Interactive Control of Multi-Agent Motion in Virtual Environments

Joseph Henry

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2016

Summary

With the increase in the power of computers, simulation of large crowds in films and computer games has become much more prevalent. With this comes a need to control various aspects of these crowds, including their collective movement and formation and the interactions between the individual characters in the crowd and their environment. Controlling these factors helps an animator to convey a story, or a player to complete tasks in a computer game.

Existing methods require a lot of time and user input, across multiple steps, to define the different levels of behaviour in a crowd simulation. Such user control is restricted by hardware devices that accept a limited amount of input information. This often results in approaches using specific mouse clicks or touch screen gestures to produce only a selection of crowd behaviours. This results in a discrepancy between the amount of input a user can provide and the number of characters they are controlling in a crowd simulation. This mismatch means there is no direct link between the individuality or expressiveness in the user's interaction, and the resulting crowd behaviour.

In this thesis we present two approaches to solve the problems outlined above. In our first approach, users manipulate an intermediary shape on a touch screen that defines the movement of the crowd over time. Our simulation adapts the user's control to fit with the virtual environment and moves the characters in the crowd to follow this control as efficiently as possible. A user study shows that crowds can complete tasks more efficiently when controlled with our scheme as compared with a mouse-based control. We extend this approach to allow characters to interact with their environments by negotiating obstacles. Our simulation accounts for how this interaction affects the speed of the characters' movement when providing them with instructions to follow the user's control.

In our second approach, we infer the crowd's movement based on a user's input to a touch screen device instead of requiring the user to explicitly move a shape on screen or perform a set of controls. We do this by considering the similarity of a new user input to a set of example inputs, taken from real users, that each correspond to a particular crowd movement. Given a set of similar user inputs from our database, we can generate new crowd motion that reflects a user's control by blending together the crowd motion examples that the inputs are paired with. In this way, the generated crowd motion is directly affected by variations in the user's input, providing a user with greater freedom to define the animation.

Abstract

With the increased use of crowd simulation in animation, specification of crowd motion can be very time consuming, requiring a lot of user input. To alleviate this cost, we wish to allow a user to interactively manipulate the many degrees of freedom in a crowd, whilst accounting for the limitation of low-dimensional signals from standard input devices. In this thesis we present two approaches for achieving this: 1) Combining shape deformation methods with a multitouch input device, allowing a user to control the motion of the crowd in dynamic environments, and 2) applying a data-driven approach to learn the mapping between a crowd's motion and the corresponding user input to enable intuitive control of a crowd.

In our first approach, we represent the crowd as a deformable mesh, allowing a user to manipulate it using a multitouch device. The user controls the shape and motion of the crowd by altering the mesh, and the mesh in turn deforms according to the environment. We handle congestion and perturbation by having agents dynamically reassign their goals in the formation using a mass transport solver. Our method allows control of a crowd in a single pass, improving on the time taken by previous, multi-stage, approaches. We validate our method with a user study, comparing our control algorithm against a common mouse-based controller. We develop a simplified version of motion data patches to model character-environment interactions that are largely ignored in previous crowd research. We design an environment-aware cost metric for the mass transport solver that considers how these interactions affect a character's ability to track the user's commands. Experimental results show that our system can produce realistic crowd scenes with minimal, high-level, input signals from the user.

In our second approach, we propose that crowd simulation control algorithms inherently impose restrictions on how user input affects the motion of the crowd. To bypass this, we investigate a data-driven approach for creating a direct mapping between low-dimensional user input and the resulting high-dimensional crowd motion. Results show that the crowd motion can be inferred directly from variations in a user's input signals, providing a user with greater freedom to define the animation.

Acknowledgements

I would like to begin by thanking my supervisor Taku Komura for giving me the opportunity to pursue research in an area I found both exciting and stimulating. I would also like to thank Michael Herrmann, and my collaborators on the work in this thesis, Hubert Shum and He Wang for their advice and support that has helped to keep me on track. An extra thank you to Myung Geol Choi for providing some of the motion data used in the experiments.

After many long years trying to get this thesis complete I have made a number of friends who have supported me both in and out of my research life. I am very thankful for all of my fellow PhD students and IPUBbers, especially Patrick, Adam, Peter, Vlad, and Xi who not only were able to provide valuable advice but were also great friends throughout. I also would like to give a big thanks to my family whose continued support has enabled me to follow my chosen path. Hopefully now they can see what I have been doing all these years.

Finally, I would like to thank my partner Kayleigh, who was great throughout but without her I don't think I would have been able to get through the final gruelling months of writing up. I am convinced that without her support there would have been times I would have just become a gibbering wreck. I definitely owe her more than one.

Declaration

Table of Contents

List of Figures

Chapter 1

Introduction

Crowd control research has become increasingly popular due to its potential applications in computer games and animation. In modern films, crowd animation is used in large-scale scenes to make the scene more active and interesting. Crowd animation can be employed in the background of a scene to depict, for example, a busy town centre in which the protagonists are situated. Alternatively, the scene may focus on the crowd itself, for example when depicting a battlefield scene. In both cases it is important that the crowd motion be appropriate to the scenario so that the audience is not distracted by unusual movement of the characters. Additionally, during creation of the animation the collective crowd and individual character motion should be controllable, allowing the animator to govern how the crowd moves through the environment in order to convey the storyline and atmosphere of the scene. In interactive applications, such as computer games, appropriate control of a crowd of characters can help to immerse a user in the game, increasing overall enjoyability and enabling the user to perform appropriate tasks effectively. This is evident in real-time strategy games such as *StarCraft 2* and *Age of Empires Online*, where controlling military units to attack the opponents is a key criterion for success in the game.

Despite the popularity of crowd animation in the film and game industry, specification of detailed crowd movement often requires large amounts of input and time. Although recent research proposes methods for alleviating some of this burden, multiple stages are still required to achieve the desired output. This can mean that production of a scene is costly and time-consuming. On top of this, existing approaches use input devices to indirectly alter the crowd's movement by specifying constraints on the simulation, rather than using the device as a means to express the motion of the crowd. As a result, a user's experience when creating an animation or playing a computer game is diminished due to the lack of direct interactivity.

In this chapter, a brief overview is given of techniques used to produce crowd movement, particularly with regards to control of group formation and motion and its application in interactive systems such as video games. We provide reasons for why effective, realtime control is difficult to achieve, and why this is important for use in current applications. A description of the problem is provided along with an outline of the research contained in this thesis that offers solutions to these issues.

1.1 Demand for Interactive Crowd Control

The main issue with controlling a crowd in animation and interactive applications comes from its high degrees of freedom, in the case where each character is able to move around freely in the environment. There is a discrepancy between this high-dimensional movement of a crowd and the low-dimensional signals from user input devices. To cope with this, a common approach for controlling crowds is to use high-level commands to specify goals to a group of characters as opposed to goals for individual characters. While this is effective for producing general crowd movement, in most cases a user is unable to specify low-level subtleties in the crowd movement as the crowd is in motion, such as making changes to the crowd shape whilst it is moving or specifying the routes individuals should take through the scene. These changes in formation and character pathing are useful for conveying individuality to the crowd motion or allowing a user to achieve certain objectives in applications such as computer games. Often, specification of such details occurs in a secondary process, where additional constraints are defined in order to manipulate the original simulation of the crowd.

To produce fine details in crowd motion, some crowd animation approaches allow a user to manipulate the individual trajectories or behaviours of every character in the crowd. Crowd simulation software like *Massive* requires the animators to carefully design the behaviour of the characters, such as programming their synthetic sensors and effectors, in order to control the formation of a crowd [Kanyuk (2009)]. For small groups of characters this can be effective, however, for increasing sizes of crowds this process becomes cumbersome. Moreover, the macroscopic behaviour of the crowd can be unpredictable as it is not always clear how the individual behaviours of the characters will cause them to interact in larger crowds. This can lead to greater workload during animation production. Furthermore, this kind of approach can adversely affect a user's experience in interactive applications. For example, in real-time strategy games, players have to control individual units using mouse gestures, which consist of multiple clicks and drags, to be able to specify complex motion and formations of a group of characters. In such applications, these extra steps can hinder a user's enjoyment due to the long time it takes to create the desired crowd movement.

In crowd animation techniques, users manipulate the movement of the crowd indirectly by defining keyframed formations or applying constraints to existing crowd movement. These algorithms require multiple steps for designing the crowd movements, especially when environmental obstacles are involved. Other approaches adopt a "point and click" style, where agents in a crowd are first selected and then a goal position is specified. As a result, the user has little influence over how the crowd moves to their target unless they specify a large number of waypoints to define the intermediate movement of the crowd. In interactive applications, this extra requirement to provide multiple instructions to move the crowd can mean that it is harder for a user to react quickly to new events. In many instances it would be desirable for a user to be able to specify the formation and motion of the crowd directly, influencing its shape and behaviour throughout the course of the crowd's movement. This is particularly helpful if a user wants to control a crowd to perform different movements simultaneously to achieve a certain task. For example, in order to improve the level of coverage of a given environment the user might want to control the crowd to move and to spread out at the same time. It is also useful if a user needs to adapt the crowd's formation quickly in response to a change in their surroundings or, for example, to gain the upper hand against an opposition team in virtual sports or war games.

