-points. Then the initialization ends by defining obstacles and agents which start at $t = 0$ (which are mostly those from way-points) in the RVO simulator.

Next in a second step, we modify the preferred velocity of agents at each loop of the simulation process, to ensure the constraints defined by IO points of patterns and way-points. The agent accelerate or decelerate according to the time he or she expects to reach his or her goal. All this process is shown is Algorithm 1.

3.3.1.2 Discussion

Our RVO-based algorithm allows to produce trajectories which predict collisions, and avoid oscillations. The Figure 3.2 display results from RVO and compare them with those from Ramirez *et al.* [2014] algorithm. However, it still does not allow to reach very dense scenario as those propose for instance in [Narain *et al.*, 2009]. Indeed, the high number of characters prevents some agents to achieve their goal in time. Consequently the system teleports missing agents, which create unrealistic behaviours.

3.3.2 Crowd patches editor

Automatic methods for generating trajectories of characters are an effective manner to create crowd patches. However, the trajectories generation may fail, because the requested density is too high, or may be inadequate because resulted motion does not fit with what we expect. For these cases or for more control on crowd motion, manual intervention could help creation of crowd patches. Consequently, we develop a tools able to generate and edit motion in a crowd patch from an empty one. In the following section, we present the features of this tool.

input : Array I, O of inputs and outputs points of the crowd patch.
input : Array Ob of obstacles of the crowd patch.
output: Crowd Patch P

`/* Initialization */`
IOPairs $L \leftarrow StableMatching(I, O)$;
`/* Speed should be more or less fifty percent of the preferred speed. */`
Float $threshold \leftarrow 0.5$;
for $l \in L$ **do**
    while $-threshold < minimumSpeed(l) < threshold$ **do**
        Point2D $p \leftarrow getBestPosionBetweenPoints(l.i, l.o)$;
        $addWayPoint(p, L)$;
    end
end
$setObstacles(Ob)$;
Agents $A \leftarrow initRVOAgents(L)$;
Trajectories T;
Patch P;
`/* Main loop */`
while $time < period$ **do**
    $modifyPrefSpeedToArriveInTime(A)$;
    $doRVOStep(A)$;
    $recordTrajectories(A, T)$;
end
$addTrajectories(T, P)$;
return P ;

Algorithm 1: RVO-based algorithm for generating trajectories in a crowd patch.

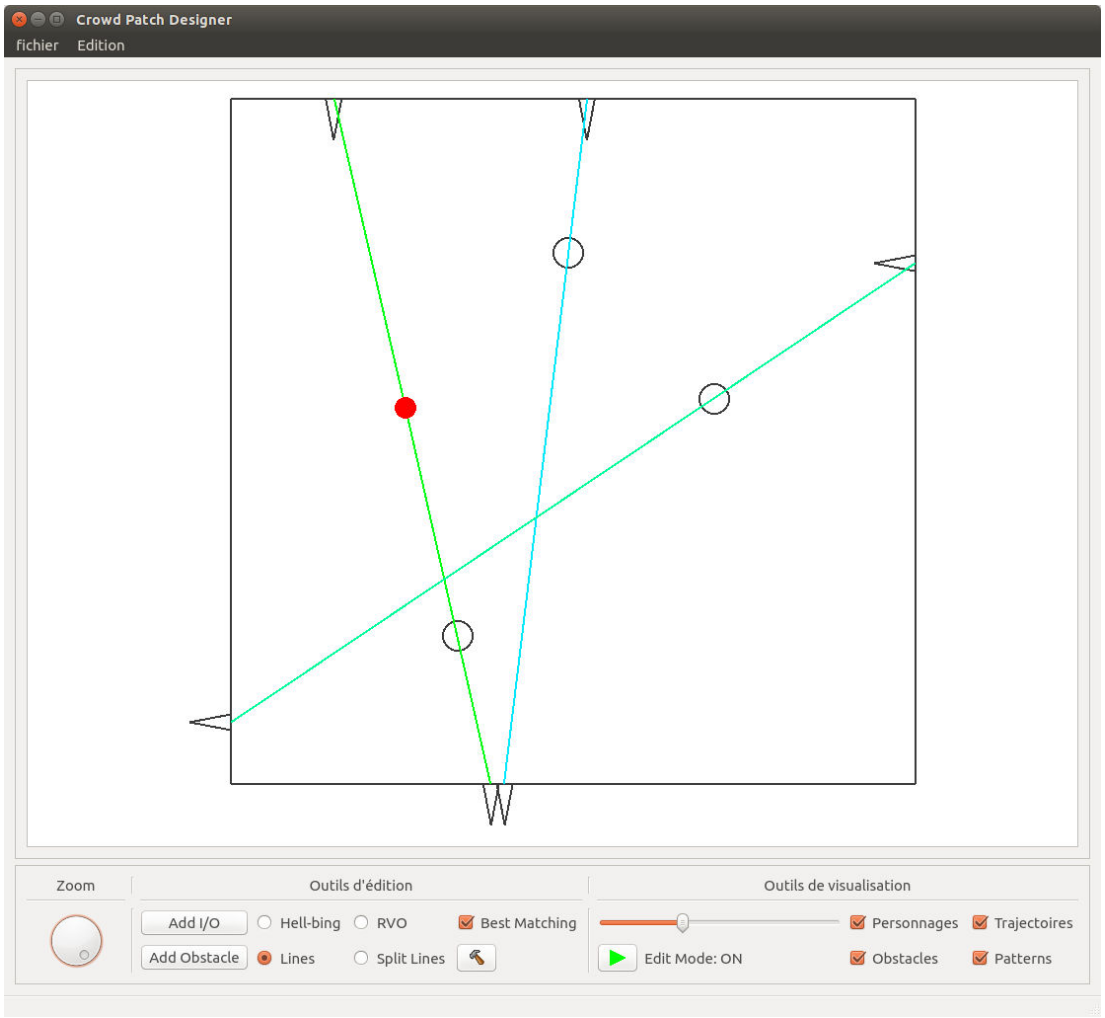

Figure 3.3: Crowd Patches Editor. The application allows to have a view of the current edited crowd patch. Trajectories are displayed following the speed of the character on it. Characters are represented by empty circles. Red circle is a control-point, manipulated by the user. The part under the main view of the application allows to edit the crowd patch, by generating trajectories or adding IO points or obstacles, and to tune the displayer.

3.3.2.1 Features of the tool

The crowd patches editor provide many features to create and edit a crowd patches. An screen-shot of the application is shown on Figure 3.3.

Intrinsic parameters of the patch and patterns The first step to create a patch from scratch is to define its patterns, i.e. the borders of the patch polygon, the period, the average speed and the width of characters. A special window allow to set these parameters.

Entry and exit points Before having moving character in a patch, the user should define entry and exit point on patterns. Adding such spatio-temporal constraint is done by specifying for both entry and exit their position in space and time and on which pattern. It also possible to randomly generate IO points.

Obstacles Can be added in the crowd patch by setting their position, orientation and size. The user can choose to do it after or before the trajectory generation. But once trajectories are generated collision obstacles/agents are not checked any more.

Trajectory generation The trajectories generation enable two operations: the pairing of entry and exit points, and the computation of the characters motion. The pairing can be made by the best matching algorithm from Ramirez *et al.* [2014], or in the order entries and exits was defined. Four types of generations are proposed: (i)the method based on Helbing simulation technique [Helbing & Molnár, 1995] proposed by Yersin *et al.* [2009]; (ii) the optimization-based method proposed by Ramirez *et al.* [2014]; (iii) the RVO-based method described in the previous sub-section; (iv) a simple straight line generation. Note that the last technique is used only for future manual editing, and does not have collision avoidance system.

Trajectory editing The crowd patches editor also allows to edit existing trajectories. User is able to grab a trajectory and to deform it following an as-rigid-as

possible principle. During this stage, collisions between agents are not checked. The user is in charge to modify trajectories taking care of collisions.

Other features Other features are classical, such as a zoom to be more accurate while deforming trajectories, or displaying or not some information of the crowd patch. Trajectories are colored depending on the speed of the character, from blue to red, green being the preferred velocity. The tool enable to play forward and backward the animation of the crowd patch, and to stop it. Finally, the user is able to load/save crowd patches, and import/export them from other project, such as crowd art or crowd sculpting.

3.3.2.2 Discussion

This editor gathers methods from several published work [Yersin *et al.*, 2009; Ramirez *et al.*, 2014] and unpublished work (see 3.3.1), and allows to edit crowd patch motion by hand. This editor is particularly useful for editing patch trajectories which have not reaches the desired quality. However, at this state of development, no collision avoidance process exist to automatically move trajectories when another is edited. A future direction of this work could be to add such feature. For instance, by inspiring of the collision avoidance process used in [Kwon *et al.*, 2008]. Moreover we can imagine modifying group of motion as it is done in [Kim *et al.*, 2014].

3.4 Viewers

As explained in the crowd patches pipeline (see Section 3.2), a system to display our results in essential. This Section presents two viewers developed in the team. The first is a basic 2D viewer displaying moving dots. The second is a viewer able to display 3D animated humanoids.

3.4.1 SFML Viewer

The SFML viewer is developed based on the Simple and Fast Multimedia Library (SFML) version 2.0 [**?**]. This viewer is able to display all information of crowd patches, such as characters shaped by dots, entry and exit points of characters, the trajectories, the obstacles and the border of the patch. A rendering of this viewer is shown Figure 3.2. The viewer is design to support generic Boost graph [**?**] able to react to keyboard, mouse and joystick event. It is easy to integrate the SFML viewer into new projects, thanks to its flexibility. In term of performance, the SFML viewer is able to display up to one billion of characters in interactive frame rate. This viewer is useful during implementations process to track bugs.

3.4.2 Unity Viewer

The unity viewer is a more complex 3D-viewer based on Unity 5 game engine [**?**], entirely developed by the engineer Tristan Le Bouffant. The viewer is shown Figure 3.4. In this viewer characters are 3D animated humanoids running in realistic environments with several light effects. As input the viewer is able to load files listing positions of characters over time. Thanks to this feature, we are able to display crowds from any projects. However, for interactive applications, such as crowd sculpting [Jordao *et al.*, 2014], a special plug-in is developed. This viewer is mainly used during demonstration processes to make realistic rendering of crowd in immersive environment. This Unity viewer is able to run a crowd of a thousand characters in interactive frame rate. A special offline mode enable to create video clip for more characters.

3.5 Summary

During this thesis a full and flexible framework for crowd patches has been developed, for developing new expressive methods based on this paradigm. Many features have been introduced for crowd patches, such as new methods to generate trajectories, or deforming the geometry of the patch (as explained in Chapter

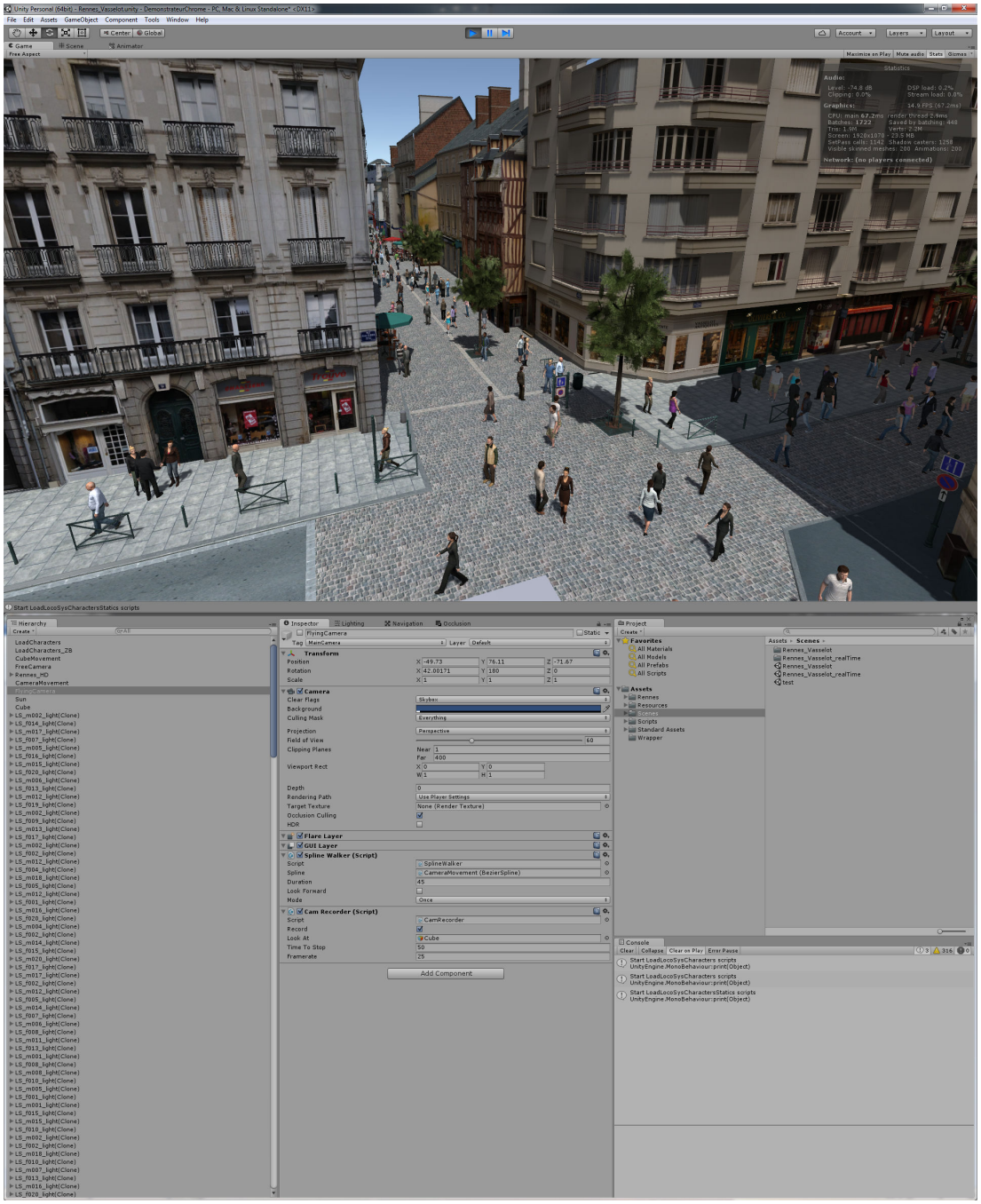

Figure 3.4: Crowd Patches Unity viewer. Top: a view of the scene. Bottom: A window to set parameters of the scene.

4), and a new tool to interactively edit them. The framework is used by several colleagues and many projects depend on it.

The framework is used as a C++11 library with the following dependencies : the Standard Template Library (STL), boost graph v1.59.0 [**?**], Eigen v3.2.5 [**?**], and RVO v2.0.1 [**?**].

Chapter 4

Shaping Crowd Animations

Contents

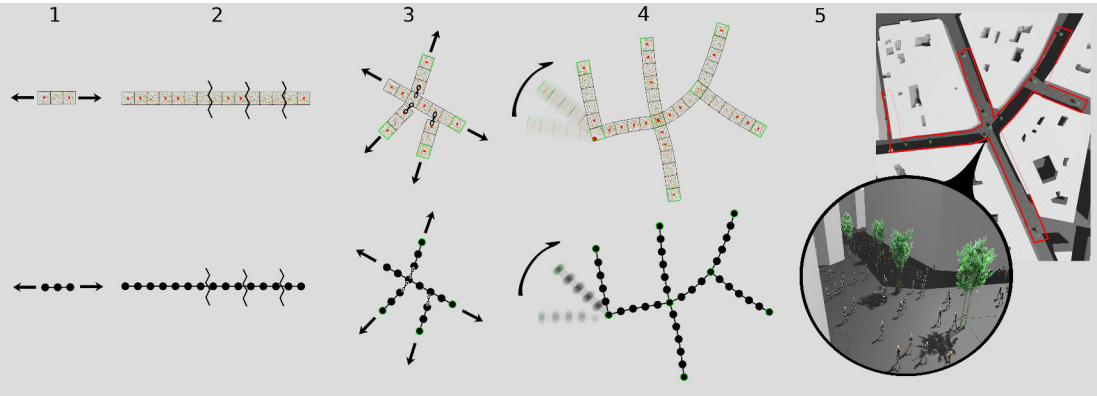

Figure 4.1: Five steps illustrating the creation and interactive manipulation of Crowds Patches to populate a virtual environment by introducing spatio-temporal mutable elastic models. Pictures on the top line illustrate the manipulation of the patches while pictures on the bottom line illustrate the changes in the graph representing the Mutable Elastic Model.

4.1 Introduction

This chapter presents a method to interactively design populated environments by using intuitive deformation gestures to drive the spatial coverage of a crowd motion. Our approach assembles large environments from sets of *Crowd Patches*, to avoid expensive and difficult-to-control simulations. It also overcomes the limitations of motion editing, that would result into animations delimited in space and time. The insight in this work is to replace the large deformations needed for sculpting a whole crowd by a number of local deformations and of local changes of animated content: using such changes will ensure that each crowd patch, while being able to continuously deform, only undergoes acceptably small deformations. More probably, each crowd patch will follow the user's gesture, such as stretching, shrinking or bending, and be transformed to other content whenever the deformation becomes too large, in a similar manner to the Mutable Elastic Models (MEMs) [Milliez *et al.*, 2013]. The reader can refer to Figure 4.1 for a clear picture of this process. This approach however leads to a number of challenges. Indeed, the overall spatial and temporal consistency of the crowd should be maintained, which requires continuity between crowd patches (therefore be-

tween agent trajectories). Moreover, they need to be provided with immediate visual feedback of the whole animated content, which is indeed the only way for them to achieve interactive design.

Our examples demonstrate that our method allows the space-time editing of very large populations and results into endless animation, while offering real-time, intuitive control and maintaining animation quality. Our specific contributions are:

- a novel method to interactively design complex animated crowds for virtual environments, with high level gestural control and immediate visual feedback. Note that the animations we generate can be endlessly looped in time, thanks to the crowd patches method we rely on.
- an extension of the mutable elastic models, introduced for shapes only, to enable the manipulation of structured geometries containing animations, while preserving spatial continuity between them.

4.2 Overview

The principle of our approach is to represent an animated crowd as a graph-based structure in which nodes represent animated crowd patches (the reader can find a description of the crowd patches in Chapter 3), and edges represent the connected flows between the crowd patches. See Figure 4.2 for understanding the structure of our method: the top-left part represents the structure of the crowd as a graph, the top-right part displays the non-deformed crowd patches corresponding to the graph nodes, and the bottom part represents the final deformed patches that shape the crowd animation and ensure continuity. From there, by proposing a novel extension of Mutable Elastic Models [Milliez *et al.*, 2013], user interactions applied to this representation can (i) impact the topology of the graph by inserting, removing or connecting nodes together thereby switching patches and changing the way patches are connected, (ii) impact the geometry of the patches (bending, stretching, shrinking thereby changing the way agents are moving inside a patch).

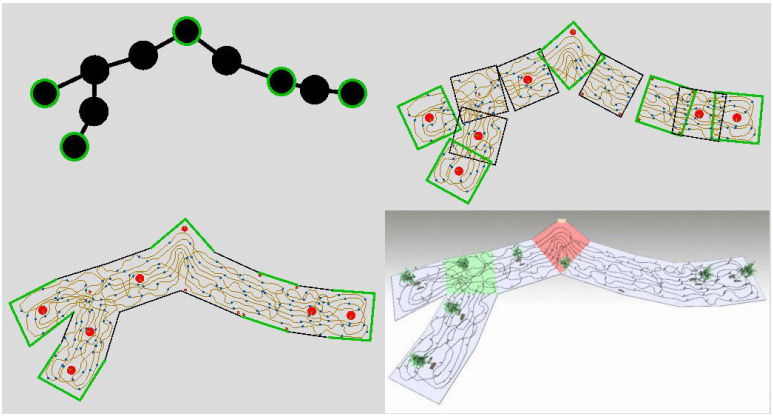

Figure 4.2: A simple example to illustrate the representation we use for continiously deforming crowd animation. **Top-Left**: A geometrical graph $\mathcal{G}$ captures the structure of crowd patches. **Top-Right**: Each vertex of $\mathcal{G}$ corresponds to a crowd patch. **Bottom-Left**: Crowd patches are locally deformed to enable seamless animation continuity between patches. **Bottom-Right**: A 3D rendering of the crowd.