When using interactive applications a key aspect of the enjoyment for the user, particularly in computer games, comes from the feeling of being able to directly control the movement of characters or manipulate objects in the game. This can be seen through the popularity of Nintendo's Wii console and Microsoft's Kinect device. This sense of being able to influence virtual characters and environments in realtime helps the user to immerse themselves in the application and improves the overall enjoyment for the user. Direct control is also important for allowing the movement of the crowd to correspond well to a user's input. This is prevalent given that a user's experience can be diminished when there is a disconnect between their input and the generated crowd motion that is seen on the screen. Furthermore, direct, intuitive control methods make an application easier to use. Providing this kind of interaction for applications enables users to quickly learn to generate crowd movement as well as to more easily convey their desired crowd behaviour. Both of these aspects have a positive influence on a user's experience.

As games become more advanced due to increases in computational power, virtual environments are populated with greater numbers of obstacles and characters are expected to perform more interesting motions that involve interacting with surrounding objects. In this sense, the complexity of crowd simulation is a key feature in its visual and interactive appeal. Previous group control approaches often do not consider the existence of the rest of the environment. For those that do, the obstacles avoidance and environment interaction is either performed in an offline, iterative manner or the algorithm is abstracted to a group-wide level. To produce more realistic looking scenes a crowd should be able to interact directly with their surrounding environment. Such interactions can include how the shape of a crowd will be affected when encountering an obstacle as well as how individual characters interact with environmental obstacles that can alter their movement, including objects to crawl under or jump over. To allow the application to remain interactive, this character-environment interaction must be handled in real-time. Previous research can simulate the interaction of characters with environmental objects however, because this simulation requires a number of constraints, these approaches do not allow a user to directly influence the characters' interactions or to provide interactive control over their movement. An algorithm is required that incorporates character-environment interactions whilst still respecting a user's instructions for the movement of the crowd. To do this, a crowd control scheme should account for the effect of obstacles on the movement of characters when following a user's control signal. This is especially important for real-time games where unforeseen perturbations could cause characters to become stuck if they are unable to adapt their motion.

1.2 Problem Definition

The goal of this thesis is to provide a method for realtime control of the motion of a crowd in interactive applications. Throughout this thesis the words "character" and "agent" will be used interchangeably to mean an individual, intelligent entity in the crowd and the words "user" and "animator" to mean the individual controlling the motion of the crowd. We define an interactive application to mean a piece of software to which a user can apply a control and expect to see an update to the application almost immediately. In this section we define the main issues highlighted in section

1.1 and provide an overview of how the work in this thesis provides solutions to these problems.

1.2.1 Realtime Control of Crowd Formation and Movement

Although previous research has addressed the control of crowd motion and formation specification, these methods still require a number of stages in order to produce the final crowd movement. This prevents interactive control over crowd motion and can increase the time required for specification of crowd movement. An algorithm is required that allows a user to define the shape and motion of a crowd simultaneously so that the appropriate crowd motion can be generated in a single step. This will benefit users in interactive applications by speeding up the process of producing crowd movement. As a result, users will be able to control crowds more effectively to achieve a desired formation and guide the crowd to carry out tasks in dynamic virtual environments.

This thesis tackles the problem of providing a realtime control scheme for crowd motion by harnessing the capabilities of multitouch devices. A multitouch device lets a user provide several inputs simultaneously, thus allowing them to supply control for both the movement of the crowd and its shape at the same time. We define a crowd's target formation using a two-dimensional mesh that can be directly manipulated by a user's touch inputs in order for a user to be able to specify the crowd's movement and shape. This mesh is also affected by the surrounding environment so as to prevent characters in the crowd from colliding with obstacles. The combination of these two signals means that the target formation for the crowd is defined by the user whilst respecting the constraints of the scene. The characters' final positions in the user-defined mesh are reassigned each frame based on minimising the travel time for the whole crowd to reach the target formation. In doing this, our approach can create crowd motion that follows the user's input well in a variety of static and dynamic environments. Our algorithm lets a user focus on the high-level crowd behaviour by handling the low-level interactions between the environment and the crowd's formation. More details can be found in chapter 3.

1.2.2 Control of Crowd Motion in Complex Environments

In animation and computer games, crowds are often required to move in detailed environments in order to convey a certain storyline or to provide interesting scenarios for users to interact with. Simulating the interactions between characters in the crowd and their environment, such as climbing over or crawling under obstacles, helps to improve the quality of the animation or gaming experience. However, incorporating such interactions whilst still allowing a user to interactively control a crowd is an unsolved problem. Beyond simple collision avoidance, no scheme exists for considering the effect of environment interactions on a character's ability to follow the control input provided by a user.

This thesis proposes a method for incorporating a variety of character-environment interactions into the planning and motion of a user-controlled crowd. The method proposed in chapter 5 replaces the basic Euclidean distance metric used to assign positions in the user-defined formation in chapter 3 with an environment-aware cost metric that accounts for a variety of different objects in the scene. This metric considers the effect of both traditional impassable obstacles that a character cannot pass through at all, and traversable obstacles that a character can travel through but, in doing so their movement speed is affected. Examples of traversable obstacles include walls that a character can climb over or a low net that a character must crawl under. The algorithm in chapter 5 uses values from motion capture data of a character's interaction with these types of environmental obstacle. This information is embedded into the scene using data "patches" that define the area in which an obstacle will affect a character's movement. The values from motion capture data are used as part of the environment-aware cost metric to evaluate characters' positions in the user-defined formation. Experimental results show that a crowd is able to incorporate these character-environment interactions in their motion whilst still being able to follow the user's input control. More details can be found in chapter 5.

1.2.3 Direct, Intuitive User Control of Crowd motion

The ease with which a user can interact with an application has a significant impact on their productivity and enjoyment. Current crowd control techniques use indirect methods for describing a crowd's behaviour through the application of constraints on existing motion or drawing of target formations. A control scheme for defining crowd motion should be simple to use and should enable a user to express the movement of the crowd in a straightforward manner so as to improve their efficiency at completing tasks using minimal input. There should be few restrictions implied by the control scheme on how a user can provide their input so that it can be performed in a way that is natural and intuitive.

This thesis presents results from a user study showing that the approach outlined in chapter 3 enables users to perform simple tasks involving crowd motion more efficiently when compared to using a traditional mouse-based control scheme. This study also shows positive responses from users for the use of the multitouch-based control scheme, including their ability to perform the desired task, how well they felt the crowd followed their commands, and how easy the control scheme was to use. More details on this study can be found in chapter 4. This thesis also describes an approach for generation of crowd motion based directly on the properties exhibited in a user's input control. This is different to the work in chapters 3 & 5 because it does not restrict the user to providing control via manipulation of a mesh, instead it lets the user provide their control signal in a way that is intuitive to them. The algorithm presented in chapter 6 employs a data-driven approach to convert a new user input into a corresponding crowd motion. In this method, a new input gesture is matched to similar gestures collected from users that describe the control for eliciting various crowd motions. This matching is achieved using a set of features that are invariant to common discrepancies between user's input gestures, such as the number of fingers used to perform the gesture. The correspondence between the set of collected gestures and the example crowd motion data provides an implicit mapping from the new gesture to its associated crowd motion based on its similarity to the gestures in the data set. Given this mapping, our method is able to produce a new crowd motion that corresponds to the user's input. In order for our method to combine various types of crowd motion data and to be able to produce motion for crowds of any size we create models based on principal component analysis of the crowd motion examples instead of using the raw data. More details can be found in chapter 6.

1.3 Thesis Outline

This thesis is structured as follows. First, we provide a review of work related to crowd simulation and control of crowd formation and movement in chapter 2. We describe how there is a lack of approaches for direct, interactive manipulation of a crowd's motion and shape that is important for realtime control in games and for more efficient creation of animated scenes. Since the algorithms that we propose use a multitouch device for user interaction, chapter 2 includes an overview of research in the area of touch input recognition that is relevant to the work in this thesis. We then explain our approach for simultaneous control of a crowd's formation and movement using a multitouch device in chapter 3. We show how this method enables a user to control a crowd, using a single gesture, to move in environments containing static and dynamic obstacles whilst maintaining a desired formation. The results of this work are published in Henry *et al.* (2012). In chapter 4 we assess the usability and appeal of this multitouch-based control scheme through a user study. In Chapter 5 we also describe an extension to the method in chapter 3 that adds direct interactions between characters and obstacles in the environment. This work is published in Henry *et al.* (2014). The proposed method in chapters 3 & 5 places restrictions on how a user can provide their control and on the kind of crowd motion that can be produced. In chapter 6 we explore an alternative data-driven approach to producing crowd motion using a multitouch device. This approach uses gesture examples and corresponding crowd motion data to generate movement of a crowd based directly on a user's input. Finally, we summarise the findings and contributions of the work in the thesis and provide suggestions for future directions in chapter 7.

1.4 Summary

Crowd simulation is important for conveying storylines in animation and providing interesting, interactive experiences in games. However, previous research methods require a number of steps to produce crowd motion and to control their formation and do not provide a way for a user to directly express the movement of the crowd. There is also a lack of methods for controlling crowds in environments that require character's to interact with obstacles. We have identified three areas for improvement in previous research. First, it is desirable for a user to control a crowd using a single step, enabling them to adapt the crowd's behaviour in realtime to suit the needs of interactive applications such as computer games. Second, a control scheme should be intuitive and expressive, allowing a user to quickly learn how to control a crowd and to directly influence the crowd's motion through their input. This is important in the user's immersion and enjoyment of the application. Finally, to allow a crowd to be controllable in complex scenes we need a method to handle characters' interactions with the environment whilst still following a user's input.