In practice, crowd patches are organized into a bi-directional graph structure storing adjacency relationships. Up to now, no feature was proposed to deform the individual patches or to change the topology or geometry of this graph. Our insight is to extend Mutable Elastic Models (MEMs) [Milliez *et al.*, 2013] (see MEMs description in Section 2.3.3) to the crowd patches framework: MEMS are a unique solution for enabling large deformations of a complex structure while limiting the distortion of individual elements (here the geometry of crowd patches, and therefore of the trajectories). Whenever local deformation is too large, elements are either interactively swapped with more suitable ones, or some of them are inserted, removed, connected or dis-connected, as necessary, within the graph.

Although we inspired from the original idea, MEMs cannot be directly applied to our crowd patches framework. First, Milliez *et Al.* did not solve for spatial continuity between the geometric elements they manipulate. In contrast, local deformations maintaining C_0 continuity at least is required in our case, in order to

ensure smooth trajectories for virtual agents. Second, the elements of animation we manipulate require specific treatments to ensure proper temporal connections.

In the following section, we therefore present a new *mutable model*, based on crowd patches. They require defining:

- a set of rest-state configurations, each rest-state representing a specific patch type and containing a patch instance (see Section 4.3.1);
- a state space which maps user interactions to changes in the topology of the graph structure (see Section 4.3.2);
- a mean to locally deform the geometry of crowd patches to follow the graph deformations (see Section 4.3.3);

4.3 A mutable model for sculpting crowds

The first step is to define a number of rest-states together with their corresponding patches and then the state space which describes the possibilities to swap rest-states or change the topological structure of the graph.

4.3.1 Patches Rest-states

A rest-state is a predefined local configuration in the graph, a specific spatial arrangement of nodes and edges. Figure 4.3 illustrates the different rest-states available in our system, typically:

- a **dead-end** (1 edge only), $\chi^{(DE)}$,
- a **flat node** (2 aligned edges only), $\chi^{(F)}$,
- a **L-corner** (2 orthogonal edges), $\chi^{(L)}$,
- a **T-junction** (3 orthogonal edges), $\chi^{(T)}$
- or a **X-junction** (4 orthogonal edges), $\chi^{(X)}$.

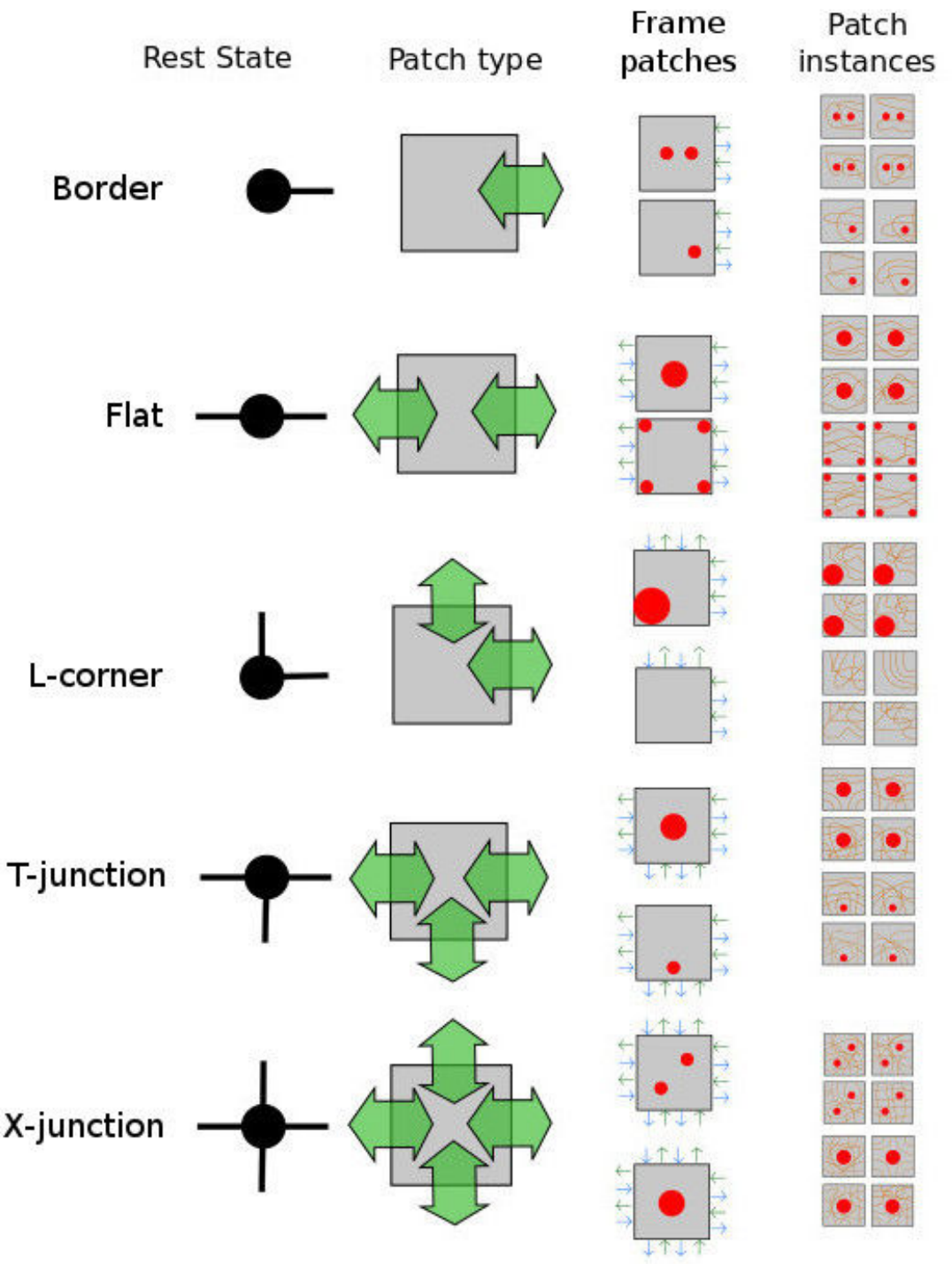

Figure 4.3: Table of possible rest-states defined in our system (1st column), corresponding types of patches (2nd column), patches layout (3rd column) and examples of patches instances (4th column). The red dots represent obstacles in the patches.

Each rest-state is first associated with a patch type. The patch type typically defines the side on which agents flow between patches, yet without precisely defining densities of agents nor entry/exit patterns of agents. For the sake of simplicity, we restricted our patches to square-shaped regions. Each rest-state can therefore be connected to a maximum of 4 other patches. Each patch type is then associated with a collection of *frame-patches* including only entry/exit patterns and obstacles, displayed in red in Figure 4.3. Each frame-patch is then associated to different patch instances which each contains different agent trajectories (all satisfying the entry/exit patterns and avoiding the obstacles), thereby providing a good degree of variability.

Frame-patches and Patch Instances: Frame-patches are delimited by patterns, which capture spacetime way-points for characters traversing patch boundaries. Frame-patches do not contain trajectories, but simply specify constraints. Patterns capture the flow going through each of the patch boundary. This means that we can easily deduce crowd patches instances from their type and frame-patch. We set empty patterns $p_\emptyset$ (without any way-point) where there is no flow allowed. We set a desired number of way-points for other patterns depending on the density of the patch. Combinations in the densities and in the patterns enable a first level of variety in the possible patches.

However, we must guarantee that the set of patches we use easily interconnects. Our solution is to create patches instances from a very limited set of patterns $\{p_1, ..., p_n\}$ as well as their mirror patterns $\{p'_1, ..., p'_n\}$. Indeed, we recall that two patches can connect if the spacetime way-points on their boundary match: one input point should correspond to one output point in the adjacent patch. This condition is true for two mirrored patterns p_i and p'_i. By using a limited set of patterns, it is possible to pre-compute all the possible combinations of patterns to create different frame-patches, and then different patch instances. For example, an instance of a dead-end patch type patch is built with the following 4 patterns for each node: $\mathcal{P}_{dead\ end} \leftarrow \{p_i, p_\emptyset, p_\emptyset, p_\emptyset\}$. This patch can be connected for example to a flat-node type patch built with these 4 patterns: $\mathcal{P}_{flat} \leftarrow \{p'_i, p_\emptyset, p_j, p_\emptyset\}$.

Once patterns and obstacles are set in a frame-patch, different trajectories can be generated (for more information on trajectories generation please refer to Chapter 3 and [Yersin *et al.*, 2009; Li *et al.*, 2012; Ramirez *et al.*, 2014]). These combinations therefore enable a second level of variety in the possible patches.

Each time a new node N is inserted in the graph, all the patch instances of that node type for which all entry/exit patterns are compatible with its neighbors in the graph, are selected in a pool of possible candidates. A random process then chooses one among the possible candidates, and associates this patch instance with the node.

4.3.2 Patches State Space

We now describe how the graph deforms and evolves under a set of possible user actions, as well as how patches switch between different rest-states.

State Space: A state space is introduced to map each node in $\mathcal{G}$ with one rest-state. There is no ambiguity in this mapping when nodes are dead-ends, T or X-junctions, because the number of connected edges straightforwardly determines the corresponding rest-state. We however have an indetermination in the case of 2-edge nodes, that can either have a flat or a L-corner node rest-state. To solve this ambiguity, we define a state for nodes with two edges (rightmost green region in figure 4.4). Our state space is two dimensional. The state s_i of the node i is defined in the 2D state space as:

$$s_i = [l_i, \beta_i] \tag{4.1}$$

where β is the angle formed by the two edges connected to the node and l the length sum of the connected edges. We can now project the node into the state space. The state space is displayed in Figure 4.4. Depending on the position of the node in this space, we deduce its corresponding rest-state (color-coded areas in the state space). The figure also illustrates that, as detailed later, the user's actions will change the state of nodes, depending on their distance to the rest-states we defined. Distance and projection in the state space are computed

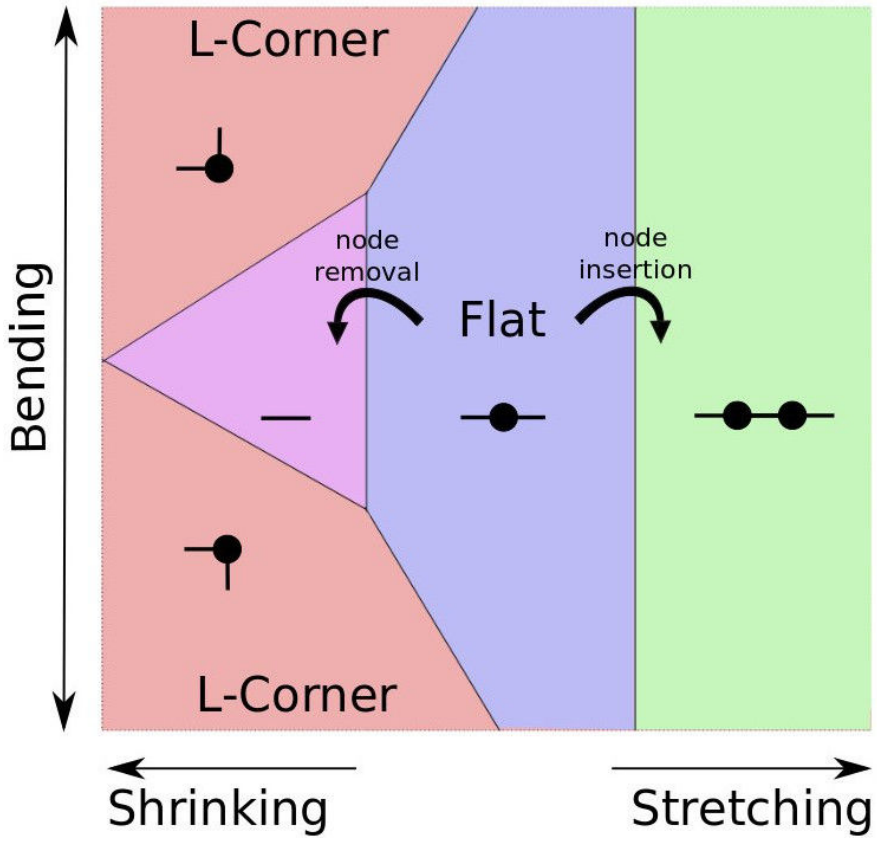

Figure 4.4: Presentation of the state space. A colored region represents a state and boundaries represent transitions. User deformations move nodes away from their rest-state and inside this state space (the distance is measured using a specific metric integrating position and orientation information of edges). When reaching a boundary in the state space, the current rest-state is either switched to another one (Flat towards L-Corner) or leads to a change in the graph structure (node insertion or removal).

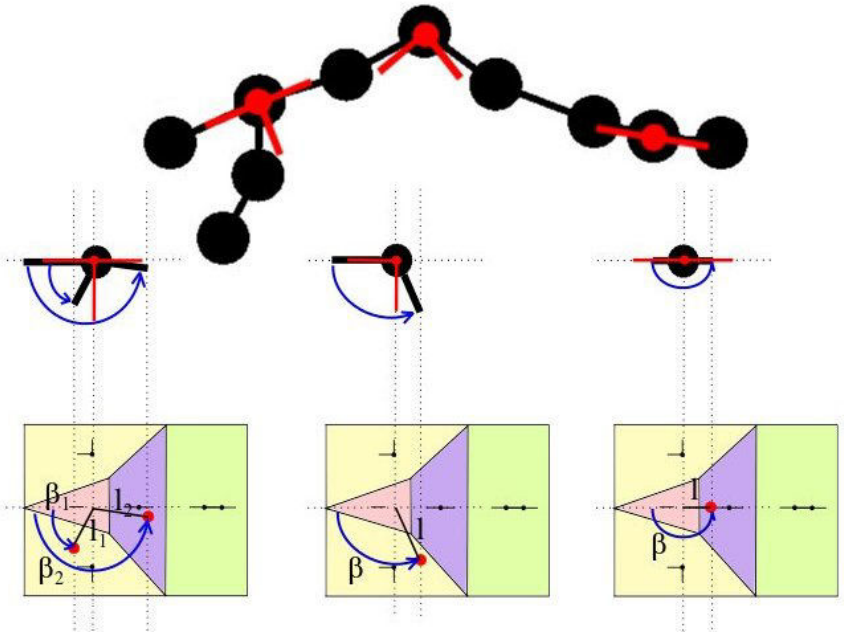

Figure 4.5: Illustration of the deformation energy $E(f)$. Each node is associated a rest-state (closest one its state space). Here three rest-states are illustrated in red, each one associated to a state space in which the current configuration of the nodes (in black) are positioned. Distance between each node and its rest-state is then evaluated (using angles and edge lengths). Distances are then summed on all nodes to compute deformation energy.

according to the metric presented in [Sorkine & Alexa, 2007] and based on the deformation energy $E(f)$:

$$E(f) = \sum_{p_i \in \mathcal{G}} dist(R_i(A_i), f(\chi_i)) \tag{4.2}$$

where $E(f)$ is the energy to minimize, p_i the graph nodes, and *dist* the distance between the configuration of the considered node $R_i(A_i)$ to its rest position $f(\chi_i)$. A_i is the set of possible rest-states $A_i = \{\chi_i^{(DE)}, \chi_i^{(F)}, \chi_i^{(L)}, \chi_i^{(T)}, \chi_i^{(X)}\}$. The method of Sorkine & Alexa [2007] is explained in Section 2.3.3. An example of mapping between the nodes of an example graph and the rest-states is displayed in Figure 4.5.

Cutting and Merging: The user is able to cut or merge parts of the graph. These actions are direct and explicit actions on the graph $\mathcal{G}$. The user determines an edge to delete from the graph (cutting) or some new edges to add to the graph between two nodes (merging). In our application, the user performs cutting by selecting two adjacent patches and selecting the "cut" action. To merge two components of $\mathcal{G}$, the user moves one of the component (global translation / rotation) close to the other one, selects two close patches and select the "merge" action. An edge is created between the two corresponding nodes of $\mathcal{G}$.

Cutting and Merging actions directly result into changes of rest-states for the considered nodes. When cutting, the edge removal will mutate the connected nodes:

- X-junction into a T-junction: $\chi^X \rightarrow \chi^T$,
- T-junction into a flat node ("trunk" edge is cut): $\chi^T \rightarrow \chi^F$,
- T-junc. into a flat node ("branch" edge is cut): $\chi^T \rightarrow \chi^L$,
- Flat or L-corner node into a dead-end: $\chi^L, \chi^F \rightarrow \chi^{DE}$.

Note that we did not consider isolated nodes because this case is useless. And conversely, when merging, edge insertion will result into change in rest-state for the newly connected nodes: $\chi^{DE} \rightarrow \chi^L, \chi^F$, $\chi^L \rightarrow \chi^T$, $\chi^F \rightarrow \chi^T$. We choose whether χ^{DE} should turn into χ^L or χ^F by choosing the rest-state at closest distance from the current state of the considered node.

Node rest-state changes also result in a change of patches type (and consequently patch instances). The newly mutated node will connect to existing neighbors. Connection is made through patterns. These existing constraints define the correct patch-frame to be used. We then randomly select any instance of a patch with the corresponding patch-frame. Examples of cutting and merging operations are illustrated in Figure 4.1, steps 2 and 3. One can see flat nodes turning into dead-end nodes at step 2. In step 3, one flat node is successively merged with two other components: it turns into a T then into a X-junction node.

Stretching, shrinking and bending: Second, stretching, shrinking or bending operations can be performed on the graph as illustrated in Figure 4.1, step 4. The part to be deformed is selected by defining control nodes (represented as pins in our examples). The in-between structure of nodes is deformed in as-rigid-as possible way. As explained in [Sorkine & Alexa, 2007], this is done computing the nodes configuration that minimize $E(f)$, Equation 4.2. This new configurations change the 2-edge nodes state.

Projection in the state-space (Figure 4.4 and 4.5) determines whether a swap in state (or mutation) is required. For example, when a flat node is stretched, its state is moved to the left of the state space: indeed, this move in the state space corresponds to a shortening of edge length. At some point, the state will go beyond the limit (between the pink and violet areas) where node removal is triggered. In the same way, when bending a flat node, the angle formed by node edges will increase or decrease: mutation to a L-corner χ^L rest-state is triggered (transition from the violet zone to the orange zone). A hysteresis at the frontier of different rest areas is implemented as suggested by Milliez to avoid disturbing oscillations between different rest-states.

As for cutting and merging, when a node mutates to a new rest-state or is added to $\mathcal{G}$, the matching patch-frame is selected and any instance corresponding to this frame is added to the animation.

4.3.3 Geometric deformations of patches

The elastic deformations controlled by the user cause some changes of distance and alignment between patches. As crowd patches contain some animated characters that move from patch to patch, un-alignment and distance changes result into discontinuities in the animation that need to be addressed (an aspect not tackled by [Milliez *et al.*, 2013]).

To ensure animation continuity, we propose to locally deform the patches geometry together with the animation to connect them seamlessly. Such a local deformation is illustrated in Figure 4.6. The results of a complex deformation is illustrated in Figure 4.2. Patches adjacency is easily deduced from $\mathcal{G}$. It allows us

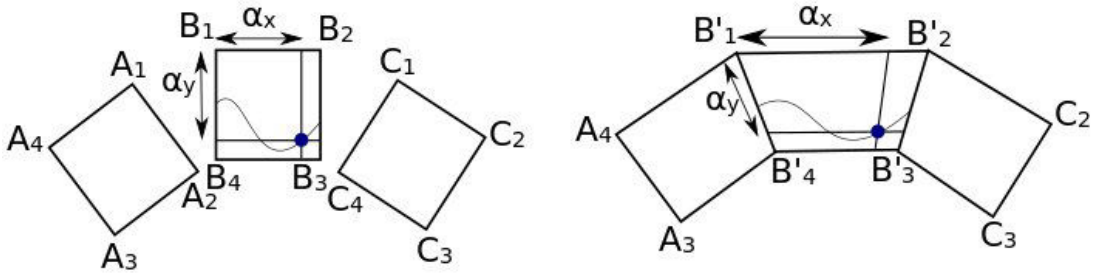

Figure 4.6: Local deformation of patches. Vertices of two adjacent patches are interpolated. A bilinear interpolation of the internal animation trajectories is performed to fit the new patch position and shape.

to map patch vertices of adjacent patches two-by-two. In the example of Figure 4.6, A_2 and B_1 are mapped, as well as A_3 and B_4. This allows to compute the new coordinates of locally deformed patches: they stand at the center of mapped vertices. For example, B'_1 is at the center of A_2 and B_1. Finally, all the internal trajectories as well as internal objects coordinates are deformed to follow the new shape of patches vertices. This local deformation is performed using a simple bilinear interpolation. Each trajectory control point τ_{xy} is located at the same coordinates (α_x, α_y) expressed relatively to the moving axis $(\overrightarrow{B'_1B'_2}, \overrightarrow{B'_1B'_3})$. Given that deformations on the patches are limited (due to changes in states that arise) the bilinear interpolation only leads to small deformations on the trajectories.

4.4 Results

We have implemented our crowd sculpting method with crowd patches framework described in Chapter 3. On top of this, two visualization components were developed for the sake of manipulation and visualization. First, a simple 2D trajectory visualization component in which characters are displayed as moving points based on SFML. And second, a more complete visualization module based on the Unity Game Engine (see Figure 4.8). Note that in the current version we are using, Unity does not allow the display of more than 1.500 characters. Therefore our examples with more characters are only showed using our 2D vi-

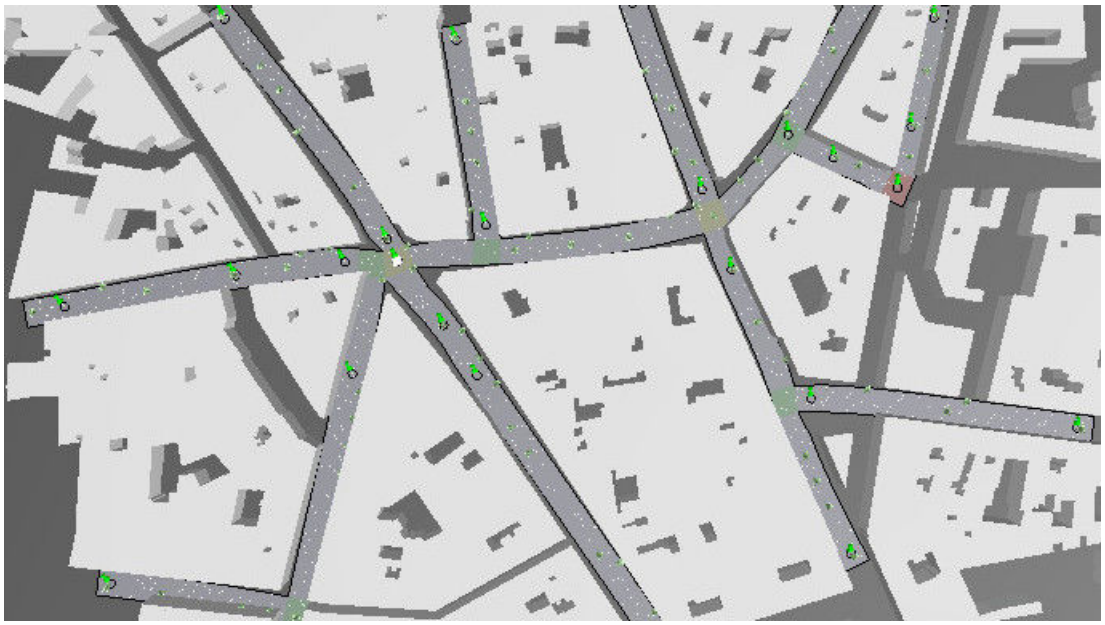

Figure 4.7: Example of an animated city. Existing geometry is populated by crowd sculpting.

sualization component.

As displayed in the companion video[1], interaction with the crowd is made using the mouse. The user selects the patches he wants to interact with by pinning them. The pinned patches are static or being moved by the mouse. The user then drags the selected patches to deform the structure. All the patches between two pinned and selected patches freely deform. As the patches are deformed all the trajectories are recomputed in real-time.

The crowd patches method is efficient, because computations are limited to data replay; complexity remains linear in the number of patches and characters. Together with deformation and the simple 2D visualization, the method is fast and able to deal with large environments. Based on the kind of examples we show in Figure 4.8, we are able to simulate the crowd and deform the structure of patches with smooth user experience (refreshing rate above 20 frames per seconds) up to 500 patches, which represents an average of 6000 characters (and given that our implementation is mono-thread, there is large room for improvement).

[1]Online version: `https://youtu.be/YNurdcADhlA`

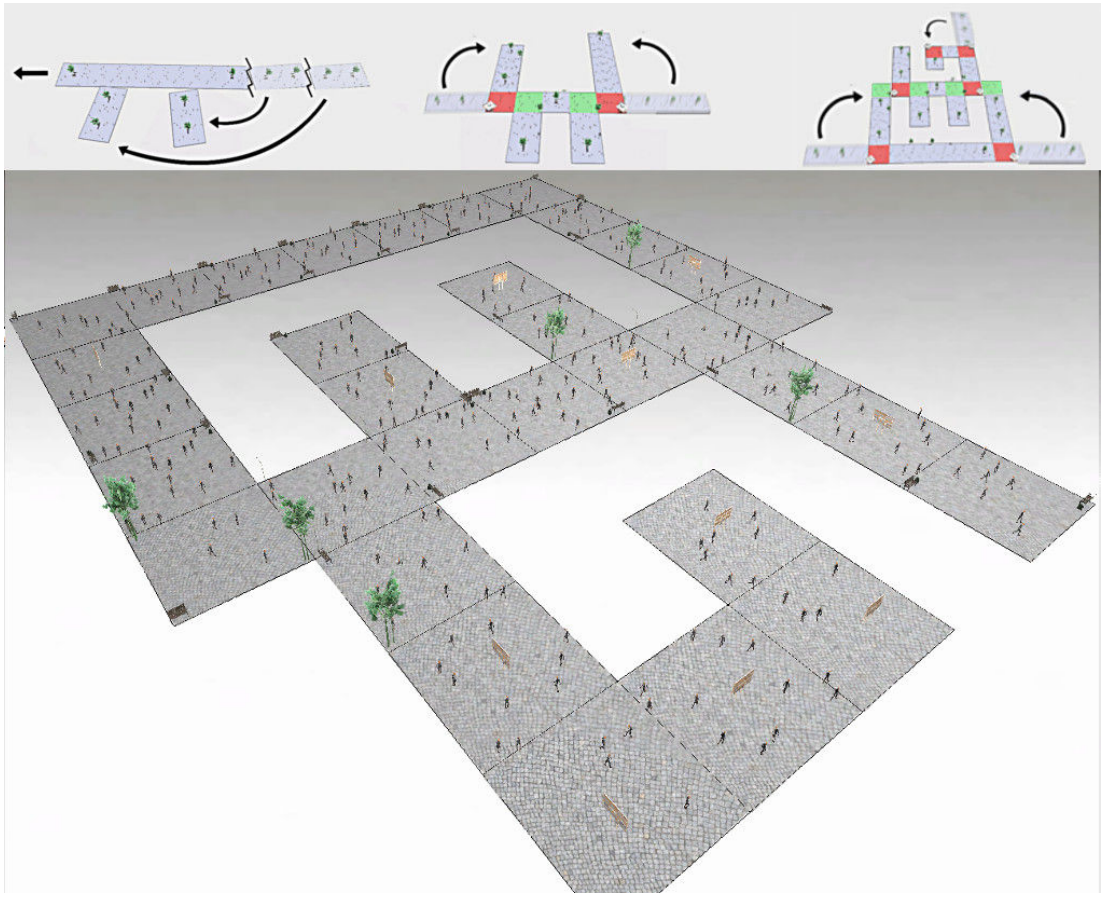

Figure 4.8: Example of an animated crowd shaped as the Eurographics logo.

Figure 4.7 and 4.8 illustrate two results. In the first example, we deformed a structure of patches to match an existing environment. We reached the desired result in 15 minutes. These 15 minutes of Crowd Sculpting are almost entirely displayed in the companion video ($\times 10$ acceleration). In the second example, we created a crowd moving along the EG logo. We reached the desired result in 5 minutes after a dozen of deformations.

4.5 Summary

The *crowd sculpting* method we presented is the first approach that makes the task of populating large virtual environments at the reach of any user. It offers

a unique interaction experience with the complex animations in a crowd: indeed, designers can quickly manipulate a full crowd motion through a few sculpting gestures, enabling complex environments to be populated easily and quickly. Compared to crowd simulation approaches, we offer direct high-level control of trajectories, instead of indirect parameters. Compared to crowd motion editing techniques, we are not limited in terms of amount of deformation, animation duration or spatial coverage. Finally, compared to CrowdBrush Ulicny *et al.* [2004], we do not edit some crowd simulation parameters but directly manipulate the coverage and shape of a crowd motion. We believe that it opens a new path for crowd motion design, by providing both new control paradigms and an interactive support for expressivity.

Still, a limitation of this editing method is that it only produces crowds with an uniform or quasi-uniform density. The next chapter introduces the first patch based method for designing crowds with space-varying density.

Chapter 5

Density and Flow Control of Crowd Animations

Contents

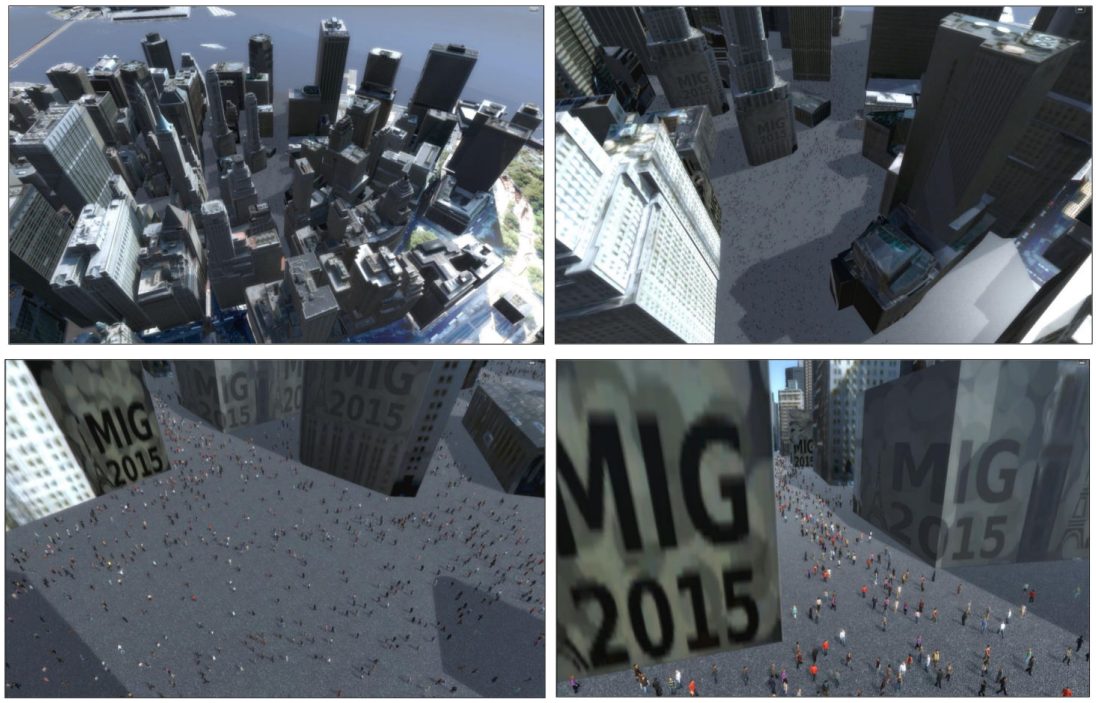

Figure 5.1: Our system can be used to populate a large city like environment such as the Wall Street area in New York with crowds of different density and direction constraints in minutes. Resulting crowd motion can then be played endlessly always satisfying the user's intent.

5.1 Introduction

This chapter presents a method to intuitively populate virtual environments by specifying two key features: localized *density*, defined as the amount of agents per unit of surface, and localized *flow*, defined as the direction in which agents move through a unit of surface. The technique we propose is also stationary in time, meaning that whatever the time in the animation, the resulting crowd satisfies both features. As in the previous chapter, we rely on the *Crowd Patches* model introduced by Yersin *et al.* [2009]. We first discretize the environment into regular patches and create a graph that links these patches. Then, we generate an assembly of patches, in which patches locally match a user-defined level of density and flow direction, while maintaining boundary constraints. This is done

by applying an optimization process that operates on the patch graph – a graph whose nodes are patches and edges are faces that connect patches. Optimization is performed along patch parameters in the graph; these are (i) the number of way-points at the boundaries of patches to account for density, and (ii) the connections between possible entry end exit way-points to account for flow direction. The resulting animation progressively converges to match the expected constraints, by using the difference between the actual features and the expected constraints as a cost function. Another computational stage is however necessary both to avoid unwanted behaviours such as characters walking in small loops, and to globally create flows that satisfy the user inputs. To this end, we first identify distant patches which differ the most from user input. A path-planning process is then proposed to compute paths in the patch graph linking such patches. Along these paths, entry and exit points are created to construct a trajectory for characters.

As a result, the method has the capacity of generating large realistic crowds in minutes that endlessly satisfy both user specified densities and flow directions, and is robust to contradictory inputs. At last, to ease the design the method is implemented in an artist-driven tool through a painting interface.

The contributions are the following:

- an optimization technique to compute crowd animations that satisfy different sizes under localized density and directional flow constraints;

- a process to avoid local loops in characters' trajectories by computing paths linking unsatisfactory patches;

- an artist-driven tool for designing crowds. Designers can create crowds very rapidly using an existing paint tool. User requirements are specified by combining image layers which specify exogenous and endogenous density, direction and obstacles.

This chapter is organized as follows. Section 5.2 provides a global description of our solution and of its main components. The optimization technique to compute patches with desired density and flow direction is detailed in Section 5.4.

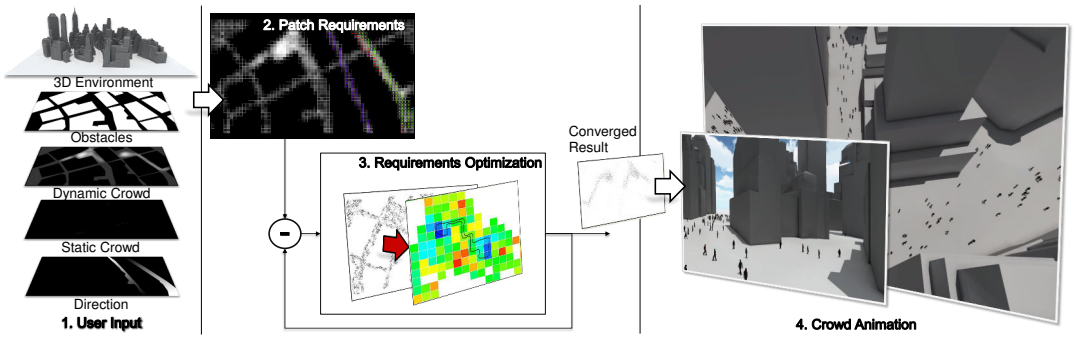

Figure 5.2: **Overview of the Crowd Art platform.** (1) Users define a set of maps that annotate the environment with crowd information. (2) These maps are merged to generate a set of crowd patches under density and direction constraints (color indicates direction). (3) Crowd patch parameters are iteratively optimized to satisfy user requirements which are then used for (4) real-time animation of large crowds.

Section 5.5 introduces a simple interface for users to sketch those inputs. Finally, results are discussed in Section 5.6.

5.2 Overview

Our solution to control crowd density and direction first requires (i) the provision of an interface to design the crowd requirements and (ii) an optimization process by which the crowd is generated (Figure 5.2). In the first step, users can use a painting interface to draw areas of endogenous and exogenous people as well as paint motion directions through color gradients (Section 5.5). They can use and combine as many layers of constraints as necessary; additionally these maps can be overlayed on top of environment maps to match obstacle free regions.

Secondly, these maps are merged and discretized to generate a graph describing density and flow direction requirements for the environment. Each node of the graph corresponds to a *crowd patch* [Yersin *et al.*, 2009]. These patches are then constructed and iteratively optimized to satisfy the user constraints (Section 5.4).

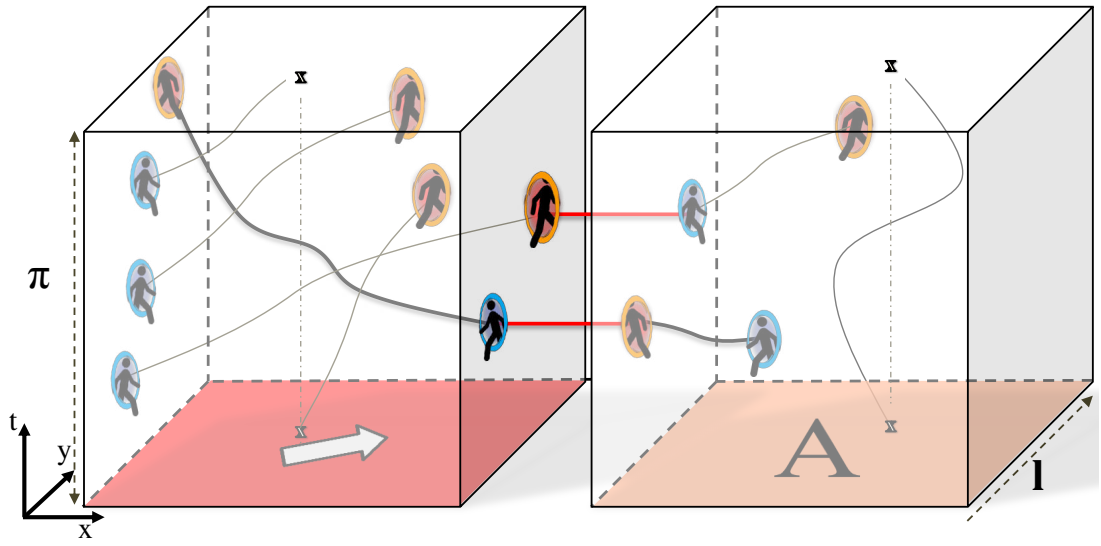

Figure 5.3: **Patches and Patterns** Adjacent patches can be connected if they have matching mirror patterns. Shading on the base of the patch indicates density and arrows represent flow direction.

5.3 Density and Direction Control in Patches

We aim in computing parameters of our crowd patches from user-defined density and flow directions values over period π. Both are supposed to remain constant in time. Density in a patch can be defined as:

$$\rho = \frac{1}{\pi}\int_0^{\pi} \rho(t)dt = \frac{1}{A\pi}\int_0^{\pi} |\mathbf{D}(t)|dt. \tag{5.1}$$

Assuming patches of constant size A and period π, density can be changed only by playing on the number of characters $|\mathbf{D}(t)|$ that are present in the patch at any given time. Recall that we have two kinds of characters; endogenous $\mathbf{D}_{en}$ and exogenous $\mathbf{D}_{ex}$. Endogenous affect density during the entire period of the patch, whereas exogenous affect density dynamically and are dependant of the number of input and output points and inter-connections between them (Figure 5.3). Equation 5.1 therefore can be written as:

$$\rho = \frac{|\mathbf{D}_{en}|}{A} + \frac{1}{A\pi}\int_0^{\pi} |\mathbf{D}_{ex}(t)|dt. \tag{5.2}$$

Therefore density can be controlled in two ways:

- by modifying endogenous characters $\mathbf{D}_{en}$.
- by modifying input and output points on the sides of the patch (this changes $\mathbf{D}_{ex}(t)$).