The approaches presented in this thesis are primarily designed for use in interactive applications, where a user is concerned with a high level of control over the movement of the crowd in order to achieve a certain objective. Often this movement can look relatively artificial when compared to real-life crowd scenes. While the system presented could be used for prototyping of basic animated scenes it does not provide the requisite realism for full animation production and as such it is not targeted at such an application.

Chapter 2

Related Work

In the past twenty years a large number of approaches have been proposed for simulating the movement and interaction of individuals, groups, and crowds of characters in virtual applications. This chapter discusses previous research in the areas most relevant to the work presented in this thesis, i.e. user control of the motion and formation of a group of virtual characters. The concept of multi-agent control is not exclusive to the fields of animation and computer graphics (for example, there exists extensive research on group coordination in robotics) however, this thesis focuses specifically on control of multiple virtual characters in interactive applications. As such, this review mainly discusses work in animation and computer graphics with some reference to work in other fields where appropriate.

In order to provide an appropriate background, Section 2.1 outlines work on crowd simulation as a whole, with a focus on how local and global approaches have been used to produce emergent crowd behaviours and realistic crowd motion. For an extensive review of crowd simulation, readers are referred to the work by Thalmann & Musse (2013) and Pelechano *et al.* (2008). Section 2.2 highlights work on user control over crowd simulations. Generally speaking, these approaches either provide tools for pre-specifying the motion of the crowd or for manipulation of existing simulations. Finally, a key component of the work in chapter 6 is a method for identifying gestures performed on a touch device therefore we present a brief review of related work in gesture recognition in Section 2.3.

2.1 Crowd Simulation

Broadly speaking, crowd simulation research can be categorised into either microscopic (local) or macroscopic (global) approaches. Microscopic approaches (Section 2.1.1) consider individuals within a crowd, concentrating on how they interact with other characters and planning behaviour locally. These can be considered as so-called agent-based approaches. On the other hand, macroscopic methods (Section 2.1.2) consider the crowd as a whole, using global information to govern the simulation and putting less emphasis on the individuality of the characters. This type of research is much more concerned with producing appropriate flow-like behaviour in the crowd. Despite this distinction, microscopic and macroscopic methods do not always appear in isolation. Hybrid methods often perform navigation for the crowd on a global scale and have the agent use a local controller in order to follow their assigned path (Section 2.1.3). A common approach in microscopic, macroscopic, and hybrid methods is to set a velocity for each agent in the scene in order to control their movement in the simulation. This velocity is produced as a result of various parameters that differ depending on the chosen approach.

2.1.1 Microscopic (Local) Approaches

As the more prevalent approach for crowd simulation, microscopic methods treat each character in the crowd as an individual entity, acting independently of others. Each member of the crowd has its own goals and characteristics, planning each of its movements using local information and acting accordingly. The idea is that realistic, crowd-like behaviour will emerge as a result of these local interactions between individuals. This approach was introduced in the seminal work by Reynolds (1987) where agents or “boids” followed simple local rules that defined their behaviour. These rules served for the agent to avoid collisions with other members of the crowd whilst remaining close to and aligning themselves with the rest of the flock. This work showed how a small set of rules can effectively produce the sort of flocking exhibited by birds and schools of fishes. This was later extended to incorporate additional steering behaviour in autonomous agents [Reynolds (1999)]. A variety of crowd scenarios can be produced with microscopic approaches by manipulating the types of rules that govern an individual’s behaviour and the importance of those rules. The level of control that a method can instil, and its predictability, is dependent on the number of parameters available and how obviously they interact with one another.

Reynolds' *rule-based* method has inspired several other similar approaches that are outlined in Section 2.1.1.1. Similarly, the influential work by Helbing et al. [Helbing & Molnar (1995); Helbing *et al.* (2005)] in civil and traffic engineering has led to *force-based* approaches for multi-agent simulation (Section 2.1.1.2). More recently, *data-driven* approaches have introduced techniques for controlling individuals in simulations based on example data from real crowds (Section 2.1.1.3). In contrast to these approaches, *geometric* methods (Section 2.1.1.4) focus on guaranteeing collision avoidance as opposed to reproducing specific crowd behaviours.

2.1.1.1 Rule-based Approaches

Rule-based methods have been used throughout animation and computer graphics to mimic complex, lifelike behaviours for individuals and can often result in realistic crowd behaviours. Sophisiticated models are capable of creating autonomous agents with various characteristics and specific goals [Musse & Thalmann (2001)]. Sakuma *et al.* (2005) introduced a two stage framework using personal space and virtual memory rules based on findings in the field of social psychology. This model improved the realism of the simulation at both local and global levels. Similarly, Pelechano *et al.* (2007) incorporated psychological and geometrical rules with physical forces (akin to *force-based* methods, see Section 2.1.1.2) to control individual agents in high density crowds. This framework modelled a number of crowd behaviours including lane formation and fast propagation of panic in crowd scenarios. Counterflows and overtaking behaviours can be elicited by incorporating parameters such as distance to and relative velocity of obstacles as well as surrounding crowd density. Binary parameters such as whether or not an individual is in a state of panic can be used to override or influence other parameters including an individual's maximum walking speed. In turn, controlling an individual's tendency to panic will affect how they will behave in a variety of ways.

Cognitive models have been used to simulate sophisticated agents that are able to learn from their environment [Funge *et al.* (1999)]. These models are extensible and have been successfully combined with behavioural, perceptual, and motor components [Shao & Terzopoulos (2007)]. In Shao & Terzopoulos (2007), a decision network is used to allow agents to decide on higher level behaviours, such as moving to a specific location within a virtual scene. Yu & Terzopoulos (2007) introduced an extension to this decision network framework that faithfully approximated real human behaviour by enabling the agents to handle uncertainties in their surroundings. Since this approach is modular, these networks can be easily augmented with nodes representing new aspects of a decision process. However, the high computational cost of the framework when handling complex decisions represented by large decision networks limits its use to small crowds. Finite state machines (FSM) have also been implemented to represent different behaviours for an agent in a crowd [Sung *et al.* (2004)]. The probability of transitioning to each state in the FSM depends on the current internal state of the agent and can also be affected by user-defined areas of the environment. FSMs can be easily grown by adding new states, allowing for characters in the simulation with varying complexity of behaviour. Often in such simulations the individual's movement can be controlled by specifying goal positions. The approach used will attempt to reach this desired position whilst accounting for other factors in the model.

"Composite agents" were introduced in Yeh *et al.* (2008) as a proxy-based method for simulating crowd behaviours including social priority, authority, aggression, etc. with little computational overhead. This approach associated with each agent various proxy agents that represented an individual behaviour for that agent in the simulation. For example, an aggressive proxy was placed ahead of the agent in the direction it was wishing to move, causing other agents to move aside to get out of the way. In addition to these approaches, rule-based methods have been developed to simulate sociological grouping [Musse & Thalmann (1997)], visual stimuli and motor response laws [Ondřej *et al.* (2010)], minimisation of energy expenditure through the principle of least effort [Guy *et al.* (2010)], and following behaviours [Lemercier *et al.* (2012)]. In the case of Ondřej *et al.* (2010) individual character behaviours can be adjusted by altering the safe distance the individual will keep from obstacles, their anticipation time for potential future collisions, and how strongly they will react to an impending collision. Variation in these parameters on a per-agent basis can lead to a variety of different crowd behaviours. Commercially, simple rules and fuzzy logic have been employed to great effect in the *Massive* software used for simulation of large crowd scenes in *The Lord of the Rings* movie trilogy.

A major issue with rule-based approaches is in the time-consuming design of the rules used to govern individuals. Furthermore, there is no guarantee of these rules consistently generating the desired behaviour at a crowd level. This is particularly the case when the environment is very dynamic and complex and often these approaches require a large amount of parameter tuning to achieve desired results.

2.1.1.2 Force-based Approaches

In their novel work, Helbing et al. developed the influential agent-based *social force* model [Helbing & Molnar (1995); Helbing *et al.* (2005)]. In this approach, the forces themselves are derived from the social tendencies of the agent to avoid collisions with other agents and the environment and to move in a specific direction at a desired speed. The social force model, and its derivatives, have since been used to simulate empirically observed emergent crowd behaviours including, but not limited to, arching and congestion at narrow exits, vortices arising from crossing pedestrian flows and lane formation from flows in opposing directions [Helbing *et al.* (2005, 2001); Hoogendoorn & Daamen (2005)]. These approaches model interactions between characters (and obstacles) as repulsive and tangential forces based on their relative positions. Force-based approaches also incorporate principles from social and psychological fields to generate these interaction forces. Helbing and colleagues have used the social force model to replicate observations from a series of controlled experiments [Moussaïd *et al.* (2009)]. Other extensions have been made to simulate pedestrians in normal and emergency panic situations [Teknomo (2006); Helbing *et al.* (2000)] as well as to simulate the formation of groups in a crowd using additional attractive forces [Braun *et al.* (2003)]. Closely associated to the force-based models are *particle-based* approaches. These approaches recreate human crowd phenomena by accounting for motion dynamics of characters [Brogan & Hodgins (1997)], and modelling collective crowd movement using a mass-spring damper system [Bouvier & Guilloteau (1996); Heigeas *et al.* (2003)].

Force-based approaches lack anticipation and prediction; the virtual characters only interact with other agents when they are sufficiently close. As a result, these methods fail to produce the correct microscopic behaviour. Additionally, *force-based* methods suffer from the same consistency and tuning issues as those found in *rule-based* techniques. This issue with designing and tuning laws is known as the steering problem. Various strategies have been proposed to solve this issue including use of synthetic vision [Ondřej *et al.* (2010)], principle of least effort [Guy *et al.* (2010)], and reproduction of experimental observations [Paris *et al.* (2007)]. These approaches represent the steering behaviour of a single character in a crowd using heuristics based on analysis of crowd and multi-agent motion examples. As discussed in Section 2.2.1, approaches exist for allowing an animator to adjust these behaviours via user-friendly interfaces. Alternatively, these parameters can be set for the crowd to display particular social grouping behaviours (see Section 2.2.2.1).

2.1.1.3 Data-driven Approaches

Data-driven approaches use example behaviours from video or motion capture data to govern the behaviour of individual characters in the crowd. In Lerner *et al.* (2007) a database of trajectories and associated spatio-temporal scenarios was created from video recordings of real crowds. At runtime each agent selects a trajectory from the database as a reaction to the current environment configuration. In a similar way, Lee *et al.* (2007) employ a regression-based learning algorithm to associate a person's perceived state with their resulting behaviour and Pettré *et al.* (2009) learn behaviours from experimental interactions data involving multiple characters in crossing scenarios. In this case it was seen that avoidance during crossing behavior is not purely reactive and involves an observation phase when one person registers the presence of people nearby. Ju *et al.* (2010) introduce a method for reconstructing crowds of arbitrary size and shape using spatio-temporal behaviours sampled from captured and simulated crowd data. Alternatively Lai *et al.* (2005) develop a group behaviour model in the form of a motion graph that captures the motion of the agent group as a whole, including the configuration of the group. This model is learnt using examples simulated using Reynolds' flocking model and not real crowd motion data. These *Group Motion Graphs* are only suitable when a group moves through an environment as a cohesive unit due to its high computational cost. Such data-driven approaches extend to use of individual character-character and character-environment interactions based on full-body motion capture data that can be concatenated to form larger crowd scenes. As with other data-driven approaches the types of motion produced by these approaches can be controlled by careful selection of the data provided to the system and the use of goal positions as constraints for the movement of characters. Furthermore, larger crowd scenes can be produced by animator control over the concatenation of these motion patches (see Section 2.2.1).

Compared to rule-based and force-based approaches that typically only reproduce a small set of predefined behaviours or predefined situations, data-driven approaches can reproduce complex and subtle behaviours in simulation that are present in real crowds. These approaches make no assumptions on the behaviour of the agents, however, the type of motion that is produced using these methods is limited by the scope of the input data which is often very expensive to capture. Furthermore, the processing and analysis of the data, and in a number of cases the synthesis of the crowd motion, must occur offline as it is too computationally expensive for real-time interactive applications.

The size, storage and access of the data is also an issue should the technique rely on referencing this data at runtime.

2.1.1.4 Geometric approaches

Instead of mimicking social or behavioural forces in crowds, geometric approaches consider the local area of an agent and select a future velocity that will produce collision free motion. This kind of approach is well used in robotics, generally working with incomplete or noisy information with regards to the robot's local surroundings. Fiorini & Shiller (1998) proposed the use of *Velocity Obstacles* (VO) for collision avoidance between multiple robots. A VO is defined in the velocity space of a given agent, representing the set of velocities that would cause a collision at some moment in time with another agent moving at a given velocity. The union of VOs from any number of obstacles constitutes the set of unsafe velocities for an agent. Choosing a velocity outside of this set will lead to collision free motion.

van den Berg *et al.* (2008b) extended the concept of VOs for use in both robotics and animation. Their introduction of *Reciprocal Velocity Obstacles* (RVO) assigned half of the effort of avoiding pairwise collisions to each agent. This meant that agents only move as much as is needed to prevent collision, ruling out large oscillatory behaviours. To select an appropriate velocity for the agents, the velocity space is randomly sampled and a heuristic search is performed to determine a collision and oscillation-free velocity that is as close as possible to the agent's desired velocity. The selection of an agent's velocity in this manner is quite inefficient. Guy *et al.* (2009) introduced faster RVO-like behaviour by guaranteeing collision free velocities on a discrete time interval and employing a discrete optimisation approach for selecting the final velocity. Similarly, fast, optimal selection of the agent velocity was achieved by solving a low-dimensional linear program in van den Berg *et al.* (2009). Further variations of the RVO framework have been introduced in robotics [Snape *et al.* (2009); Wilkie *et al.* (2009)] and animation [Kim *et al.* (2013)]. In Kim *et al.* (2013), the authors simulated physical interactions between characters and obstacles by introducing them as additional constraints in the *Optimal Reciprocal Collision Avoidance* (ORCA) framework from van den Berg *et al.* (2009). Geometric approaches can be used to control the movement of crowd in rigid formations based on desired goal positions relative to other members of the crowd (see Section 2.2.2.2).

Geometric approaches are able to produce locally smooth and efficient trajectories for agents in large crowds. However, as with other microscopic approaches, geometric approaches do not consider global factors in their solution. This can often lead to congestion when in complex environments or dense crowd scenarios. Additionally, the computation is performed on a per-agent basis and as such this can become prohibitive with very large crowds.

2.1.2 Macroscopic (Global) Approaches

Macroscopic approaches focus on the production of appropriate global motion of the crowd and are typically best employed as methods for simulating large, dense crowds. Unlike microscopic methods, these approaches place an emphasis on simulating emergent crowd phenomena, ignoring the individual characteristics of agents. The homogeneous approach of macroscopic methods makes them more appropriate for generation of large-scale crowd scenes, where the general visual flow of the crowd is more important than the behaviour of each character. Macroscopic simulations are generally controlled using crowd-level properties such as the vorticity and density of the crowd and are less concerned with characteristics of individual agents.

Macroscopic approaches have been used extensively in civil and traffic engineering to produce *regression* [Milazzo *et al.* (1998)], *queueing* [Løvås (1994)], and *route choice* models [Hoogendoorn & Bovy (2004)]. The approaches are used to assess the quality of pedestrian infrastructure designs in terms of comfort levels, walking times, safety and the management of pedestrian flows. Despite this wide variety of features, these models are unable to produce self-organisation phenomena seen in crowds.

The concept of fluid dynamics for representing the flow of crowds in simulation was introduced by Henderson (1974), who suggested that the motion of the crowd should be governed by a set of partial differential equations that specify the motion of gases and fluids. This idea was confirmed empirically by Helbing *et al.* (2005, 2001) who showed that at low density, crowds moved in a way similar to molecules in a gas. At increasing densities, crowds exhibited motion more and more similar to fluids, eventually mimicking granular flows at very high density. Helbing (1992) created a gas-kinetic model for crowds as an extension to the work by Henderson (1974). This model accounted for avoidance and deceleration behaviours that occur as a result of pedestrian interactions, however, the numerical solution to the gas-kinetic equations was difficult to obtain. To overcome this, Hoogendoorn & Bovy (2000) devised a discrete gas-kinetic model for pedestrian flows that used a novel, particle representation for the agents in the crowd.

In the work by Hughes (2002, 2003), a human crowd was interpreted as a flow of "thinking fluids"; a continuous density field whose behaviour is affected by its situation. This *continuum dynamics* approach uses a pair of non-linear, time-dependent, partial differential equations to produce a goal-dependent field that guides the pedestrians towards their target whilst avoiding collisions. Using video data, Hongwan *et al.* (2003) confirmed the relationship between pedestrian velocity and pedestrian acceleration described in Hughes (2002, 2003) and later, this approach was successfully used for medieval battle analysis [Clements & Hughes (2004)]. Hughes' work was adapted by Treuille *et al.* (2006) to recreate various macroscopic behaviours of crowds for computer graphics and animation. Treuille *et al.* (2006) used a particle representation and an eikonal (non-linear differential) equation that described the motion dynamics of the crowd. By solving the eikonal equation on a grid representing the environment, the method dynamically constructed a density-dependent velocity field that was smooth and goal-directed. This velocity field defined a mapping between a point in the environment and a direction vector that an agent should follow when they find themselves at that point. The overall quality of the simulation is dependent on the resolution of the grid used to approximate the equation. Since the solution of the eikonal equation is computationally expensive, real-time simulation using the method in Treuille *et al.* (2006) requires an appropriate trade-off between the resolution of the grid and the quality of the simulation. This issue was somewhat alleviated in Jiang *et al.* (2010) who combined the continuum representation with an adaptive grid size method based on the complexity of the environment. In addition to the grid resolution issue, the expense of computation in Treuille *et al.* (2006) required that large crowds had to consist of a small number of homogeneous groups with common goals and behaviours, limiting the variation in the simulation. Narain *et al.* (2009) achieved near-interactive simulation of very large, dense crowds using a hybrid continuum-based method. Their approach considered the crowd as a "unilaterally incompressible" fluid; a fluid that is neither purely compressible nor purely incompressible, and imposed a volume constraint limiting the maximum volume of the crowd. In the method, each agent's desired direction of movement contributes to the "flow" through a given grid square. The resulting averaged flow is then combined with neighbouring flows and adjusted according to the unilateral incompressibility of fluids, producing a flow field that handles inter-agent collisions on a macroscopic level.