Playing only with endogenous characters would result in quasi-static crowds. Therefore, playing with the number of inputs and outputs point is a much better solution. The difficulty to find a correlation between these points and the density value. In our work, we use the following approximation : we directly use the number of input and output points as an approximation for density.

This is justified by the experimental measurements shown in Figure 5.4 that demonstrate a direct correlation between the two with small variance (due to temporal placement and optimal connections between points). To compute these data, we generated multitudes of patches of different parameters (area A, period π), and different numbers and placement of IO points; endogenous characters were not considered. For each of these patches, we found optimal connections [1] between points and then measured the resulting density (Equation 5.2).

Flow direction in patches can be defined by the way input/output points are placed and how they are interconnected. A connection between an input and output point defines a single direction $\mathbf{d}_i$ that lasts t_i seconds. By knowing all connections $\mathbb{D} = \{\mathbf{d}_i : i \in [1, n]\}$ in a patch, we define the main flow direction of a patch as:

$$\mathbf{d} = \sum_{i=1}^{n} t_i \mathbf{d}_i. \tag{5.3}$$

One can observe the difference of density and flow direction between two neighboring patches in the simple example displayed in Figure 5.3.

[1] Optimal connections were retrieved using the approach of Ramirez et al. Ramirez2014.

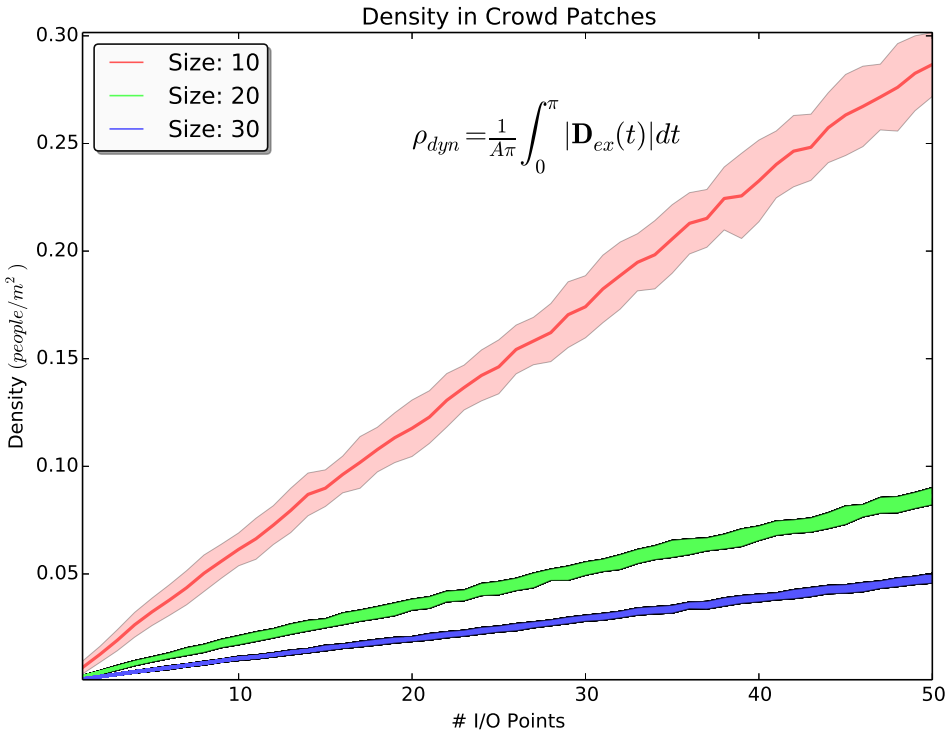

Figure 5.4: **Density and I/O points** Experimental data demonstrating the correlation between density and the number of I/O points for patches of different size and same period ($\pi = 30$ secs). Shaded regions represent the variance in density.

5.4 Optimizing Crowd Requirements

In this section we explain how to construct an assembly of crowd patches with locally controlled density and flow direction. An optimization-based strategy is employed to find the optimal crowd patches parameters without violating boundary conditions between neighboring patches.

Optimization Strategy The main principle behind our optimization strategy is to add or remove sets of input and output points at boundaries of patches at each step of the optimization loop aiming to get as close as possible to the user inputs. We measure a signed error for each patch. The error is computed as a combination of (i) a density error measuring the difference in terms of number of characters between existing and expected values in a patch and (ii) a direction error measuring the difference between the expected direction and the weighted sum of actual characters' directions (Equation 5.3). Given that every change on a patch impacts its neighboring patches – therefore changing neighboring densities and flows, the steps are repeated until convergence. Convergence is met when error change is negligible.

Constraints During optimization, two types of constraints need to be satisfied: user and patch constraints. User constraints define desired density and direction where needed; every patch has a density constraint whereas direction is optional and can be defined in parts of the environment. Patch constraints on the other hand concern cardinality of input and output points as well as compatibility between patterns of neighboring patches (Section 3.1 and 3.2); i.e., patches must be connectable and periodic. Additionally, there is a maximum allowed density value ρ_{max} for all patches.

Representation Our problem is modelled as a graph $G = (V, E)$; nodes V and edges E indicate patches and connections between them respectively. Each node $u \in V$ stores the measured density $\rho_{u,c}$ in the patch, the required density $\rho_{u,r}$, the measured direction $\mathbf{d}_{u,c}$ (if needed) and the required direction $\mathbf{d}_{u,r}$. Directions are unit length 2D vectors.

5.4.1 Algorithm

For all the remaining, please consult Algorithm 2; numbers near the paragraph titles indicate lines in the algorithm.

Initialization *(Alg. 1, lines 1, 2)* The graph is initialized by setting all patches to have the same density value so that each one of them has the exact same number of input/output points and can be easily connected without violating patch constraints (setting all patches to a zero density value is a possible initialization). The initial values for density affect the speed of convergence which is also dependant on the complexity of the density and direction requirements (Section 5.6). Given the initial density value, we use a regression on experimental data in order to compute the initial number of input and output points in patch. Figure 5.4 displays the relation between density (vertically) and number of input/output points for different patch sizes. Additionally patch size A and period π are uniformly set by the user depending on accuracy requirements; smaller patch sizes lead to better approximation of density.

Finally, during initialization the graph is split into strongly connected components using Tarjan's algorithm [**?**]; optimizations are then performed independently on each connected component (Section 5.4.2). Splitting the graph first reduces the overall complexity of the process and then allows to concentrate the modifications in strongly connected areas (i.e. where there are more possibilities of connections between components).

Measuring Convergence *(Alg. 1, lines 3, 22)* To measure the error to the desired solution, we define a function $E(G)$ that composes density and direction errors of the entire scene at each iteration step ($E_\rho(G)$ and $E_d(G)$ respectively):

$$E(G) = E_\rho(G) + E_d(G) \tag{5.4}$$

We define $E_\rho(G)$ to be the Root Mean Square Error (RMSE) of density scaled to the maximum allowed density ${\rho_{max}}^2$:

[2]This is currently user defined and set to 0.35 $characters.m^{-2}$.

input : Graph $G = (V, E)$ of density and main direction user constraints.
output: Graph $G = (V, E)$ modified to satisfy all constraints.

```
1  InitRandomSolution(G);
2  𝕊ℂ ← StronglyConnectedComponents(G);
3  while not converged do
       /* Mutate Graph G                                          */
4      for each component G_i = (V_i, E_i) ∈ 𝕊ℂ do
           /* Update error values on nodes and edges              */
5          UpdateSignedError(G_i, V_i);
6          UpdateEdgeWeights(G_i, E_i);
           /* Find error extrema and fix them in groups           */
7          max ← ErrorMaxima(V_i) > 0;
8          min ← ErrorMinima(V_i) < 0;
9          MG ← CreateGroups(max);
10         for each group of maxima m ∈ MG do
11             path ← MinimumCostPath(G_i, m);
               /* Check if points can be deleted                  */
12             if CanRemove(path) then
13                 RemovePoints(path);
14             end
15         end
16         MG ← CreateGroups(min);
17         for each group of minima m ∈ MG do
18             path ← MinimumCostPath(G_i, m);
19             AddPoints(path);
20         end
21     end
22     globalError ← measureGlobalError(G)
23 end
24 return G;
```

Algorithm 2: Optimizing density and main direction constraints. Each mutation manipulates I/O points and affects a set of patches.

$$E_\rho(G) = \frac{1}{\rho_{max}} \sqrt{\frac{1}{|V|} \sum_{u \in V} (\rho_{u,r} - \rho_{u,c})^2} \tag{5.5}$$

We then define $E_d(G)$ based on the angle between $\mathbf{d}_{u,c}$ and $\mathbf{d}_{u,r}$:

$$E_d(G) = \frac{1}{|V_d|} \sum_{u \in V_d} (1 - \mathbf{d}_{u,r}.\mathbf{d}_{u,c}) \tag{5.6}$$

where $V_d \subseteq V$ represents the subset of patches that have direction requirements. Now the interesting aspect here is that rather than computing the current direction $\mathbf{d}_{u,r}$ from existing trajectories, we actually optimize the assignments between inputs and output points and then measure the direction. The assignments are based on optimal matching by extending the work of [Ramirez *et al.*, 2014]. The direction is measured using the weighted average direction of connections (a connection being a straight line between I/O points). Weights are simply defined as the duration between the input/ouput points of a connection. Ramirez *et al.* [2014] optimize the connections between pairs of input/output points in a patch by using a score function that gives more imortance to points on opposing patterns based on the preferred speed of each agent. We extend this in two ways; by defining a new matching function that additionally takes into account the direction contraints and by setting the preferred speed of agents in any given patch based on the density requirements. We base the latter on the fact that people tend to move slower in dense rather than sparse situations [**?**].

5.4.2 Optimization steps

During each step of Algorithm 2, three basic operations are performed to minimize Equation 5.4: (i) finding areas with large errors, (ii) selecting subsets of them and (iii) removing/adding points in patches that lie on these paths of minimal cost between them.

Error and Local Extrema *(Alg. 1, lines 4-8)* At every optimization step, we set an error value on each node $u \in V$ of the graph:

$$e(u) = e_\rho(u) + sign(e_\rho(u)).e_d(u) \tag{5.7}$$

$e_\rho(u) = \rho_{u,c} - \rho_{u,r}$ is the signed error in density, $e_d(u) = |1 - d_{u,r}.d_{u,c}|/2$ is the error in direction and $sign(.)$ is the sign function. Positive values of $e_\rho(u)$ indicate that the node has more density than requested whereas negative indicate that the node is lacking density. Values of $e_d(u)$ near 0 indicate good direction whereas values near 1 indicate opposite direction. *Local positive maxima* of $e(u)$ in G indicate neighborhoods of patches that either have an abundance of density or direction is not correct, whereas *local negative minima* indicate areas that need characters and have bad direction (Figure 5.5).

Fixing errors *(Alg. 1, lines 7, 8, 9, 16)* We use these positive maxima and negative minima patches as starting points to fix problems in the patches. First, all of them are found and grouped based on distance, type (maxima and minima) and if they belong in the same connected component; these groups typically consist of 2–5 extrema. Then, circular paths that aim in minimizing error between the patches of a single group are found. We emphasize that these paths are not actual paths; rather they act as on/off switches between patches that open doors so that when the time comes, connections can be made that satisfy user constraints. Also, by forcing circular paths we ensure that pairs of input and output points are added so that boundary constraints are not violated. Additionally, we limit the matching of extrema based on distance so that it is easier to find paths and minimize error instead of actually increasing it.

Path finding between extrema *(Alg. 1, lines 10-15, 17-20)* Having a set $\mathbb{O} = \{O_i : 1 \leq i \leq k\}$ of k extrema, a path between each pair $\{O_i, O_{(i+1)\%k}\}, \forall i \in [1, k]$ of extrema is found. In the case where these extrema are positive maxima, points must be deleted (Figure 5.5 right). To do so, a path between two maxima is found that minimizes the derivative $e'(u)$ between any two patches in the path (i.e., the direction of slower descent). More importantly, each pattern that connects two patches in the entire path must have enough I/O points to allow for deletion. After the entire path between all maxima is found, pairs of input and output points that connect consecutive patches are deleted. In the case where the extrema are negative minima, the operation is slightly easier since we do not need to make sure that points exist or not on each pattern touched by the path (Figure 5.5). Here, we again aim in minimizing $e'(u)$ and when the entire path is

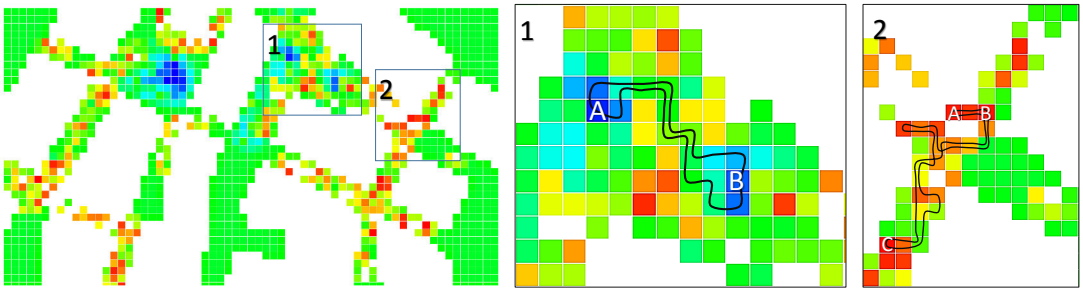

Figure 5.5: **Fixing errors.** (left) The leftmost image shows the signed error map at one step of the optimization. Green values indicate 0, cold and hot colors indicate negative and positive values respectively. The darkest blue indicate negative minima and red positive maxima. (middle) Two minima A and B have been selected based on distance and a path of minimal cost is found between A and B and back; pairs of output/input points are added on the sides that connect two patches of the path increasing density. (right) Three maxima have been selected and paths from A to B, B to C and C to A are found; pairs of input/output points are removed on the path only if points can be removed from all patches on the path. Observe that in this case, patches that were ok (green) were modified to fix erroneous ones.

found, points are added instead of deleted.

We note that in both cases, it is possible to have patches in the path where error increases; this effect is minimized by following patches with "high" same sign errors and by adding a penalty if we cross the boundary where we have $e(u) = 0$. Even though some patches can have an increase in error, these are typically fixed on a following update step. Finally, instead of modifying all groups of maxima and minima found at each step, we select a few; this selection is simply a linear function of the number of patches in a given connected component, i.e. if $G_i = (V_i, E_i) \subset G = (V, E)$ is a connected component, we update at each step $\lfloor V_i/400 \rfloor + 1$ paths.

Internal Trajectories *(Alg. 1, line 22)* At the end of each optimization step, current density and direction values ($\rho_{u,c}$ and $\mathbf{d}_{u,c}$ respectively), must be

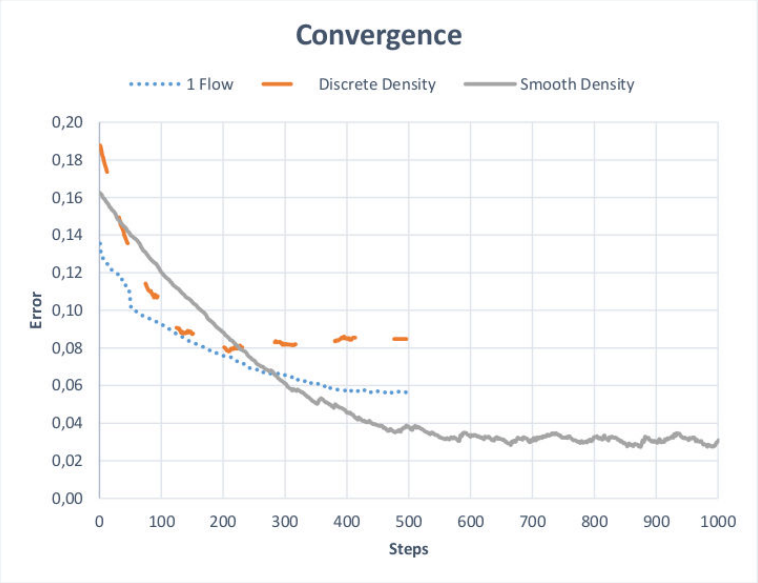

Figure 5.6: **Convergence** Algorithm 2 convergence for some of the experiments presented in Section 5.6.

computed for each patch. To speed up calculations, simplified trajectories are found that do not take into account collision avoidance; this simplification does not affect accuracy (Section 5.4.1). Accurate trajectories are calculated at the end of optimization using a Reciprocal Velocity Obstacles based algorithm described in Section 3.3.1.

Algorithm Convergence *(Alg. 1, line 3)* The proposed approach converges to a globally good solution (Equation 5.4) after a few minutes (Figure 5.6). Convergence depends on the initial configuration of each patch, the complexity of the user constraints, the number of paths updated at each step and and the number of requested characters. See Table 5.1 for the convergence time of the experiments discussed in Section 5.6.

5.5 User Interface

This section describes the user interface for specifying crowd density and direction requirements. The proposed approach is paint-driven and exploits tools familiar to artists such as brushes, selections, movement, gradients, strokes, etc. The system was integrated in an open source image processing tool [Harford, 2000] as a set of plugins. An artist can define a crowd by painting a set of grayscale layers on top of the environment map; these layers are essentially metadata having all the information needed to generate a crowd with density and direction controls. We define four basic types of layers (Figure 5.7): exogenous and endogenous density, direction and obstacles; a user can create as many layers of a specific type as she pleases. Additionally, layer and pixel opacity are used to assign weights to layers and pixels respectively.

Crowd Maps Exogenous and endogenous density layers are used to define the density of exogenous and endogenous characters respectively. Density is defined through the intensity of pixels; the brighter the values the higher the density. Direction is defined by drawing dark to bright gradients and finding the layer's $2D$ gradient. Finally, obstacle layers are used to easily mask out areas where density must be zero; this can also be achieved by careful painting of density. We found that this approach eases the process for various kinds of pre-existing environment maps where obstacles can be selected based on color.

Layer Merging Layers are then separately merged to generate three grayscale layers describing the final requirements for overall (a) exogenous density, (b) endogenous density and (c) direction. All direction and density layers are accumulated using weights based on pixel and layer opacity. Additionally areas where obstacles are present are removed.

From maps to crowd patches Having global merged layers, three additional parameters are defined: the desired number of characters, the crowd patches size and global period π. These parameters affect the quality and accuracy of the generated crowd. Each crowd patch is defined by a square set of pixels depending on maps resolution and patch size; desired density and direction for a patch are computed by the average of the pixel values. Empty patches are removed and a

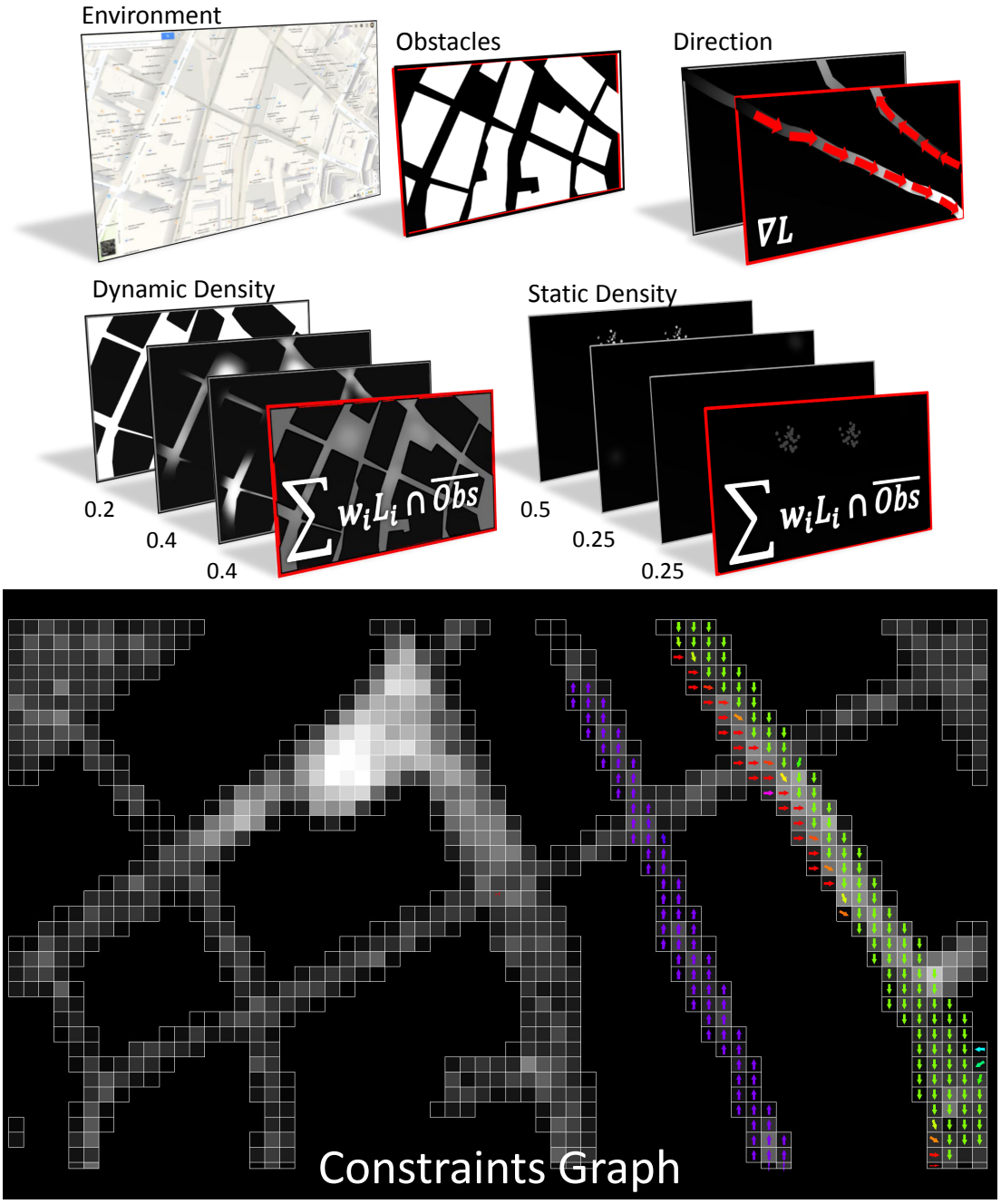

Figure 5.7: **Defining Constraints** Users can annotate an environment with various information using image layers; these include obstacles, exogenous and endogenous density and directions. Layers are separately accumulated to generate final constraints. (bottom) Finally, a graph of density and direction contraints is generated.

graph of interconnected patches is generated based on neighborhood information (Figure 5.7 bottom); this graph is given as input to the optimization approach described in Section 5.4 to generate the desired crowd. We found that users need only a few minutes to populate scenes such as the ones shown in Section 5.6.

5.6 Results

We evaluate our method according to different types of scenarios. First, simple scenarios explore how accurately our method actually match users' inputs; both for density and flow requirements. Secondly, we demonstrate our approach in typical use cases that include populating city like environments. Finally, we analyze the performance of our approach.

Please refer to the accompanying video[3] for animated versions of the paper results; this is especially useful for the flow experiments. We emphasize here that all resulting animations can be efficiently and endlessly played with user requirements *being constantly satisfied*; i.e., direction and flow direction remain constant over time and the optimized result is never violated. In comparison with previous approaches, only our method provides such a feature. For all presented examples, patches were defined so that patch area A ranged between 16 and $100m^2$.

5.6.1 Density Control

Our system is capable of generating crowds of different density requirements (Figure 5.8); to simulate different density patterns, users provide grayscale density maps, the required number of characters and the size of the area and patches. We remind that users can provide two kinds of density maps; exogenous and endogenous. Here we demonstrate moving crowds and not static which are trivial to handle.

The proposed system is capable of handling very diverse inputs such as simple uniform density (not shown), lanes of discrete density, smooth gradients and even

[3]Online version: `https://youtu.be/TUCr7zBRxOM`

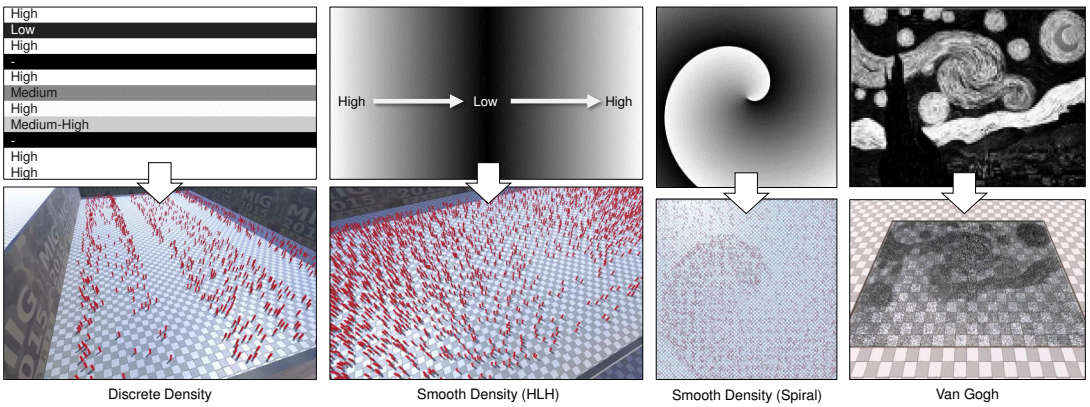

Figure 5.8: **Density Control** Our system can simulate crowds of different density patterns ranging from the very simple discrete density cases of a few thousand characters to the very complex ones (e.g., paintings) of hundreds of thousands. We note that characters move continuously between densities without violating the overall density requirements.

complex ones such as paintings with minimal errors (Figure 5.8). The demonstrated examples consist of crowds of different sizes, ranging from 1000 characters up to 100000 characters. We note that characters move around the environment between areas of high and low density and are not localized.

5.6.2 Flow Direction Control

Flow direction can easily be controlled with our approach. We set up three simple scenarios; a uniform density crowd with a flow constraint in a part of the crowd (Center Flow), a circular moving crowd (1-Circular Flow) and finally three circular motions (3-Circular Flows); in both of the circular motions there is a small number of characters moving in the remainder of the region without any direction constraint (Figure 5.9).

In **Center Flow**, uniform density is achieved and the characters in the middle follow the requested direction. Importantly, in this scenario, flows emerge in other parts of the environment that guide the characters from the end of the

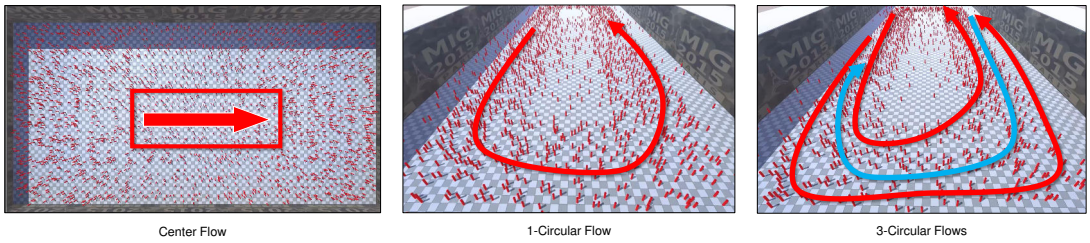

Figure 5.9: **Flow Direction Control** Our system allows for easy control of flow direction.

flow, around the flow and back at the beginning. In both **1-Circular Flow** and **3-Circular Flow**, circular lanes of characters are satisfied; characters in the areas without any flow constraints enter the lanes, follow them and either leave them to satisfy density constraints around the lanes or just follow the motions. These kind of control can be used to generate scenes such as strikes or people entering/leaving a train station.

5.6.3 Use Cases

Having demonstrated our system in typical scenarios, we can populate virtual environments with combinations of constraints. We demonstrate these results in two example scenarios; a single street around a park under different constraints (Figure 5.10) and a simulation of the Wall Street area in New York city (Figures 5.1 and 5.12).

Changing User Requirements A user can change the requirements for the same environment quite easily as demonstrated in Figure 5.10; here the scene is populated with a set of immobile characters (endogenous density) that are lying on the lawn and a set of moving characters with different flows around them. Moving characters avoid both exogenous and static ones.

Urban Examples It is also easy and intuitive to populate large city-like environments such as the one shown in Figures 5.1 and 5.12. We populate this environment with a non uniform crowd of 5000 characters and add direction constraints in some areas (Figure 5.12). Obstacles such as buildings are additionally

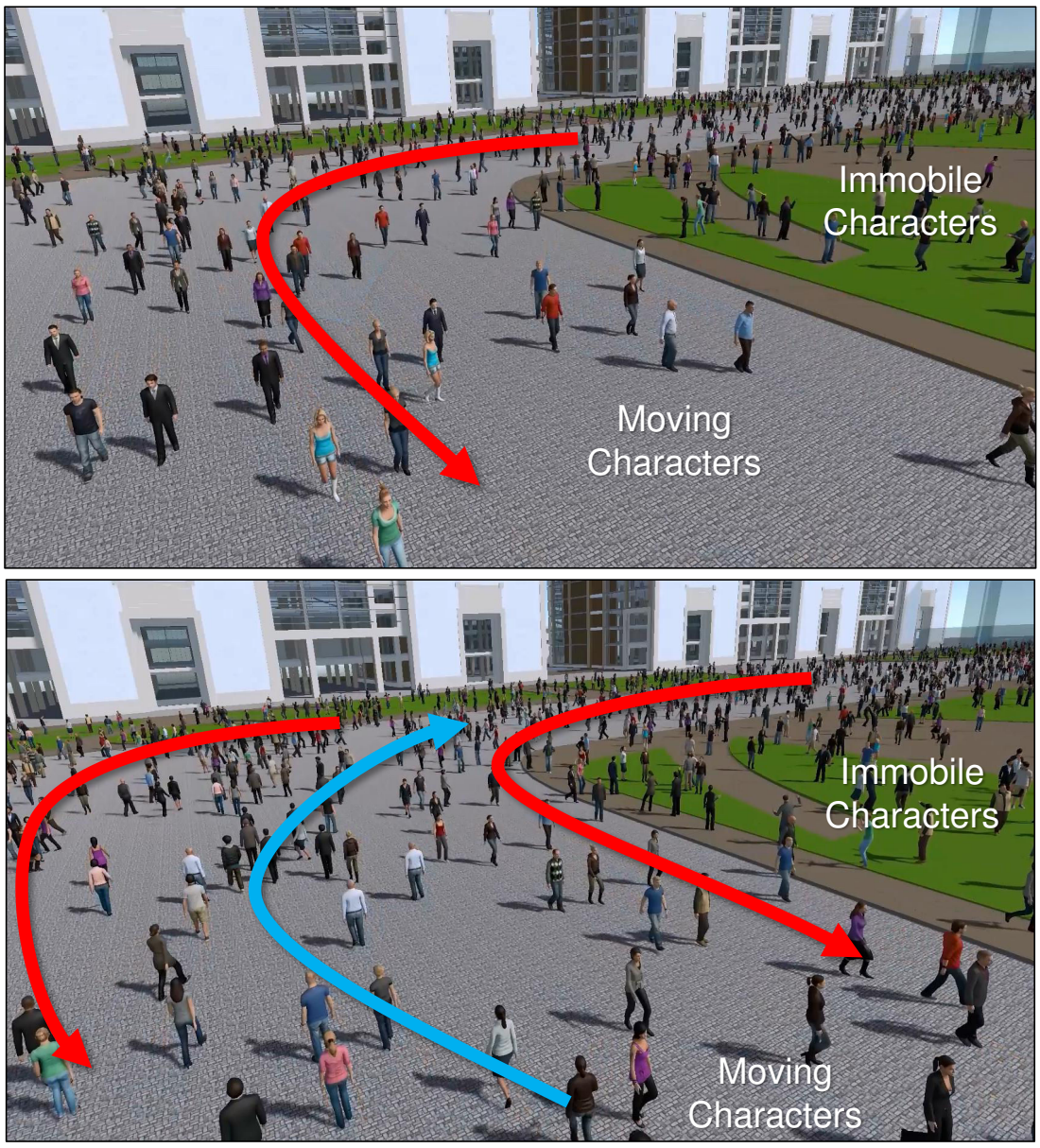

Figure 5.10: **Same environment, different constraints.** With the proposed system it is easy to change constraints in an area.

Figure 5.11: Two pictures of a district of Rennes generated with our method. Top: the vasselot street. Bottom: the Honoré Commeurec place.

marked through the user interface. Our system converges to a good solution satisfying density and flow constraints with minimal error in just a few steps.

Crowd sculpting also allows to populate a district of the city of Rennes, as shown in Figure 5.11. These results are the fruit of a collaboration between our laboratory and Dassault Systèmes company.

Massive Crowds Finally, we demonstrate that our system can scale to very large crowds of hundreds of thousands of characters under complex constraints such as the one in Figure 5.13. This scene consists of ∼100000 moving characters in an area of $0.8km^2$ (∼13000 patches) satisfying an image based density pattern. The system converges to density patterns that are close to the requested even though characters move in areas where the requested density was very low; this happens because of the high contrast between high and low density areas. Again, the resulting animation is endless and collision free.

5.6.4 Performance

The proposed framework can take up to a few minutes to find crowd patches configurations under user constraints (see Table 5.1); these results do not include the time to resolve internal collisions for the patches which is out of the scope of this work. We provide time for both optimization and resolving collisions using our velocity based implementation for completeness; notice that this time is significantly larger than the time to optimize patch parameters. Optimization time is affected by the number of patches, number of characters and the complexity of user constraints; it typically takes minutes. All of the performance measurements were collected on a 64-bit Linux based system having an 8 core Intel®Xeon(R) CPU E5-1620 clocked at 3.60GHz, with 16GB of RAM and a GeForce GTX 680/PCIe/SSE2 GPU card.

5.7 Discussion

A method to intuitively design crowd motion with simultaneous control of density and direction has been presented. These two quantities are essential to define

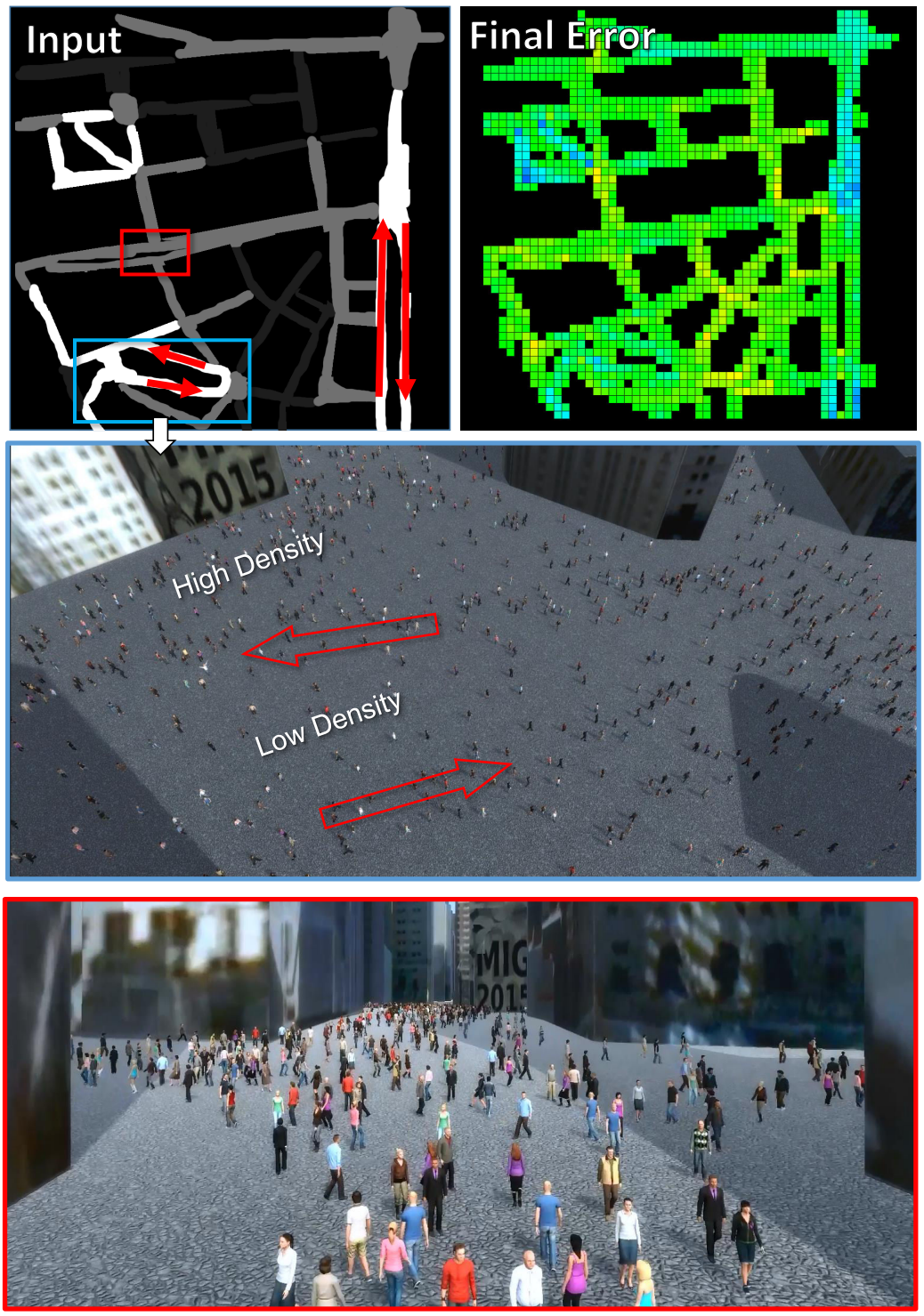

Figure 5.12: **City simulation** (top) User requirements and error in the optimization. (middle-bottom) Views of the final generated scene.

Scenario	Figure	Patches	Characters	Iterations	Time (s)	Internal Traj. Time (s)	Error
Density HLH	5.8	200	1097	276	8	184	0.0713
Discrete	5.8	160	2014	221	9	597	0.0697
Spiral	5.8	400	5379	310	27	1569	0.0457
1-Circular Flow	5.10	382	4825	340	22	268	0.0673
Wall Street	5.12	1587	5158	67	7	51	0.0671
MIG logo	–	2304	11601	135	36	1632	0.0376
Van Gogh	5.13	13132	95236	250	1181	~7 hrs	0.0756

Table 5.1: **Performance** for most of the experiments presented in this Chapter.