Velocity field methods are an intuitive way to guide a set of agents in a crowd simulation. In general, the vector field is either computed automatically or it can be manually defined to produce certain behaviour. In Courty & Corpetti (2007) the authors extract a time series of velocity fields from videos of real crowds. This set of velocity fields is then used to advect agents in the scene to produce new crowd motion. Similarly, Musse *et al.* (2007) combined Helbing's social force model with vector fields obtained from video sequences to synthesise movement of virtual agents. Chenney (2004) designed divergence-free velocity fields that remove the need for explicit collision avoidance but, due to the static nature of the grid tiles, the fields are unable to handle any interaction between agents. Design of velocity fields for crowd simulation has also been employed in Jin *et al.* (2008) and Patil *et al.* (2011), the latter allowing creation of several macroscopic crowd behaviours and overcoming the issue of Jin *et al.* (2008) by guaranteeing smooth, goal-directed *Navigation fields*, free from local minima.

Macroscopic approaches, especially methods based on continuum dynamics, are particularly effective at simulating thousands of agents in real-time because the movement of the crowd can be modelled at a coarse level using flow velocity and agent density. Large crowd scenes are typically viewed at a distance and so, in this case, the individual behaviour of the agents is less important than the overall motion of the crowd. The control of macroscopic approaches focuses on the manipulation of the velocity field that governs the movement of the agents. This can either be through directly altering the velocity field or adjusting the parameters used to generate the continuous potential field on which the velocities are based (see Section 2.2.1).

2.1.3 Hybrid Approaches

In contrast to the single framework approaches to crowd simulation discussed in Section 2.1.2 [Hughes (2003); Treuille *et al.* (2006)], *hybrid* approaches combine global path planning with local methods (Section 2.1.1) to navigate agents in crowd simulations. These two-level approaches use a roadmap or a graph of the environment to govern the global motion of each agent in the simulation, allowing a local planner to handle collision avoidance with other nearby agents and obstacles. For example, Bayazit *et al.* (2003) combined Reynolds' flocking model with probabilistic roadmaps to produce motion of cohesive groups through a virtual environment. van den Berg *et al.* (2008a) showed how the RVO framework can be used with pre-computed roadmaps to produce effective multi-agent navigation in crowded environments.

A number of hybrid approaches have focused on appropriate decomposition of the environment in order to improve the global motion of agents. Sud *et al.* (2008) computed *Multi-Agent Navigation Graphs* using first and second order Voronoi diagrams of all obstacles and agents in the environment. Though this provided maximal clearance between agents, its computational cost limits its use to small crowds of only a few hundred characters. Other approaches handle global navigation by assessing the open space of the environment. Lamarche & Donikian (2004) represented walkable areas with a set of convex cells produced using a constrained Delaunay triangulation of the environment while Pettré *et al.* (2005) decomposed the navigable space into a set of interconnected cylinders centred on the Voronoi diagram. Variations on the paths can be produced by iteratively adjusting the edges of this *Navigation Graph* [Pettré *et al.* (2006)]. Geraerts & Overmars (2007) created a *Corridor Map* using the medial axis of the virtual environment and information on the minimum clearance from an obstacle. A path is generated using the medial axis to direct the global motion of the character and a potential force-field approach is used to control the local motion of the character to remain within the collision-free corridor. The use of the force-field produces smooth paths whilst allowing the character to locally deviate from it's path if it encounters dynamic obstacles or other characters.

As an alternative to the two-layer approach in the previous methods, Golas *et al.* (2013) proposed a hybrid method for long range-collision avoidance that dynamically blended between solutions from local, discrete planners and continuum-based planners depending on the density of the crowd. The key idea behind their algorithm was to extrapolate the crowd motion a set number of discrete timesteps into the future and perform collision avoidance with increasing uncertainty. This provided greater anticipation of collisions at a relatively small additional computational overhead. Despite this, the method does not consider global path planning for the agents and in some cases the motion is quite unnatural.

Hybrid methods are able to produce global movement of agents in a crowd whilst still simulating heterogeneous characteristics through use of local methods. Unfortunately, the separation of global planning from local planning in some hybrid methods means that they are still susceptible to local minima when determining paths. This can lead to congestion situations in highly dynamic and dense environments that can only be resolved by unnatural motion of the characters. To overcome this, many methods have been introduced both in robotics and animation to prevent congestion and collisions by performing global planning for multiple agents rather than on a per-agent basis. In general, decoupled planners plan for each agent individually and then try to coordinate the resulting motion [Peng & Akella (2005); Simeon *et al.* (2002)], whilst centralised planners compute the motion of all the agents simultaneously in a composite configuration space [Li & Chou (2003); Schwartz & Sharir (1983)]. Due to this combined planning space, the computational cost of such planners grows exponentially with the number of agents. In the case of congestion, a common method is to simply have the agents replan their trajectories using information from previous planning episodes [Koenig & Likhachev (2002)]. Unfortunately, this is often insufficient for producing a feasible trajectory and can be quite impractical when dealing with large numbers of agents. Another approach is to plan using a space-time graph of the environment and agents' movement. This allows the planner to coordinate the arrival and departure of agents at nodes in the graph using local [Singh *et al.* (2011)] or global [Karamouzas *et al.* (2013)] information. Alternatively, a priority can be determined and each agent then plans their route in turn [Lau & Kuffner (2006); Sung *et al.* (2005); van den Berg & Overmars (2005)]. Each successive agent treats the previous agents' plans as dynamic obstacles. The growing complexity of the planning space limits the number of agents that can be simulated with this method. Finally, the motion of the agents can be considered as a dynamic flow problem [van den Akker *et al.* (2010); Karamouzas *et al.* (2013)]. In this case, a capacity is associated with each arc of a graph (and node in the case of Karamouzas *et al.* (2013)) representing the free space in an environment. This capacity indicates the maximum number of agents that can exist on that part of the graph per unit time. Given this information, the planner finds paths that minimise the average travel time for all agents to reach their goal by iteratively updating paths only when they reduce the cost for the agents.

2.1.4 Summary

There exist many techniques for producing multi-agent motion both in robotics and computer animation. These can generally be classified into microscopic (local), macroscopic (global), and hybrid approaches. Local approaches are capable of producing heterogeneous agent characteristics using simple rules based on collision avoidance, behavioural factors, and example-based local decision making. However, these methods require a lot of tuning to achieve desirable collective behaviour of the agents and there is no guarantee that the correct global behaviours will occur in simulation. Furthermore, because computation is on a per-agent basis, the cost of such approaches grows with the size of the crowd. Global methods provide a way to elicit macroscopic behaviour in crowds consisting of thousands of agents but they produce homogeneous crowds that lack individualism in the characteristics of the agents. Hybrid methods provide a good middle ground between these two approaches, combining global navigation with local collision avoidance. Methods have been introduced in the literature that account for multi-agent planning issues that arise from basic hybrid techniques such as congestion and local minima.

2.2 Crowd Motion Control

The techniques listed in Section 2.1 are capable of producing realistic multi-agent motion for use in animation and computer games. However, to do so, these methods often require a lot of parameter tuning to achieve desired motion and they do not provide precise control over the behaviour of an individual agent. This can mean that to achieve an appropriate animation the simulation must be run several times with varying parameters, requiring excessive time and effort. To alleviate this issue, interactive editing approaches have been proposed that enable an animator to have direct control over animated crowd behaviours. Control of multi-agent motion is an extensive topic and there have been many approaches covered in the literature. In computer graphics and animation, methods tend to cover two main areas: 1) agent pathing and crowd flow control (Section 2.2.1), and 2) group formation and interaction control (Section 2.2.2). In each of these areas, the user is able to specify constraints on the behaviour of the crowd using example data, sketch-based interfaces, and manipulation of velocity fields amongst other techniques. These constraints can either be applied before or during the simulation in order to guide the agents' movement or they are used to edit pre-existing crowd motion.

2.2.1 Agent Pathing and Crowd Flow Control

For general crowd scenes, it is important for an animator to be able to define the goal-directed movement of agents. For example, an animator may wish to define the entry and exit point and locations that agents visit in the scene, and to describe the "flow" of the crowd (the global behaviour elicited by the collective motion of the agents). One of the first techniques for authoring crowd scenes was introduced by Ulicny *et al.* (2004). Their *Crowdbrush* system allowed animators to draw characters in a scene using a brush-style interface. The animator could not only create characters within the scene but also control the characters' paths and activation of simple behaviours using variations on the brush tool. Although the interface enabled production of large scenes in only a few minutes, it is still necessary for a user to adjust the brush parameters to specify details such as the portion of the crowd that they were manipulating or the operator (creation, behaviour, colouring etc.) that would be applied to the selected area of the crowd scene.