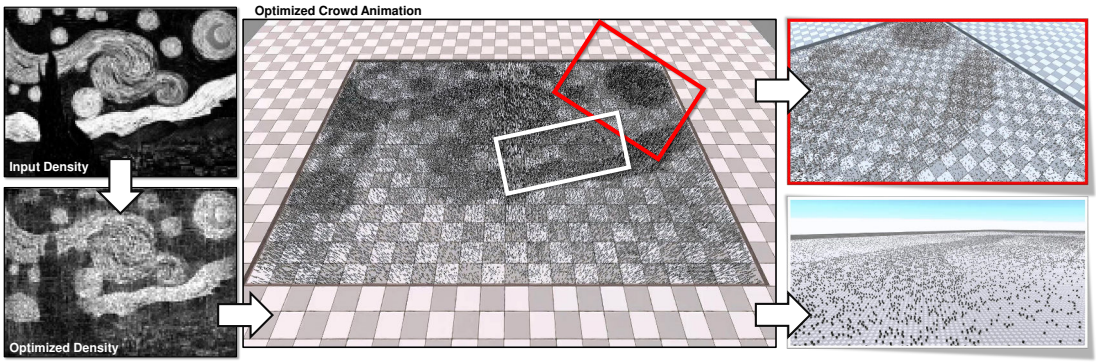

Figure 5.13: **Massive Crowds** Our system can handle complex density patterns based on chapitre3/fig or photographs such as this one based on a painting. We demonstrate here the generated patches' density and direction results for a massive crowd of 100000 characters in a $0.8km^2$ area. Notice that there is some error in the resulting density due to the complexity of the pattern. We note that the resulting animation is collision free and can be played real time (no rendering).

the visual aspect of crowd motion, but no previous methods allow users to easily control them over large-scale crowd motion: many of our demonstrations required minimal time to define and optimize. Our framework is robust to various kind of inputs such as simple user strokes, city maps or digital images and is particularly efficient to populate environments with ambient crowds of certain patterns such as cities and expressive artistic crowds. Users can create crowds using our prototype system in minutes, with no specific knowledge, no need to annotate environments and no need to tune complex sets of parameters. Additionally, generated crowds satisfy user constraints endlessly and not just for a short period of time.

However, satisfying user constraints endlessly can be a drawbacks if the goal is to reproduce a convincing crowd running an entire day. Indeed, during a day densities in the crowd evolve over time, but densities and flows of our resulting crowds remains the same. A next feature for this work could be finding transitions between several set of crowd patches, and add more motion variety in crowd patches.

The next chapter tackles these issues by proposing solution to avoid repetition of motion in crowd patches, and also allowing for external events to modify crowd patches content.

Chapter 6

Temporal Editing of Crowd Patches

Contents

This chapter presents solutions to add motion variety over time on crowd patches. Firstly, we introduce a method to avoid repetition of same motion over period of time, due to crowd patches paradigm. The key idea of this technique is to switch at the end of the period a crowd patch by another. These crowd patches get the same constraints, but different motion, to ensure animation continuity and non repetitive content. Secondly, we discuss a possible solution to make crowd patches reactive to external events, such as activity of a player. This is enabled thanks to a hybrid crowd patch/simulation engine prototype.

6.1 Crowd Patches Permutation

This Section presents a method enabling to permute crowd patches with another to avoid repetition of the motion over time. This method allows to add motion variety of motion in a given crowd patch without impacting animations in neighboring patches. This method is a part of the contribution of [Jordao *et al.*, 2014].

6.1.1 Overview

Crowd patches captures time-periodic animations. This enables endless replay of crowd motion. If a user carefully observes the same area for quite a long period of time, he may detect animation loop repetition. This is a drawback of the initial crowd patches we address here. In order to improve motion variety and lower the risk of animation loop detection, where needed (e.g., in the portion of the environment likely to be the most observed), we propose a mean to control the temporal variety. Our key-idea is to use different instances of patches at a given place instead of a unique one and play patch instances in sequence (patches permutation is performed at the end of the period). See Figure 6.1.

In a given patch, trajectories are deduced from two constraints:

- a set of space-time constraints at the limit of the patch: the patterns. Characters need to exit or enter patches through the space-time way-points

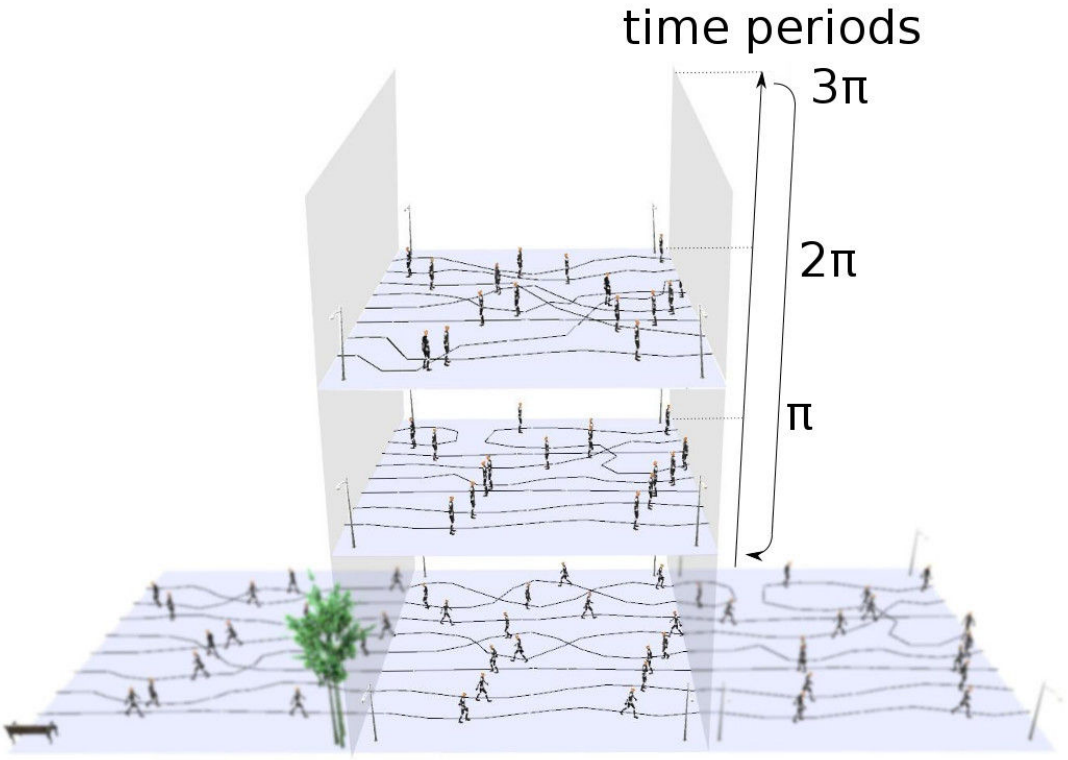

Figure 6.1: Time permutation: three different instances of a patch built from the same patterns and initial state are used to animate this specific area. Patch instances of same period are looped in sequence. Given that initial states are the same, the patch instances are seamlessly permuted at the end of each period.

as defined by patterns to enable smooth animations between two connected patches.

- the initial position of the characters in the patch at $t = 0$. Characters remaining inside the patch at the end of the period ($t = \pi$) need to be located at the same positions than the ones where some characters were at the beginning ($t = 0$). Note that different characters can occupy these same positions.

To enable time editing, we allow the possibility to enqueue different patch instances and play them sequentially (all have the same period). These motions are computed using the constraints of the initial patch, to ensure continuity between neighbor patches. As a result motions inside a patch appear longer (for they are not repetitive) and much more different than a patch with shorter period. Finally, we compute several *versions*[1] of motions for a given patch instance to offer diversity when sculpting a crowd patch in time. Several versions of a patch are represented in Figure 6.2.

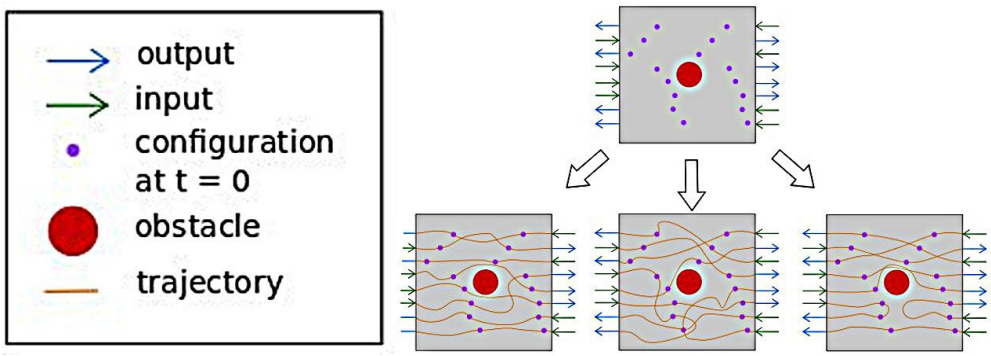

Figure 6.2: Three different versions of a patch with similar patterns (blue and green arrows) and time boundary conditions (purple points).

In the task of computing the animation trajectories of a given patch, the first step is to choose a mapping between each input and output space-time way-points

[1]Patches instance and patches versions are close notions. To help differientating them, in simple words, a patch instance is a set of spatiotemporal constraints that drive trajectories. A patch version is a patch with animation that satisfy those constraints. There can be several version of a same patch instance.

as defined by the four patterns forming a patch. This mapping is a great help for the generation of motion variety. Indeed, to compute several variants of a given patch instance, we simply randomize the way we connect input and output points over all the ones defined by patterns. An example of insertion of 3 different versions of a same patch is displayed in Figure 6.1. Instead of one period of time, the motion will look different for at least 3 periods of time (however characters enter and leave the patch at the same time and position). Note that the selection order for versions of the patch can be randomized, to get the repetition harder to notice. Moreover the repetition become even harder in a whole scene using this method.

6.1.2 Conclusion

With this method we can avoid the visual artifact of repetitive motion within a patch at each period. However, it could be interesting to realise an experimental study on the perception of this artifact. This would allow to estimate the efficiency of this technique. Moreover, the alternative patches need to be pre-computed. It offers few possibilities for user interactions. The next section presents the idea of a possible extension to get rid of this limitation.

6.2 Reactive Crowd Patches

So far, we developed several methods based on crowd patches, that ease the creation of populated environments by a designer. However, the crowd patches technique gets limited to non interactive application, because trajectories of crowd patches characters are precomputed. Consequently external characters or events can not modify behaviour of inner characters. Nevertheless, in a context of interactive applications, such as video games, characters motion should be influenced by player actions. By interacting with a non-player characters, the player can trigger events which modify their trajectories, such as talking to them or entering in collision with them. This section describes a solution enabling reactive crowd patches, i.e. adapting characters behaviour from a crowd patch according

to user's actions. This solution is an on-going work and remains unpublished.

6.2.1 Principle

The main principle of our approach is to combine two techniques to steer characters. First is the crowd patches one, the second is an existing classical simulation-based steering method. Practically, we use the Reynold's steering approach Reynolds [1999]. A character may then be steered from one or the other approach, and is respectively called a *patch-character* or a *reactive-character*. We may transfer characters from patch-based or simulation-based steering, and three cases may occur:

- *patch-character* can enter in interaction with an external element. The state of this *patch-character* (position, velocity, goal, etc...) is transferred to the simulator engine, preliminary set with obstacles of the environment. A new *reactive-character* is initialized in the simulation system. Consequently, it is the simulator engine which handles the interactions with the player: the *reactive-character* path is adapted to the user's actions. Moreover, because crowd patches play pre-computed data, a character transferred in the simulation engine is not deleted in the crowd patches but marked as hidden. These hidden *patch-characters* are still animated by crowd patches, but not displayed on the screen.

- By entering in the simulation engine, a character animated by the simulation, called *reactive-character* leave a *free-slot* in the crowd patches system. This last becomes the goal for the *reactive-character*. He or she can be transferred back to the crowd patches system if and only his or her position is identical to his or her *free-slot* goal.

- A last case happens when an animated character, handled by the crowd patches or the simulator, becomes definitely out of the animation system (the player kill him or her). This character become a *free-slot* if he or she is handled by crowd patches, or deleted if is handled by the simulation engine.

Animated characters can have different behavior depending on the interaction with the player. If they need to avoid the player, they are steered by the *free-slot* as goal. The interaction may last longer (the player talk with them), so the interaction must be over before chasing their goal. Finally, if the player kill them (a common situation in video games), the characters have to be permanently removed form the animation system.

To steer *reactive-characters* correctly, the simulation engine gets information about neighbour animated characters as well as environment geometry from the crowd patches system: *patch-characters* positions is retrieved and synchronized with the simulation. Note that the crowd patches structure allows doing this in an efficient manner by limiting this synchronisation to the closest neighbors of the *reactive-characters* (the patch containing the *reactive-characters* as well as its 8-connected patches).

6.2.2 Results and Discussion

This on going work is a solution to get background characters in a large interactive environment. This is due to two main components: (i) crowd patches, which are able to efficiently handle crowds composed of many characters, (ii) and a simulation engine which is able to solve local interactions. Our current technique allows to operate several interactions with non-player characters, which are avoiding the player, stopping to talk with him or her, and dying. One of these interactions is presented in Figure 6.3.

These first results are encouraging but many points can be improved. The current version of the method is implemented with a steering behaviour [Reynolds, 1999] simulation approach. This choice was made to quickly develop a first version of the algorithm. Recent simulation system could replace our basic one, for better motion quality when a reactive-character avoid a collision. Moreover, the current version of the method set as a goal for reactive-character the free-slot he or she just left. Possible other free-slots could be a better option for more realistic behaviour. For instance, the free-slot goal of a reactive-character could be far away, specially if he or she just talked to the player, whereas possible other free-slot could be

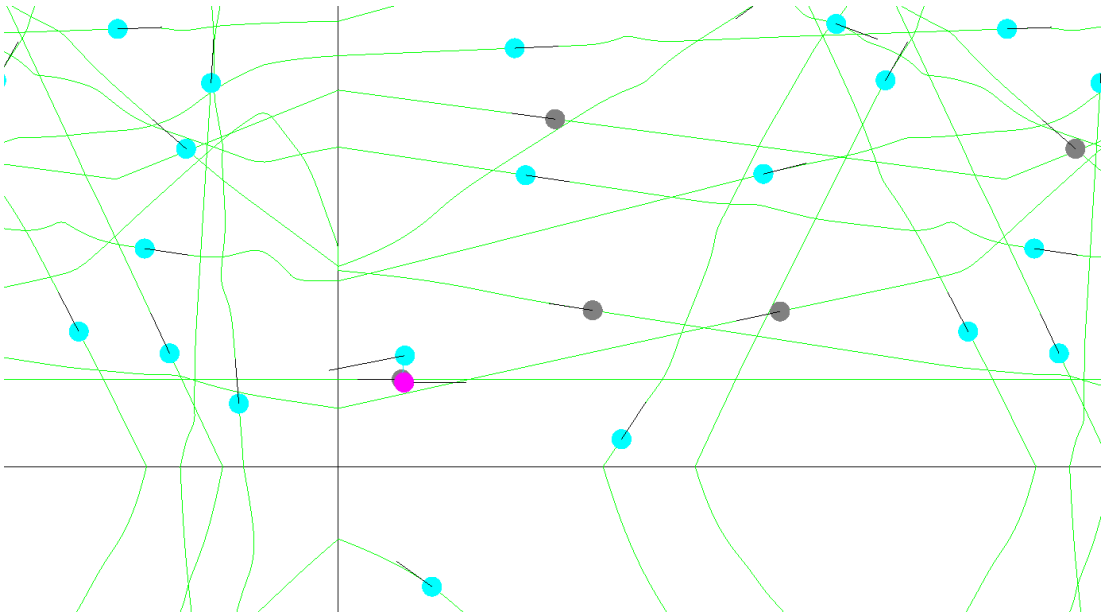

Figure 6.3: **Reactive Crowd Patches**. **Pink**: the player. **Light-blue**: patch-characters. **Dark-blue**: reactive-character. **Grey**: free slot. **Black lines**:the borders of crowd patches. The player get in collision with a patch-character. This last become a reactive-character and avoid the player and try to get the slot it just lost. Some others free-slots are available in the scene. (May be because the player killed them..)

closer.

6.3 Conclusion

In this chapter, we presented two techniques that break temporal repetition of crowd patches. The first enabling permutation of different crowd patches. The second is an on-going work allowing to make crowd patches responsive to actions of a player. These work are still in progress and present several limitations. More details on future directions are discussed in the conclusion of this thesis.

Chapter 7

Conclusion

The main goal of this thesis was to provide expressive tools to intuitively control the design of animated of crowds, in large environments. This is an important problem which had too few attention before. The methods we developed can be used to generate endless animation of very large crowds. This contributes to design of more lively virtual worlds.

7.1 Contributions and discussion

This thesis led to two main contributions in the computer graphics and animation domain, as published as full papers in international conferences. The first one, Jordao *et al.* [2014], is a new method using a sculpting metaphor to edit crowd motion in space and time. The second one, **?**, is a paint tool enabling to design a crowd based on density and main direction of motion. A detailed review of these works and of their current limitations is provided in the following sub-sections.

7.1.1 Crowd sculpting

Crowd sculpting is the first method for interactively sculpting everlasting crowd motion at a large spatial scale. In addition to intuitive spatial control through deformation gestures, the method for temporal editing is one of the first solutions for providing interactive visual control of temporal content. This work enables

non-specialists to quickly shape lively environments. It therefore represents a large step towards making authoring tools more widely available.

In the current version, our approach however displays several limitations, discussed next.

Extensions to multi-scale control: Our approach provides the users with control over the spatial layout of patches and enables local edits of motions in time. This control could be completed by larger or lower scales of control: at larger scales, it would be relevant to control global parameters, such as the evolution of global density in time (to reproduce daily cycle of a living city). At lower scales, it would be interesting to edit some individual trajectories, while automatically preserving the integrity of the patch (space and time constraints on the trajectories, typically entry and exit points).

Trajectories lengthening & shortening: The different bending, stretching or shrinking operations on patches result in similar variations on the animation trajectories. Typically, this impacts the speed of walking characters (given that timing of entry and exit points in a patch are not changed). In our implementation, the maximum increase in speed is fixed to $\times 1.5$ the initial speed, a value above which node insertion is performed, bringing the speed back to $\times 0.66$ $\times 1.5$ the initial speed. The longer the strip of stretched patches, the smaller the proportion of speed change (for example 10 patches turned into 11 patch yield a $\times 1.10$ speed increase). However, such changes in speed may induce possible artifacts in animations (such as characters slowing down in large, straight, ways). An easy solution would be to extend the number patch types by including shorter or longer, rectangular shaped patches.

Environments and animation content: Though we illustrated our approach with examples of pedestrian streets, our method is not limited to these environments. Green parks, public squares, public buildings, etc. are easily integrated by first, defining the set of rest states and second, mapping and computing the corresponding types of patches. For example, parks and squares would require a

2D stretching mode in addition to the 1D stretching used for pedestrian streets.

For populating very large environments, an improvement would be to provide a semi-automatic technique offering a balance between automated and interactive steps, *e.g.* automatically integrating the topological constraints of the environment, while offering all our other interactive features. For example, a designer would sculpt a crowd along the geometry of existing sidewalks and building walls, with the system automatically fitting the required width for the patch along these sidewalks and walls.

Linear Deformation of patches: As detailed in Section 4.3.3, local deformations of patches are computed using a bilinear interpolation technique. While other techniques (such as group motion editing Kwon *et al.* [2008]) could be used to increase the quality of the resulting trajectories, the low computational cost of bilinear interpolations enables the simultaneous deformations on a very large number of patches.

7.1.2 Crowd Art

The crowd art method we presented is the first approach that proposes to populate large environments by controlling crowd densities and main directions of motion. This method is based on crowd patches produces crowds which satisfy both constraints over time.

However, the results of optimization process are not computed in real-time. We can only offer an intuitive, but not interactive, interface to design constraints map. Our interface is not as expressive as the one of crowd sculpting.

In the future, more diverse ways to enhance crowd control should be concidered, such as more diverse flow directions inside patches (e.g., bi-directional or cross sections); this can be achieved for example by having multiple directions in each patch. Another possible improvement is adding sources and sinks (buildings, subway stations, etc.); this will remove circular paths and improve convergence. Importantly, we are considering conducting user studies to asses the quality or results and the user experience of our approach both by naive and expert users.

Another direction for future work is to consider an approach to find an optimal set of convex crowd patches that fit precisely obstacles and user requirements.

Finally, we would like to be able to transition between different constraints to model situations like different hours of the day for the same places; e.g., parks, business areas, etc. These transitions should look natural and be continuous, so a careful method of morphing between patches should be considered.

7.2 Future Directions

7.2.1 Authoring tool to control crowds over time

Despite this thesis and other techniques which bring new possibilities for crowd design, expressive modeling for crowds is still at its beginning. In both crowd sculpting and crowd art, the extension through temporal control of animations was evoked. A first direction for a future work would consist in finding and expressive method to create transition between two crowds (composed of crowds patches or not). We can imagine a simple process as linear transition between two (or more) scenes of crowd patches, generated at first with crowd art. Another idea could be to create a more expressive method based on a paint interface. Some time brushes, for instance a brush for the morning, the afternoon and the evening, could paint densities for specific period of the day, and the system would automatically compute transitions between these periods.

7.2.2 Reactive Crowd Patches

Crowd patches get good properties such as being scalable to large environment and producing endless animation of crowd. Thank to this this thesis, they are now enhanced with expressive method to easily design populated crowd patches environments. However in interactive application like video games, crowd patches are not relevant. Indeed, crowd patches store motion data, so a player character can not interact with a patch character, because the patch character has no knowledge about external characters. A direction of research could be to improve

and extend the on-going work presented in Section 6.2, to make crowd patches reactive to external components of the environment. We can imagine a simulation system mixing the efficiency of crowd patches to run a huge background crowd in real time, with a modern crowd simulator to process interactions between elements from the environment and those from crowd patches. It would enable to interact with characters of crowd patches for game applications. For instance, by making them stop for a while, or even to make them disappear. This contribution would encourage crowd patches in applications using crowds, since the drawback of the non reactivity of crowd patches would be overcome.

Author's publications

Articles

Technical papers

TP1. **K. Jordao**, P. Charalambous, J. Pettré, M. Christie, M.-P. Cani. "Crowd Art: Density and Flow Based Crowd Motion Design". *Motion In Games 2015*.

TP2. **K. Jordao**, J. Pettré, M. Christie, M.-P. Cani. "Crowd Sculpting: A space-time sculpting method for populating virtual environments". *EUROGRAPHICS 2014*, Volume 33 (2014), Number 2.

Posters

P1. **K. Jordao**, J. Pettré, M.-P. Cani. "Density-controlled crowds". *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2014).*

Short papers

SP1. **K. Jordao**, J. Pettré, M.-P. Cani. "Interactive techniques for populating large virtual cities". *Eurographics Workshop on Urban Data Modelling and Visualisation (2013).*

Bibliography

Ahn, Junghyun, Wang, Nan, Thalmann, Daniel, & Boulic, Ronan. 2012. Within-crowd immersive evaluation of collision avoidance behaviors. *Pages 231–238 of: Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM.

Bayazit, O Burchan, Lien, Jyh-Ming, & Amato, Nancy M. 2003. Better Group Behaviors in Complex Environments using Global. *Artificial Life 8*, **8**, 362.

Bokeloh, Martin, Wand, Michael, Koltun, Vladlen, & Seidel, Hans-Peter. 2011. Pattern-aware shape deformation using sliding dockers. *Page 123 of: ACM Transactions on Graphics (TOG)*, vol. 30. ACM.

Bokeloh, Martin, Wand, Michael, Seidel, Hans-Peter, & Koltun, Vladlen. 2012. An algebraic model for parameterized shape editing. *ACM Trans. Graph.*, **31**(4), 78:1–78:10.

Braun, A., Musse, S.R., de Oliveira, L.P.L., & Bodmann, B.E.J. 2003 (May). Modeling individual behaviors in crowd simulation. *Pages 143–148 of: Computer Animation and Social Agents, 2003. 16th International Conference on.*

Charalambous, P., & Chrysanthou, Y. 2014. The PAG Crowd: A Graph Based Approach for Efficient Data-Driven Crowd Simulation. *Comp. Graph. Forum*, **33**, 95–108.

Chenney, Stephen. 2004. Flow tiles. *Pages 233–242 of: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Emilien, Arnaud, Vimont, Ulysse, Cani, Marie-Paule, Poulin, Pierre, & Benes, Bedrich. 2015. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM transactions on Graphics, Proceedings of ACM SIGGRAPH*, Aug., 11.

Fiorini, Paolo, & Shiller, Zvi. 1998. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, **17**(7), 760–772.

Gu, Qin, & Deng, Zhigang. 2013. Generating freestyle group formations in agent-based crowd simulations. *Computer Graphics and Applications, IEEE*, **33**(1), 20–31.

Guay, Martin, Cani, Marie-Paule, & Ronfard, Remi. 2013. The Line of Action: an Intuitive Interface for Expressive Character Posing. *ACM Transactions on Graphics*, **32**(6), Article No. 205.

Guay, Martin, Ronfard, Rémi, Gleicher, Michael, & Cani, Marie-Paule. 2015a (June). Adding dynamics to sketch-based character animations. *In: Sketch-Based Interfaces and Modeling (SBIM) 2015*. Proceedings of sketch-based interfaces and modeling.

Guay, Martin, Ronfard, Rémi, Gleicher, Michael, & Cani, Marie-Paule. 2015b. Space-time sketching of character animation. *ACM Transactions on Graphics (TOG)*, **34**(4), 1.

Guy, Stephen. J., Chhugani, Jatin, Kim, Changkyu, Satish, Nadathur, Lin, Ming, Manocha, Dinesh, & Dubey, Pradeep. 2009. ClearPath: Highly Parallel Col-

lision Avoidance for Multi-agent Simulation. *Pages 177–187 of: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New York, NY, USA: ACM.

Guy, Stephen J, Lin, Ming C, & Manocha, Dinesh. 2010. Modeling collision avoidance behavior for virtual humans. *Pages 575–582 of: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 2-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems.

Harford, Alex. 2000. *GIMP essential reference*. New Riders professional library. Indianapolis, IN: New Riders.

Heigeas, Laure, Luciani, Annie, Thollot, Joëlle, & Castagné, Nicolas. 2003 (Sept.). A Physically-Based Particle Model of Emergent Crowd Behaviors. *Pages 1–9 of: Graphicon 2003 - 13th International Conference on Computer Graphics*. Proceedings of Graphicon 2003. Session "Physically Based Simulation".

Helbing, Dirk. 1998. A fluid dynamic model for the movement of pedestrians. *arXiv preprint cond-mat/9805213*.

Helbing, Dirk, & Molnár, Péter. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E*, **51**(May), 4282–4286.

Helbing, Dirk, Farkas, Illes, & Vicsek, Tamas. 2000. Simulating Dynamical Features of Escape Panic. *Nature*, **407**(6803), 487–490.

Henderson, LF. 1971. The statistics of crowd fluids. *Nature*, **229**, 381–383.

Henderson, LF. 1974. On the fluid mechanics of human crowd motion. *Transportation research*, **8**(6), 509–515.

Jin, Xiaogang, Xu, Jiayi, Wang, Charlie CL, Huang, Shengsheng, & Zhang, Jun. 2008. Interactive control of large-crowd navigation in virtual environments using vector fields. *IEEE Computer Graphics and Applications*, 37–46.

Jordao, Kevin, Pettré, Julien, Christie, Marc, & Cani, M-P. 2014. Crowd sculpting: A space-time sculpting method for populating virtual environments. *Pages 351–360 of: Comp. Graph. Forum*, vol. 33. Wiley Online Library.

Ju, Eunjung, Choi, Myung Geol, Park, Minji, Lee, Jehee, Lee, Kang Hoon, & Takahasi, Shigeo. 2010. Morphable Crowds. *Pages 140:1–140:10 of: Proc. of ACM SIGGRAPH Asia*. SIGGRAPH Asia '10. ACM.

Kim, Jongmin, Seol, Yeongho, Kwon, Taesoo, & Lee, Jehee. 2014. Interactive Manipulation of Large-scale Crowd Animation. *ACM Trans. Graph.*, **33**(4), 83:1–83:10.

Kim, Manmyung, Hyun, Kyunglyul, Kim, Jongmin, & Lee, Jehee. 2009. Synchronized Multi-character Motion Editing. *Pages 79:1–79:9 of: ACM SIGGRAPH 2009 Papers*. ACM.

Kim, Manmyung, Hwang, Youngseok, Hyun, Kyunglyul, & Lee, Jehee. 2012. Tiling motion patches. *Pages 117–126 of: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Kwon, Taesoo, Lee, Kang Hoon, Lee, Jehee, & Takahashi, Shigeo. 2008. Group motion editing. *ACM Trans. Graph.*, **27**(3), 80:1–80:8.

Lamarche, Fabrice, & Donikian, StÃ©phane. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, **23**, 509–518.

Lee, Kang Hoon, Choi, Myung Geol, & Lee, Jehee. 2006. Motion patches: building blocks for virtual environments annotated with motion data. *Pages 898–906 of: ACM Transactions on Graphics (TOG)*, vol. 25. ACM.

Lee, Kang Hoon, Choi, Myung Geol, Hong, Qyoun, & Lee, Jehee. 2007. Group Behavior from Video: A Data-driven Approach to Crowd Simulation. *Pages*

109–118 of: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Lemercier, S., Jelic, A., Kulpa, R., Hua, J., Fehrenbach, J., Degond, P., Appert-Rolland, C., Donikian, S., & Pettré, J. 2012. Realistic Following Behaviors for Crowd Simulation. *Comput. Graph. Forum*, **31**(2pt2), 489–498.

Lerner, Alon, Chrysanthou, Yiorgos, & Lischinski, Dani. 2007. Crowds by Example. *Computer Graphics Forum*, **26**(3), 655–664.

Li, Yi, Christie, Marc, Siret, Orianne, Kulpa, Richard, & Pettré, Julien. 2012. Cloning Crowd Motions. *Pages 201–210 of:* Lee, Jehee, & Kry, Paul (eds), *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*. Lausanne, Switzerland: Eurographics Association.

Metoyer, Ronald A., & Hodgins, Jessica K. 2004. Reactive pedestrian path following from examples. *The Visual Computer*, **20**(10), 635–649.

Milliez, A., Wand, M., Cani, M.-P., & Seidel, H.-P. 2013. Mutable elastic models for sculpting structured shapes. *Computer Graphics Forum*, **32**(2pt1), 21–30.

Milliez, Antoine, Noris, Gioacchino, Baran, Ilya, Coros, Stelian, Cani, Marie-Paule, Nitti, Maurizio, Marra, Alessia, Gross, Markus, & Sumner, Robert W. 2014. Hierarchical Motion Brushes for Animation Instancing. *Pages 71–79 of: Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*. NPAR '14. New York, NY, USA: ACM.

Moussaïd, Mehdi, Helbing, Dirk, Garnier, Simon, Johansson, Anders, Combe, Maud, & Theraulaz, Guy. 2009. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society of London B: Biological Sciences*.

Musse, S.R., & Thalmann, D. 1997. A Model of Human Crowd Behavior : Group Inter-Relationship and Collision Detection Analysis. *Pages 39–51 of:* Thal-

mann, Daniel, & van de Panne, Michiel (eds), *Computer Animation and Simulation â€™97*. Eurographics. Springer Vienna.

Narain, Rahul, Golas, Abhinav, Curtis, Sean, & Lin, Ming C. 2009. Aggregate dynamics for dense crowd simulation. *Page 122 of: ACM Trans. on Graphics (TOG)*, vol. 28. ACM.

Ondřej, Jan, Pettré, Julien, Olivier, Anne-Hélène, & Donikian, Stéphane. 2010. A Synthetic-vision Based Steering Approach for Crowd Simulation. *Pages 123:1–123:9 of: ACM SIGGRAPH 2010 Papers*. ACM.

Paris, S., Pettré, J., & Donikian, S. 2007. Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach. *Eurographics'07: Comp. Graph. Forum*, **26**((3)), 665–674.

Patil, S., van den Berg, J., Curtis, S., Lin, M.C., & Manocha, D. 2011. Directing Crowd Simulations Using Navigation Fields. *Visualization and Computer Graphics, IEEE Transactions on*, **17**(2), 244–254.

Pettré, Julien, Laumond, Jean-Paul, & Thalmann, Daniel. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *Page 194 of: First International Workshop on Crowd Simulation*, vol. 43. Citeseer.

Pettré, Julien, Ciechomski, Pablo de Heras, Maïm, Jonathan, Yersin, Barbara, Laumond, Jean-Paul, & Thalmann, Daniel. 2006. Real-time navigating crowds: scalable simulation and rendering. *Computer Animation and Virtual Worlds*, **17**(3-4), 445–455.

Pettré, Julien, Ondřej, Jan, Olivier, Anne-Hélène, Cretual, Armel, & Donikian, Stéphane. 2009. Experiment-based Modeling, Simulation and Validation of Interactions Between Virtual Walkers. *Pages 189–198 of: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New York, NY, USA: ACM.

Ramirez, Jose Guillermo Rangel, Lange, Devin, Charalambous, Panayiotis, Esteves, Claudia, & Pettré, Julien. 2014. Optimization-based Computation of

Locomotion Trajectories for Crowd Patches. *Pages 7–16 of: Proceedings of the Seventh International Conference on Motion in Games*. MIG '14. New York, NY, USA: ACM.

Reynolds, Craig W. 1987. Flocks, herds and schools: A distributed behavioral model. *Pages 25–34 of: SIGGRAPH '87: Proc. of the 14th annual Conf. on Comp. Graph. and Interactive Techniques*. New York, NY, USA: ACM.

Reynolds, Craig W. 1999. Steering Behaviors For Autonomous Characters. *Pages 763–782 of: Game Developers Conference*.

Shao, Wei, & Terzopoulos, Demetri. 2007. Autonomous pedestrians. *Graph. Models*, **69**(5-6), 246–274.

Sorkine, Olga, & Alexa, Marc. 2007. As-Rigid-As-Possible Surface Modeling. *Pages 109–116 of:* Belyaev, Alexander, & Garland, Michael (eds), *Eurographics Symposium on Geometry Processing*. Barcelona, Spain: Eurographics Association.

Thalmann, Daniel Thalmann, & Musse, Soraia Raupp. 2007. *Crowd Simulation*. British Library.

Treuille, Adrien, Cooper, Seth, & Popović, Zoran. 2006. Continuum Crowds. *Pages 1160–1168 of: Proc. of ACM SIGGRAPH 2006*. SIGGRAPH '06. ACM.

Ulicny, Branislav, Ciechomski, Pablo de Heras, & Thalmann, Daniel. 2004. Crowdbrush: interactive authoring of real-time crowd scenes. *Pages 243–252 of: Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

van den Berg, J., Lin, Ming, & Manocha, D. 2008a (May). Reciprocal Velocity Obstacles for real-time multi-agent navigation. *Pages 1928–1935 of: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*.

van den Berg, Jur, Patil, Sachin, Sewall, Jason, Manocha, Dinesh, & Lin, Ming. 2008b. Interactive navigation of multiple agents in crowded environments.

Pages 139–147 of: Proceedings of the 2008 symposium on Interactive 3D graphics and games. ACM.

van der Berg, Jur, Guy, StephenJ., Lin, Ming, & Manocha, Dinesh. 2011. Reciprocal n-Body Collision Avoidance. *Pages 3–19 of:* Pradalier, CÃ©dric, Siegwart, Roland, & Hirzinger, Gerhard (eds), *Robotics Research*. Springer Tracts in Adv. Robotics, vol. 70. Springer Berlin Heidelberg.

Yersin, Barbara, Maïm, Jonathan, Pettré, Julien, & Thalmann, Daniel. 2009. Crowd patches: populating large-scale virtual environments for real-time applications. *Pages 207–214 of: Proceedings of the 2009 symposium on Interactive 3D graphics and games*. I3D '09. New York, NY, USA: ACM.

List of Figures

List of Tables

Résumé étendu

.1 Contexte

La recherche en animation de foules est devenue très active ces dernières décénies, grâce entre autre, aux succès des jeux vidéos, du cinéma et l'exploration d'environnements 3D. Les foules jouent un rôle primordiale dans ce genre de média.

Dans les films, les animations de foules d'arrière plan, tel que des pétions discuttant ou marchant, permmettent de plonger les acteurs principaux dans un monde vivant. Le film Astérix et Obélix : au service de sa majesté ou série Game of Thrones utilisent un logiel de conception de foules Golaem Crowd, pour afficher des foules de supporters de rugby ou une population paniquée, essayant d'echaper a une attaque d'un dragon. L'animation de foule est aussi très populaire pour les scène de batailles de grand evergure, comme c'est le cas dans le film Le seigneur des anneaux : le retour du roi, quand Minas Tirith est attaquée par des hordes d'urukai. Les foules de d'arrière plan ou de scène de bataille permettent de renforcer le sentiment d'immersion du spectateur. Par conséquent, la foule doit avoir un comprtement qui correspond au sénario, que ce soit sur ses déplacements ou sur ces interactions locales, pour ne pas perturber les spectateurs.

Dans les jeux videos, il est commun d'avoir des villes entierement modélisées à échelle réel, peuplées de personnages virtuels. C'est par exemple le cas pour le jeux Assassin's creed Unity, ou le joueur a la possibilité d'explorer les moindre recoins de la ville de Paris 1789, pendant la révolution française, dont les rues sont innondées de manifestants. Les foules de ce jeux permet une grande immersion dans ce contexte geo-politique. De plus, dans des jeux en ligne tel que Star Craft II ou Age of Empire Online, le joueur peut contrôler de maniere directe le mouvement de ses troupespour attaquer les adversaires. Un bon contrôl des unités est crutial pour le game-play de ces jeux.

Une evolution possible dans le domaine de l'architecture et du tourisme pourrait être d'intégrer des foules dans des villes modélisées en 3D. Pour le cas de projet urbain, les maquettes virtuelles pourrait etre accompagnée de foules pour avoir un meilleur ressenti du quartier future ou de la ville. Dans le cas du tourisme

on pourrait imaginer du tourisme virtuel dans des villes existante ou fantastique peuplées de manière realiste par des foules de personnages. Ces foules permettraient d'avoir un bon ressenti de l'ambiance et de l'atmosphere qui pourrait regner dans ces villes virtuelles. On peut imaginer une extension future du très connus explorateur du monde, Google Earth, intégrant des foules pour peupler les villes du monde entier. Le type de foules attendues pour ce domaine, sont des foules capable d'être animées sur une longue période de temps, pour être capable de visionner une partie du monde à n'importe quel moment de la journée. De plus, les foules devront être de taille astronomique, pour permettre à l'utilisateur d'explorer une ville dans son ensemble s'il le souhaite.

Que ce soit dans les jeux videos, le cinéma, ou l'exploration d'environnement 3D, les foules ont un grand impact sur le media finale proposé par ces domaines. Cependant, la creation de tels foules avec un contrôl précis du comportement et du mouvement reste très complexe avec les techniques actuelles. La principale méthode pour pour créer des foules consiste à utiliser un algorithme de simulation de foules, qui est une boite noire prennant en entrée un jeux de paramètre défini par l'utilisation et qui produit un mouvement de foules en conséquence. La difficulté principale de ce genre d'approache est que l'utilisateur n'a pas un contrôle directe sur le mouvement qu'il créé et que l'utilisation de tels système requière une expertise dans le domaine. Les techniques modernes consiste à faire un premier essaie avec un jeux de paramètre, puis d'ajouter petit à petit le jeu de paramètre pour obtenir le mouvement de foule souhaité. Le temps passé pour concevoir de tels mouvement peut être très long et couteux en therme de dévelloppement. Cette complexité croit avec le nombre d'agents dans la foules que l'on souhaite.