Since the work by Ulicny *et al.* (2004), methods for controlling crowds have focused on manipulating characters' trajectories by assuming that an agent's movement is governed by a local, agent-based steering strategy with a global velocity (flow or navigation) field. The general concept has been to manipulate the velocity fields in order to create new motion or alter existing crowd movement. As examples, alteration of the crowd motion can be achieved by specifying constraints on the velocity field using example data or using sketch-based interfaces. Chenney (2004) enabled a user to design divergence-free velocity fields, however, these cannot be controlled interactively and are unable to provide goal-directed navigation of the agents. As a way of letting a user define the velocity field directly, two sketch-based approaches have been defined. The first, Jin *et al.* (2008), allows a user to sketch the desired velocities at certain anchor points in the environment. These velocities are then interpolated using a radial basis function approach to compute a continuous vector field for controlling the agents. There is, however, no guarantee that navigation using these fields will not end up in local minima. In contrast to this, Patil *et al.* (2011) guarantee singularity-free guidance fields generated from user-drawn paths or flow fields extracted from real crowd footage. Their method produces smooth, goal-directed fields and allows a user to define a wide range of macroscopic behaviours and resolve issues of congestion using their authoring interface. Alternatively, Park (2010), inspired by the work of Goldenstein *et al.* (2001) who used harmonic functions to prevent agents from ending up in local minima, used keyframed *control particles* and harmonic functions to define the flow of a crowd in environments containing obstacles. The adjoint method has been popular for computation of simulation forces used for keyframe control of fluids [McNamara *et al.* (2004)], and later, particle systems including flocking [Wojtan *et al.* (2006)], and directing crowd models [Allain *et al.* (2014)]. In Allain *et al.* (2014) the adjoint procedure is adapted depending on the underlying method for crowd motion control, and the type of user input control being provided (i.e. whether the control is given at the individual agent level or with higher level representations such as density, vorticity, and velocity). Their method requires that the underlying crowd model must be differentiable by the state that it controls. The adjoint method provides a speed gain for optimising control parameters by solving the much simpler dual formulation of a linear system. It does however still use complex optimisation requiring carefully-designed objective functions.

The use of gradient-based methods can provide smooth trajectories for agents, however, due to the discrete nature of agents, some issues can arise with handling of collisions between individuals. Furthermore, the methods presented only adjust the velocity field itself and do not allow direct specification of an individual's goals. In many cases, manipulation of the flow field is specific to one area of the environment and not to the crowd. This is effective in cases where an animator wishes to define how a crowd moves around a scene but not for heterogeneous control of agents or groups of agents in the crowd. Wolinski *et al.* (2014) present an alternative sketch-based approach to allow a user to specify a rough idea of the motion of the crowd based on a given metric such as distance between agents. Their framework automatically computes the best underlying crowd simulation algorithm (from boids, social forces, and RVO-based approaches) along with an appropriate set of parameters in order to achieve the user specified motion.

As an alternative to velocity field control, a number of methods exist for creating crowd scenes by populating the environment with existing motion data through the use of *patches*. These patches are generally 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. Lee *et al.* (2006) proposed motion patches that represent interactions in a small rectangular area between a character and objects in an office or playground. These can be combined during runtime to create a large scene with characters interacting with many different objects. A similar approach is used by Yersin *et al.* (2009) however, their *Crowd Patches* are created by precomputing the behaviours shown by multiple walking characters when they are avoiding each other. While this approach relied on generating trajectories inside a patch, given a starting time and entry/exit point for an individual, Li *et al.* (2012) concentrated on extracting patches of data showing periodicity and the issue of connecting these patches to others showing a similar pattern of motion. Jordao *et al.* (2014) extended the work of Yersin *et al.* (2009) and introduced manipulation of the patches through local deformation. This allowed the authors to provide space-time editing of crowd motion patches in the form of stretching, bending, cutting and merging gestures. Instead of altering crowd simulation parameters, their method allows an animator to directly manipulate the coverage and shape of a crowd motion which are more direct ways of specifying the final simulation. Further patch-based methods have been used to simulate dense interactions between characters such as fighting, cooperative carrying of objects and coordinated collective motions [Shum *et al.* (2008); Hyun *et al.* (2013)]. All of these methods require appropriate constraint fulfillment in order to successfully combine multiple motion patches. Rigid, uniform patches can be concatenated easily but at a cost in the variety of scenes they can produce. More flexible patches can create more interesting scenes, however, as a result the correct tiling of these patches becomes more computationally costly. In either case, the exact trajectories displayed by the characters is limited by the patch's motion data.

2.2.2 Group Formation and Interaction Control

Sometimes in computer animation or interactive applications a user would like to control a group of characters to configure themselves in a certain shape or to obey a set of formation constraints whilst in motion. There can be several reasons for this that can largely be summarised with the following: the user wishes the group of agents to perform a certain task and, by adopting a particular group configuration or obeying certain constraints on their formation, the group can complete this task more efficiently. Such tasks can include searching environments and collecting objects, where certain formations provide better coverage for speeding up the execution of the task and reducing the chance of failure. In battle scenarios or in team sports, certain formations can give a group of characters a competitive edge over their opponent making all the difference in the result. Equally, coordination of multi-agent motion can help in cooperative tasks such as moving of obstacles or passing efficiently through areas of an environment. Finally, by adopting set configurations, groups of characters can be used for stylisation in large displays such as marching band performances, or for storytelling, for example, when a school of fish is used to mimic the Sydney opera house in the "Finding Nemo" movie by Disney Pixar. Group formation control research is well studied in both the robotics and computer animation fields and can be broadly classified into *behaviour-based* (Section 2.2.2.1) and *geometric* approaches (Section 2.2.2.2).

2.2.2.1 Behaviour-based Formation Control

A number of *behaviour-based* techniques derived from the microscopic approaches of Reynolds (1987) and Helbing (1992) (Section 2.1.1) have been utilised for grouping in multi-agent systems both in robotics [Mataric (1992, 1993)], and animation [Musse &

Thalmann (1997); Brogan & Hodgins (1997); Braun *et al.* (2003)]. In these methods, simple motion primitives are designed and combined to generate complex patterns as a result of the interaction of several agents. The agents are able to form collective groups by acting independently and without need for explicit communication. Often these methods are used to perform cooperative tasks [Arkin (1992)], or to replicate the sorts of grouping and formations observed from real data. Qiu & Hu (2010) employed a behavioural approach similar to Reynolds' flocking model to simulate different group structures in pedestrian crowds. Their method used intra-group and inter-group matrices that defined the influence agents (and groups) had over one another's aggregation and following behaviours. Further methods have been proposed that can handle the navigation of groups of agents including Bayazit *et al.* (2003) who combined flocking methods with probabilistic roadmaps, Kamphuis & Overmars (2004) who performed path planning in "corridors" of the environment to maintain group cohesion, and Li & Chou (2003) who used a centralised planner to compute the simultaneous motion of multiple agents. However, with multi-agent motion it is frequently desirable that the group forms a particular shape or spatial configuration either during or at the end of their motion, and none of these methods provide such control.

Often the requirement in crowd formation control is for agents to move in a similar direction to other agents while maintaining an overall formation. The work by Ju *et al.* (2010) is capable of forming arbitrary group formations, however, the method is too computationally expensive for interactive applications and is unable to handle challenging path planning problems. A special case of behaviour-based formation control, *leader-follower* approaches have been used to simplify group motion planning to a single leader agent and have the rest of the group follow their reference motion. This technique has been successfully used for crowd simulation [Loscos *et al.* (2003); Qingge & Can (2007)] and for maintaining group formations in robotics [Wang (1989); Desai *et al.* (2001); Poonawala *et al.* (2013)] where the followers attempt to maintain a particular state relative to the leader. This state is often defined as a vector offset from the leader's position [Wang (1989)] but it can, for example, be defined as a combination of distance and angle from the reference motion [Desai *et al.* (2001)].

A reference motion does not always have to be defined in terms of the group's leader. In Balch & Arkin (1998), groups of two to four robots were shown to arrange themselves into diamond, line, column, and wedge formations using a schema-based approach. The robots achieved their formation by moving towards *attachment sites* that were positioned relative to a reference point in the formation (either the average position of the group, the position of a leader robot, or the position of a neighbouring robot). This approach was extended to handle larger groups with arbitrary geometric formations [Balch & Hybinette (2000)]. The authors also prevented crossing of the robots when changing formations by having them choose their final position in the formation as the nearest point in a list of candidate attachment sites.

Many behavioural approaches look to replicate the multi-agent formations seen in nature [Klotsman & Tal (2012)], and in real crowds, where it has been observed that a large portion of the crowd forms small groups of up to four pedestrians that display distinct formations based on their social interaction [Peters & Ennis (2009); Moussaïd *et al.* (2009)]. These configurations include “Line-abreast” (individuals walking side by side), “V-like” (two individuals walking slightly ahead of another), and “River-like” (individuals walking one behind the other). In Peters & Ennis (2009), the authors were able to create visually plausible crowd scenes by creating formation templates based on these observations. Each group of agents was assigned a state machine containing various templated formations that they would switch between based on their surrounding environment. Alternatively, Moussaïd *et al.* (2010) added an additional *interaction term* to Helbing’s social force model to account for social interactions within a group and replicate the small group formations. The interaction term included a force for maintaining gaze direction towards the rest of the group as well as attraction and repulsion forces for agents to keep a certain distance from the group without overlapping one another. Karamouzas & Overmars (2010) unified these observed configurations into a single objective function requiring less tuning to achieve a good simulation. This function minimises deviation from the desired velocity of the group, collisions, and the cost of selecting a candidate formation given that the “Line-abreast” formation is the most desirable, and “River-like” is the least.

Behaviour-based methods provide a way to define multi-agent motion both as an arbitrary group and as a rigid geometric formation. Furthermore, these techniques have been shown to replicate the behaviour of groups from real crowd observations. On the other hand, these approaches can require a lot of tuning to achieve good results and they do not allow for easy manipulation of the formations during the course of the group’s motion. Should an animator wish to adjust the group’s configuration, these methods do not consider the smoothness of any formation transition.

2.2.2.2 Geometric Formation Control

In geometric formation control the focus is on the shape and interconnectivity of the crowd. As such, approaches frequently employ geometric constraints to define the configuration of the group. In contrast to the leader-follower methods described in Section 2.2.2.1, that define follower positions as offsets from a leader, geometric approaches impose a structure on the group as a whole and each agent's target position is defined relative to this. This makes these approaches well suited for user control of the group configuration as they allow a user to manipulate the group's structure directly. The agent's can then update their target positions within the altered formation. Before continuing with the discussion of these geometric approaches, it is worth mentioning another, very similar formation control approach based on *generalised coordinates*. In Spry & Hedrick (2004), the authors use generalised coordinates to represent the location, orientation, and shape of a formation allowing the specification of piecewise trajectories for each of these terms. These trajectories define the movement of the formation as a single unit and allow for shape-safe motion by considering constraints on the parameterisation of the formation shape.

A common approach for geometric formation control, particularly in robotics, is the use of a *virtual structure*. Introduced by Lewis & Tan (1997), a virtual structure constitutes a rigid body defined by a system of point masses that are stationary with respect to a frame of reference. The structure describes the enforcement of geometric relationships not by a physical system of constraints, but by a human-made control system (hence virtual). Typically, the movement of a formation is achieved by first aligning the virtual structure with the current positions of the agents, moving the virtual structure using a virtual force field, calculating the trajectories and moving the agents to the desired points on the virtual structure and then repeating. This approach is generally used in robotics to control formations of Automated Guided Vehicles (AGVs), Unmanned Air Vehicles (UAVs), and Autonomous Underwater Vehicles (AUVs). Li & Liu (2008) used a virtual structure to define the relative positioning of UAVs flying in formation. Mehrjerdi *et al.* (2011) combined the Lyapunov technique, for trajectory following, with graph theory, for the coordination of multiple robots, and used these inside a virtual structure to achieve group motion whilst maintaining formation. To follow curved trajectories, Low & San Ng (2011) extended the flexibility for virtual structures, allowing formations to be able to turn continuously and smoothly whilst moving along a path.

Geometric approaches also constitute *shape-constrained* methods, largely prevalent in computer games and animation. Approaches have been proposed for constraining the motion of flocking agents according to a user-defined shape [Anderson *et al.* (2003); Xu *et al.* (2008)]. The method in Anderson *et al.* (2003) requires a lengthy sampling process but allows specification of position and timing constraints for individuals, group centroids, and group outlines. Xu *et al.* (2008) used spherical projection to establish correspondences between flock members and points sampled in the formation shape to guide agents, however, their method requires the use of fuzzy logic to fine tune the flocking parameters. Chang & Li (2007) consider formation path planning in terms of a shape template whose centre must move through the free space in an environment but its boundary can still overlap with obstacles. This template represents the desired group formation. The agents use fuzzy logic to move within a set of grid cells that best fit the current position of the template whilst only occupying open space.

By defining an agent's position in a formation as a vector offset and relative angle from a predefined path, Ho *et al.* (2010) are able to perform global navigation of a flock that adapts its formation acccording to the curvature of the path. Their approach also handles obstacle avoidance by disengaging formation following and later regrouping a crowd so that it can pass around objects in the environment and through narrow spaces. This method was presented as part of the Flocking Animation Modeling Environment (FAME) system [Ho *et al.* (2012)] that also used uniform sampling and a one-to-one mapping process to move agents between discrete formation shapes. As a full resampling of the target shape would be computationally costly with smaller, continuous changes in formation the FAME system used barycentric coordinates to define agents' positions relative to adjustable control points. The agents' target positions were then updated using these coordinates when a user altered the position of one of the control points. To allow smoother transitions between formations the system also enabled swapping of target positions in the formation based on a pairwise minimisation of the agents' distance to their goals.

A sketch-based interface for defining target crowd formations is presented in Gu & Deng (2011b). The authors prioritised sampling of agents' positions in the formation at the boundary and used a flood-fill algorithm to fill the rest of the user-defined shape. To maintain adjacency relationships, *formation coordinates*, a local coordinate system similar to polar coordinates, were proposed as a way to define an agent's equivalent position in the source and target formations. The transitions between formation shapes however were not optimal i.e. agents did not always fill in the target formation smoothly. This was handled by allowing a user to define the preferred direction of agents during formation transitions using extra strokes [Gu & Deng (2013)].

Alonso-Mora *et al.* (2011) presented a method for generating smooth and oscillation-free trajectories for formation tracking in open environments with a multi-robot system. A Centroidal Voronoi Tesselation (CVT) was used to compute the final distribution of the robots in a target shape. Each robot was assigned a goal using the Hungarian algorithm to minimise the movement for all of the agents and their path planning was handled by the ORCA framework. Zheng *et al.* (2014) used shape morphing to interpolate a series of keyframed formations. Like Alonso-Mora *et al.* (2011), the authors employed the CVT and Hungarian algorithm for optimal positioning and assignment of agent positions in the target formation however, their focus was on producing smooth transformations between group formations. Agent paths followed the centroids of the Voronoi cells generated for each intermediate shape, guaranteeing smooth, well distributed, and collision free trajectories that maintained correspondence during the group movement. With a similar focus on formation transformation, Xu *et al.* (2012) also used the Hungarian algorithm to compute correspondences between Delaunay triangulations of source and target configurations. In this work however, the cost metric considered the "disorder" of agents' speed and local structure variations to achieve a smooth transition. A later extension maintained local relationships in the crowd by considering subgroup structure and employing mutual-information feedback at run-time [Xu *et al.* (2014)].

The concept of spatial relationships and group structure is well used in formation control and group motion editing. Takahashi *et al.* (2009) presented a method for interpolating between spectral-based structures of keyframed formations. Spectral decomposition was performed on the Laplacian matrix of the group's adjacency graph (based on a Delaunay triangluation of the agent positions). This yielded a set of eigenvectors and eigenvalues that described the structure of the group's formation and allowed reconstruction of intermediate group configurations. An as-rigid-as-possible mesh editing scheme [Igarashi *et al.* (2005)] has been applied for deforming and concatenating existing crowd formations to synthesise larger scale animations [Kwon *et al.* (2008)] and to apply constraints on the interaction and movement of crowds of characters [Kim *et al.* (2009)]. More recently, a similar, cage-based deformation has been used to interactively alter spatio-temporal properties of large-scale crowd motion [Kim *et al.* (2014)].

2.2.3 Summary

Due to the high degrees of freedom and sometimes unpredictable nature of multi-agent simulations, methods have been created to offer animator control of such motion to achieve a desired outcome. These approaches broadly cover control of the overall motion of a crowd (Section 2.2.1) or more explicit control over group formation and interactions (Section 2.2.2). Control schemes allow definition of crowd flows by applying constraints to an ambient velocity field that guides the motion of the crowd. A user can alter these fields using various interfaces, however, these approaches do not allow discrete control over an individuals behaviour. Furthermore, such velocity field manipulation only alters the crowd motion at certain areas in the scene, rather than on an agent or group basis. Group-level motion can be produced through use of behavioural laws specifying social interactions or by maintaining relative positioning to a reference point in a formation. As well as this, user's are able to specify group configurations based on spatial relationships and shape templates. These formations can be pre-specified or created/adjusted at runtime using sketch-based interfaces, keyframes, or mesh manipulation. Often though, these techniques require a large amount of user input to specify these constraints and/or they do not allow a user to alter the shape or pathing of the group whilst it is in motion.

2.3 Touch-based Gesture Recognition

A number of tools have been proposed for allowing a developer to create control gestures for touch devices easily, including generating gestures based on demonstration [Rubine (1991); Lü & Li (2013)] or use of regular expressions [Kin *et al.* (2012)]. However, these techniques do not account for the variability in control style shown by users in [Micire *et al.* (2009)], when performing the same task on a touch device, including differing numbers of fingers or whether they use one or two hands. Instead these properties either have to be predefined by the author of the gestures, or gestures are limited in the number of simultaneous inputs that can be handled [Rubine (1991)]. The $-family of gesture recognisers constitute a set of simple techniques that show good performance on gesture recognition for single stroke (unistroke) gestures using the $1 approach [Wobbrock *et al.* (2007)], and gestures involving multiple sequential strokes (multistroke) [Anthony & Wobbrock (2010)]. Although used for multistroke recognition, the latter, $N, method involves forming a number of candidate unistrokes by concatenating the individual elements of a multistroke in all possible orders and directions before using the $1 recogniser to find the closest match. Each of these methods is designed for recognition of gestures involving inputs occuring one at a time but it has been shown that users often prefer to provide gestures using many fingers at once [Rekik *et al.* (2013)].