.2 Enoncé du problème

Le processus de conception interactive de foules, même pour des environnements de tailles moyennes, comme une zone de batiments, n'est pas une tâche facile et pose deux problèmes majeur. Premièrement, une foule est un système intrasec complèxe, contenant des personnages qui interagissent entre eux à l'echelle locale et globale, qui ont des comportement de groupe ou individuel et qui effectuent

des actions en rapport avec l'environnement dans lequel ils evoluent. La creation de foules realistes et complèxe demandent l'utilisation de simulation de foules sophistiqués et de modèles d'animation complèxe pour lesquel l'utilisateur doit spécifier et ajuster un grand jeux de paramètres via la répétition, générallement longue, de séries de génération et de testes. Deuxièment, le degrés de contrôle requis pour facilement et interactivement peupler de grand environnement demande la conception de nouveaux outils capable de manipuler dans l'espace (où positionner la foule, comment contrôler les flux et les densités des personnages) et dans le temps (comment une foule peut changer dans le temps en terme de comportement, flux et densités).

Les techniques basées sur des simulations de foules n'offrent seulement un contrôle indirecte et générallement globale sur la foule via la manipulation des paramètres des agents, et sans retour visuel immédiat. Ces types de techniques ne sont pas efficaces pour définir des subtilités dans le moiuvement de foule, tel que le chemin d'un personnage ou sa forme dans le temps. Cependant, ce type de contrôle est essentiel pour transmettre l'individuallité à la foule ou pour permettre à un joueur d'achever une tâche précise dans un jeux vidéo.

Une autre approche conciste à directement deformer la ou les trajectoires d'un ou plusieurs personnages de la foule, en la ou les déformant, la ou les tordant, le ou les étirant, ou les raccourciçant, pour satisfaire à la fois les désire de l'utilisateur et les contraintes de l'environnement. Cependant, de trop grosses dééformation mènnent à des comportement non réaliste, comme des accélérations subistes des personnages. De plus, quand la foule devient trop grande, un nombre important d'opérations doit être effectué par le concepteur pour atteindre son résultat. De plus, ces méthodes deviennent de plus en plus lente en fonction du nombre de personnage dans la foule, car des collisions inter-personnages doivent être résolues.

.3 Contributions

Le but de cette thèse est de developper des outils permettant à un utilisateur de concevoir des foules tout en fournissant un contrôle directe sur l'aspect visuel, le comportement locale et globale des personnages, à travers une application

interactive et artistique. Ces techniques doivent être capable de gérer des foules de grande tailles avec une animation sans fin. Par rapport a ce dernier point, nous savons que les Crowd Patches, des morceaux pré-calculés d'animations de foules qui peuvent être assemblées ensemble pour former une population vivante, est une méthode efficace pour produire de vastes foules animées sans fin. Par contre, cette technique souffre d'un manque d'interactivités pour créer, situer et connecter un ensemble de crowd patches entre eux; et d'un manque de contrôle pour créer un ensemble de crowd patches ayat de bonnes propriétés.

Dans cette thèse, nous proposons trois méthodes pour contrôler de vastes foules à travers des applications interactives ou à interfaces artistique

- **Crowd sculpting** : une nouvelle approche pour concevoir de manière interactive de complèxe foules animées dans des environnements virtuels, avec un haut niveau de contrôle par gestes et un retour visuel immédiat. Les animations générées sont joués sans fin, grâce à la méthode des crowd patches sur laquelle notre méthode s'appuie.

- **Crowd art** : une technique d'optimisation pour calculer des animations de foules qui satisfont des contraintes de densité et de flux localement. L'algorithme est capable d'éviter des boucle locales dans les trajectoires des personnages pour éviter que ceux ci ne tournent en rond, grâce a notre système permettant de lier par un chemin les crowd patches avec les plus grosses erreures. La méthode fournie un outil artistique pour concevoir les différentes contraintes. L'utilisateur peut créer des foules très rapidement en utilisant un logiciel de painture deja existant. Les exigences de l'utilisateur sont spécifiés au système en combinant les différents calques de contraintes (pour la densité, les flux, obstacles...)

- **Foules variant dans le temps** : une extension qui améliore la diversité des mouvement dans les patches de foules, grâce a un méchanisme de permutaton des crowd patches. Etant donné qu'un patche de foule capture une animation périodique, nous proposons un méthode permettant de créer différentes versions d'un patche de foule le tout en respectant ses contraintes

d'entré/sortie de personnage et de période, et de remplacer par une autre version d'un patche quand le premier arrive à la fin de sa periode. Les mouvement sont donc plus diversifiés a l'interieur d'un patche.

.4 Organisation du document

Cette thèse est organisée de la manière suivante. Le Capitre 2 est dédié au travaux existant sur la simulation de foules, sur la conception et le contrôle de tel foules, et sur la conception 3D grâce des outils hauts niveaux utilisant des métaphors artistiques. Les avantages et les inconvegnants de chaque méthodes sont discutés, et une conclusion est formulée à la fin en concentrant sur ce qui a été fait et pas fait dans le domaine.

Les méthodes développées pendant cette thèse se basent toutes sur la méthode des crowd patches. Une description détaillée sera faitte dans le Chapitre 3. Une méthode non publiée sur la génération de trajectoires périodiques, spécialisé pour les crowd patches, sera présenté.

Ensuite dans le Chapitre 4, nous abordons le problème de la modelisation spatial de foules. Nous présentons une méthode interactive pour peupler de manière simple des environnements via des gestuels intuitives, capable de de former des foules de densité uniformes dans des environnement 3D.

Dans le chapitre 5, nous présentons une méthode capable de créer de vaste foules en maitrisant la densité constante dans le temps et les flux. Les contraintes de densité et de flux sont spécifié par l'utilisateur en utilisant une interface de type artistique pour facilité le contrôle de ces paramatres.

Ensuite dans le Chapitre 6, nous présentons plusieurs méthodes pour créer de la varité de mouvement dans le temps. Premièrement, nous décrivons comment éviter la répétition des animations dans un crowd patch. Deuxiement, nous présentons une extension possible des crowd patches ou ceux-ci deviendrait reactif aux actions d'un utilisateur et aux changements de l'environnement.

Finalement, dans le Chapitre 7, nous formulons une conclusion de ce travaille, discutons des limitations et des directions de recherche future de cette thèse.

.5 Publications

.5.1 Technical papers

TP1. **K. Jordao**, P. Charalambous, J. Pettré, M. Christie, M. P. Cani. "Crowd Art: Density and Flow Based Crowd Motion Design". In Proceedings of *Motion In Games* (2015). ACM.

TP2. **K. Jordao**, J. Pettré, M. Christie, M. P. Cani. "Crowd Sculpting: A space-time sculpting method for populating virtual environments". In *Computer Graphics Forum* (2014), Volume 33, Number 2.

.5.2 Posters

P1. **K. Jordao**, J. Pettré, M. P. Cani. "Density-contrôlled crowds". In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2014).

.5.3 Short papers

SP1. **K. Jordao**, J. Pettré, M. P. Cani. "Interactive techniques for populating large virtual cities". In Proceedings of the *Eurographics Workshop on Urban Data Modelling and Visualisation* (2013) pp. 41-42. Eurographics Association.

.6 Conclusion

Le but principal de cette thèse est de fournir des outils expressifs pour contrôler la conception des animations de foules dans de vastes environnements. C'est un problème concéquant qui n'a eu que peu d'attention auparavant. Les méthodes que nous avons développées peuvent être utilisées pour générer des animations sans limite de temps de foules. Ce qui contribue dans le domaine de la conception de mondes virtuels plus vivant.

.6.1 Contributions et discussion

Cette thèse mène à deux contributions dans le domaine de l'informatique graphique et de l'animation, publiés en tant que papier long dans des conférences internationnales. La première, Crowd Sculpting, est une nouvelle méthode utilisant une métaphor de sculpture pour éditer une foule dans l'espace et le temps. La seconde, Crowd Art, est un outil de painture capable de concevoir une foule en se basant sur sa densité et ses flux. Une revue détaillée de ces travaux et de leurs limites sont exposés dans les sous parties suivantes.

.6.1.1 Crowd sculpting

Crowd sculpting est la première méthode permettant de sculpter de manère interactive un mouvement de foule pour des vastes environnemts. En plus du contrôle spatial intuitif via les gestuels de déformation, la méthode permet d'éditer dans le temps tout en gardant un contrôle visuel du contenu. Ce trvail permet au utilisateurs non expérimenter de facilement prendre en main un outil de conception de foules, et de créer rapidement des environnements vivants. Cette contribution représente un pas en avant vers la fabrication d'outils de crétions expréssifs.

Dans sa version actuelle, notre approche possède cependant plusieurs limites qui sont discutées dans la suite.

Extensions vers un contrôle multi-echel : Notre approche fourni à l'utilisateur des contrôles sur le plan spatial des patches et sur des modifications locales du mouvement dans le temps. Ces contrôles pourraient être complété par des contrôles plus ou moins grandes échelles. Il pourrait être intéressant de contrôler des paramètres plus généraux comme l'évolution dans le temps de la densité (dans le but de produire des cycles journaliés). A une echelle plus petite, il pourrait être intéressant de contrôler finement la trajectoire d'un personnage, tout en preséservant automatiquement l'intégrité des crowd patches (contraint générallement en espace temps sur les etrés sorties du patche).

Agrendissement et rétrécissement des trajectoires : Les opérations de torsions, d'étirements, de rétrécissements, on pour effet diverses variations sur l'animation des crowd patches. Typiquement, cela impact la vitesse des person-

nages qui marchent (entant donné que l'entré et la sortie d'un personnage est fixé par le patche). Dans notre implémentation, la vitesse peut augmenter au maximum de 1.5 la vitesse initial, depassé cette valeur un nouveau patche est inséré ramenant la vitese à 0.66 la vitesse initial. Plus longue est la suite de crowd patches, plus petite sera la deformation, car chaque patche recevra une part homogene de la déformation globlale. Par exemple, une suite de 10 patches aura sa vitesse deformé au maximum de 1.1. Cependant, de tel changement en vitesse peut impliqué des artefacts dans l'animation, comme des personnage allant subitement plus vite. Une solution pourrait être d'inclure des patches de différentes tailles pour intervertir un patch par un autre un peu plus grand quand celui ci est déformé.

Contenu des environnement et des animation : Bien que nous ayons illustré notre approche par des exmeples de rues pietonnes, notre méthode ne se limite pas à ça. Les parcs, les places publiques, les centre commerciaux, etc. sont facilement intégrable en définissant un ensemble d'états de repos pour la grammaire des patches aproprié au sénario, puis en associant et calculant les types crowd patches correspondant. Par exemple, les parcs et places publiques pourraient avoir besoin d'un mode de sculpture 2D en plus du mode 1D, c'est a dire faire evoluer la sculpture par carré en plus du mode ligne.

Pour peupler de très grands environnements, une amélioration pourrait être de fournir une technique semi-automatique permettant de préremplir de foules l'environnement en prennant en compte sa topologie, l'utilisateur n'aurait plus qu'a s'occuper des ajustements et des comportement spécifiques. Par exemple un artiste pourrait culpter une foule le long de la géométrie des troitoires d'une ville grâce au systeme semi automatique ces patches seraient directement bien placé le long de ces trotoires.

Déformation linéaire des patches Comme décrit dans la section 4.3.3, les déformations locales d'un patche sont calculées utilisant une technique d'interpolation bilinéaire. Alors que d'autres méthodes, tel que celle utilisée dans group motion editing de Kwon, pourrait être utilisée pour améliorer la qualité des trajectoires, cependant le faible cout de calcule de la technqiue par interpolation bilinéaire nous permet d'avoir du temps réel sur des foules composées de millier de patches.

.6.1.2 Crowd Art

Le méthode crowd art que nous avons présentée est la première méthode qui propose de peupler un environnement en contrôle a la fois la densité et les flux de la population. Cette méthode est basée sur la technqiue des crowd patches et produit des foules qui satisfont ces contraintes dans l'espace et le temps.

Cependant, les resultats de la boucle d'optimisation peuvent mettre plusieur sminutes avant d'être calculés. Nous pouvons seulement offrir un contrôle intuitif mais pas interactif sur les foules que l'on concoie. De ce fait notre interface n'est pas expressive comme celle de crowd sculpting.

Par la suite, plusieurs pistes pour améliorer le contrôle sur la conception de la foule doivent être pris en concidération, comme le fait d'avoir plusieurs flux possible dans un patche ou bien en permetant l'ajout de sources et de puits pour permettre aux personnages d'entrer et de sortir de l'environnement. Ce dernier ajout pourrait permettre d'éviter les chemins circulaires des trajectoires des personnages. Une étude utilisateur pourrait être menée pour confirmer la qualité des résultats.

Une autre direction pour un travail future est de concidérer une approche pour trouver un ensemble fini de crowd patches qui collent parfaitement avec la topologie de l'environnement.

Finallement, nous aimerions être capable de contrôler des transitions dans le temps d'un crowd art à un autre. Ce qui permettrait de modéliser des situations comme l'évolution d'une journée dans une ville. Ces transitions devrait être naturelles et continues.

.6.2 Directions future

.6.2.1 Outil de création pour contrôler la foule dans le temps

Bien que cette thèse et d'autres technique existent pour offrir de nouvelles possibilités pour concevoir des foules, la modélisation expressive de foule en est toujours à son début. Que ce soit pour crowd sculpting ou pour crowd art, l'extension des modèles à travers une dimension temporel a été évoquée. Une première direc-

tion pour les travaux future pourrait concister à trouver une méthode expressive pour créer des transitions entre foules foules. Nous pouvons imagner un procédé simpliste qui se contenterai d'une transition linéaire entre les deux. Une autre idée pourrait être de créer une méthode plus expressive basée sur une interface de painture. Des pinceau temporel, comme un pinceau matinal et un nocturne, pourrait peindre différent moment de journée et le système déterminerai automatiquement les transitions entre ces différent moment.

.6.2.2 Crowd patches reactifs

Les crowd patches possèdent de bonnes propriétés comme être très bien adapté aux grands environnements et de produire des animations sans fin. Et grâce à cette thèse ils sont maintenant renforcé de méthodes expressive pour facilement concevoir des foules avec eux. Cependant dans les apllication interactive comme les jeux vidéos, les crowd patches ne sont pas pertinants. En effet, les crowd patches collectent les données du mouvement, donc un joueur ne peut peut pas interagir avec la foule car les personnages du patch n'ont aucune connaissance sur les personnages non patches ou joueurs. Une direction de recherche pourrait être de developper et d'étendre le travail en cours présenté dans la Partie 6.2, pour rendre les crowd patches reactifs aux composants externes de l'environnement. Nous pouvons imaginer un système de simulation de foule mélangeant un simulation moderne avec les crowd patches, ce qui permettrait d'obtenir à la fois l'immensité des foules obtenues par crowd patches et la reactivité des simulateur. Cela permettrait à un joueur d'interagir avec les personnages des crowd patches pour un jeux vidéo. En les faisant s'arrêter par exemple. Ou en les faisant disparaître. Cette contribution encouragerait l'utilisation des crowd patches dans plus d'applications, etant donné que l'inconvéniant des crowd patches serait levé.

Abstract

Crowds are increasingly present in audio-visual media, such as movies or video games. They help to strengthen the immersion of the subject in the virtual environment. However, creating crowds is most of the time based on models hard to master and which do not offer a direct control on the motion that you want to create. In this thesis we propose contributions for designing crowd motions through interactive and intuitive tools.

Firstly, we present an interactive method for designing the crowds by distorting it like clay. The user can stretch, compress and twist the overall shape of the crowd to give it the shape he or she wishes. The inner characters of the crowd automatically adapt to the new shape imposed by the user.

Secondly, we present a method to paint the motion and the density of the crowd to create it. We offer the opportunity to the user to create crowds by painting a grayscale density map and a motion map by gradients. Its colored maps are transformed by our system to crowds, thanks to our iterative algorithm seeking to optimize the different values of colored maps.

Crowds obtained by these methods can occupy a very large space and are animated indefinitely. Unlike conventional methods of creating crowds, that are based on the adjustment of model parameters, our methods allow to design crowd motions based on higher level features of the crowd, as its overall shape, its internal movement or density. This offers the possibility to simply, quickly and intuitivly create animated crowd contents.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Interactive design and animation of crowds for large environments

Nom Prénom de l'auteur : JORDAO KEVIN

Membres du jury :
- Monsieur BOULIC Ronan
- Madame PELECHANO Nuria
- Madame CANI Marie-Paule
- Monsieur PETTRÉ Julien
- Monsieur ARNALDI BRUNO
- Monsieur CAZIER David

Président du jury : Bruno ARNALDI

Date de la soutenance : 21 Décembre 2015

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état
~~Thèse pouvant être reproduite après corrections suggérées~~

Fait à Rennes, le 21 Décembre 2015

Le Directeur,

M'hamed DRISSI

Signature du président de jury

Résumé

Les foules sont de plus en plus présentes dans les médias grands publics, comme le cinéma ou les jeux vidéo. Elles permettent de renforcer l'immersion du sujet dans l'environnement qui lui est présenté. Or, la création de mouvement de foule est la plus part du temps basé sur des modèles dures à prendre en main et qui n'offrent pas un contrôle direct sur le mouvement de foule que l'on souhaite créer. Dans cette thèse nous proposons des contributions sous forme de méthodes pour concevoir des mouvements de foules par le biais d'outils interactifs et intuitifs.

Dans un premier temps, nous présentons une méthode interactive permettant de concevoir des foules en les déformant comme de l'argile. L'utilisateur peut tirer, compresser et torde la forme global de des foules pour leurs donner la forme qu'il ou elle souhaite. Les personnages qui composent la foule s'adaptent automatiquement à la nouvelle forme imposée par l'utilisateur.

Dans un second temps, nous présentons une méthode permettant de peindre les mouvements et la densité de la foule pour la créer. Nous offrons la possibilité à l'utilisateur de créer des foules en peignant une carte de densité en niveau de gris, et une carte de mouvement via des dégradés. Ses cartes de couleurs sont utilisées par notre système pour le transformer en un mouvement de foule, via un algorithme itératif cherchant à optimiser les différentes valeurs des cartes de couleurs. Les foules obtenues par ces méthodes peuvent occupées un espace très large, et sont animées indéfiniment.

Contrairement aux méthodes classiques de création de foules qui se basent sur l'ajustement de paramètres de modèles, nos méthodes permettent de concevoir les mouvements de foules en se basant sur des caractéristiques plus hauts niveaux de la foule, comme sa forme globale, ses mouvements internes ou sa densité. Ce qui offre la possibilité de créer du contenu de foule animée de manière simple, rapide et intuitif.

Abstract

Crowds are increasingly present in audio-visual media, such as movies or video games. They help to strengthen the immersion of the subject in the virtual environment. However, creating crowds is most of the time based on models hard to master and which do not offer a direct control on the motion that you want to create. In this thesis we propose contributions for designing crowd motions through interactive and intuitive tools.

Firstly, we present an interactive method for designing the crowds by distorting it like clay. The user can stretch, compress and twist the overall shape of the crowd to give it the shape he or she wishes. The inner characters of the crowd automatically adapt to the new shape imposed by the user.

Secondly, we present a method to paint the motion and the density of the crowd to create it. We offer the opportunity to the user to create crowds by painting a grayscale density map and a motion map by gradients. Its colored maps are transformed by our system to crowds, thanks to our iterative algorithm seeking to optimize the different values of colored maps. Crowds obtained by these methods can occupy a very large space and are animated indefinitely.

Unlike conventional methods of creating crowds, that are based on the adjustment of model parameters, our methods allow to design crowd motions based on higher level features of the crowd, as its overall shape, its internal movement or density. This offers the possibility to simply, quickly and intuitively create animated crowd contents.

N° d'ordre : 15ISAR 41 / D15 - 41

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coëmes • CS 70839 • F-35708 Rennes Cedex
Tel : 02 23 23 82 00 - Fax : 02 23 23 83 96