Work has also been done to perform recognition on multi-touch gestures involving multiple, simultaneous strokes. The $P method performs matching of point clouds generated from the individual touch points in a user's input [Vatavu *et al.* (2012)]. While effective, this approach does not maintain any concept of a stroke i.e. ordering of the touch inputs. Consequently any information on the evolution of the gesture over time is lost. Jiang *et al.* (2012) showed effective recognition with the $1 recogniser on a key stroke extracted from multi-touch gestures. However, by performing the key stroke extraction the expressibility of multi-touch input is ultimately compromised. Rekik *et al.* (2014) proposed a method for clustering user inputs into multiple strokes instead of a single key stroke. The authors showed how recognition of multi-touch gestures is markedly improved by performing this *Match-up* step before $P recognition on the resulting strokes. Although this method maintains the multiple simultaneous inputs in the candidate gesture, the recognition step ignores the ordering of the touch input signals as a result of using the $P recogniser.

Chapter 3

Multitouch Formation Control

The major difficulty of crowd control lies in its high degree of freedom. Each character in the crowd is an entity and should be able to move independently under different circumstances. One problem that arises when directly applying previous methods for real-time crowd control is the difficulty in specifying low level details when the crowd interacts with the environment. For example, when a crowd walks through an environment that diverts into multiple smaller pathways, they should split into smaller groups and pass through each of the available routes. For a user to successfully specify this behaviour a lot of input is required and is generally achieved using a multi-pass algorithm. This typically involves the user stopping the animation and drawing multiple strokes offline to specify the individual paths for different groups of characters. This can be tedious to specify the movement of a large number of characters, as well as requiring a large amount of animator time. A possible solution is to define a vector field and move each subgroup along its gradient [Kato *et al.* (2009)]. However, there can be cases that the flow is opposite to the direction that the characters are supposed to move. We prefer to use a more interactive process allowing the users to easily intervene and adjust the trajectories on-the-fly. We observe that most of the movement in such a situation is affected by passive interactions between the crowd and the environment. In the above situation, the crowd will split so that each person walks into the street that is closest to him/her, while avoiding crossing the path of the other agents. We believe that these kind of passive interactions can be computed automatically without a significant loss in simulation quality.

When moving through an environment it is often desirable for the crowd to maintain a specific formation to pass through the environment easily or to achieve a certain task. To achieve this, previous approaches require a user to define the agents' paths using a significant amount of input [Kwon *et al.* (2008)]. The problem with previous methods of formation control is that the characters are strongly bound to a specific location in the formation, [Kwon *et al.* (2008); Gu & Deng (2011b, 2013)]; once their locations are defined in the formation, the characters are required to maintain their respective positions even if in doing so they prevent other characters from reaching their assigned location. This can happen when characters placed on the border of the formation prevent other characters from reaching the centre because they are very close together. In real situations, people in the border will simply shift toward the centre of the formation to produce space for the people on the outside. This behaviour allows the crowd as a whole to achieve the target formation.

In this chapter, we propose a new method to reduce the dimensionality of crowd control by making use of the passive dynamics between the individual agents, as well as those between the agents and the environment. In order to achieve this goal, we propose a new mesh-based crowd control scheme that makes use of a multitouch device's simultaneous input capability. We enable a user to specify the movement and shape of a crowd at the same time by using the touch device to manipulate the configuration of a two-dimensional mesh representing the crowd formation. In the current work, the update of the deformable mesh to avoid environmental obstacles and to track the user's formation control is combined into a single control signal. As a result, the mesh deforms automatically based on the influence of the environment, while keeping the overall formation specified by the user. Our method is a single-pass approach in which the user can manipulate the crowd and see the updates to its formation in real-time. We allow characters to switch target positions between frames by performing a *Mass Transport Solver* (MTS) similar to that widely used to compute the Earth Mover's Distance [Rubner *et al.* (1998)]. This essentially minimizes the overall movement of all characters and reduces the chance of potential blocking among them during formation transitions, hence producing more realistic animations.

Experimental results show that our system can produce realistic scenes of a crowd controlled through minimal, intuitive and high-level input signals from the user. We create scenes in which the crowd has to pass through complex environments such as a town with several diverging/converging routes, a street where multiple cars are driving, and constrained environments such as narrow pathways. Our system is best applied to real-time crowd control applications such as strategic games. It can also be used efficiently to create scenes such as city-scale crowd flow for computer animations.

3.1 Contributions

- A new real-time scheme to manipulate a crowd's formation and movement in constrained environments using a multitouch device. This scheme makes use of the passive dynamics of the interactions between the user-defined crowd shape and the environment allowing the user to focus on the design of high level movements, while leaving the fine details to the system.

- A method for handling perturbations to character movement caused by the environment that allows characters to track the user's control well. This is achieved by combining the deformable mesh representation of a crowd with a mass transport solver. The characters are not constrained to specific locations inside the deformable mesh, but cooperate together to occupy locations on the mesh based on the solution to the mass transport problem.

- A set of simple multi-touch gestures for effective manipulation of a 2D triangle mesh that defines a crowd's formation. These gestures enable a user to easily perform different styles of control over a 2D triangle mesh based on the number of active touch inputs. In turn this allows a user to control the formation of a crowd with varying levels of detail.

3.2 Method Overview

An outline of the system presented in this chapter can be seen in Figure 3.1. The system is composed of three layers:

1. **User input** - User input is received from the multi-touch device. This input specifies the overall movement and formation of the crowd desired by the user. This layer is responsible for converting the user input into control signals to manipulate a 2D triangle mesh that defines the crowd's formation (see Section 3.3.2).

2. **Mesh Deformation and Environment Interaction** - A 2D triangle mesh, representing the crowd's formation, is deformed according to constraints applied by the control signals from the user input layer (Section 3.3.1). This user-defined mesh configuration is further adjusted to account for interaction with the environment (Section 3.4), resulting in a mesh that defines the goal formation for the

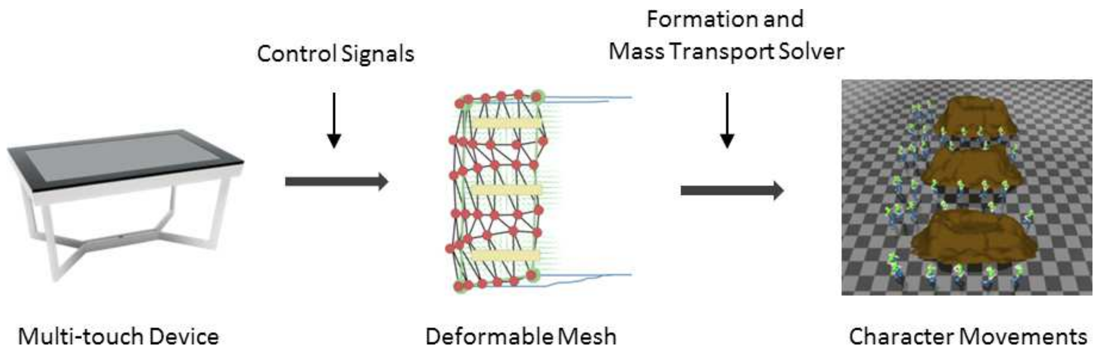


Figure 3.1: An overview of the proposed multitouch based control system.

crowd. In this way, the mesh acts as an intermediary to convert a user's high-level control into low-level control signals for the agents in the scene. It does this whilst considering the applicability of a user's desired formation within the current environment.

3. **Goal Assignment and Individual Character Movement** - Each agent's goal position in the final crowd formation is assigned using the solution to the mass transport problem (see Section 3.5). This layer also handles planning of an agent's route once their goal has been assigned.

3.3 Movement and Formation Control

In this section, we explain how we create a mesh to represent a crowd, and control the crowd with the user's input signals from the multi-touch device. We first describe the mesh representation and its deformation model (Section 3.3.1). Then, we explain our deformation scheme based on the input from the multi-touch device (Section 3.3.2).

3.3.1 Formation Representation and Control

For crowd motion control via a multi-touch interface, we adopt the deformable mesh representation, [Kwon *et al.* (2008); Sorkine *et al.* (2004); Takahashi *et al.* (2009)], as this allows complex shapes to be manipulated by low dimensional control signals. In this representation, a mesh, $\mathbf{F}$, consists of a set of vertices, $\mathbf{V}$, and the corresponding information on their connectivity, typically in the form of triangles, $\mathbf{T}$, such that $(\mathbf{V},\mathbf{T}) \in \mathbf{F}$. To specify the current target formation for the crowd at time t we define a 2D triangle mesh, $\mathbf{F}_c(t)$ (Figure 3.2, Top). For clarity in our experiments we use a rectangular shape mesh composed of a uniform triangle strip. The vertices of the mesh represent the goal positions for the agents in the crowd. The current scheme can be easily enhanced to handle arbitrary shapes by applying uniform sampling and Delaunay triangulation to generate the mesh based on a desired initial crowd configuration. Furthermore, it would be possible to handle any number of goal positions defined anywhere on the mesh through the use of barycentric coordinates, in a similar fashion to Ho *et al.* (2012). For simplicity, when discussing a goal position in the formation we refer to and use the notation for a vertex of the formation mesh, $\mathbf{v} \in \mathbf{F}$, but this can also represent any position on the user-defined shape.

The user interacts with the mesh by applying constraints using a multi-touch device (Figure 3.2, Middle). Let us define this constraint as a two-dimensional position with respect to time, $\mathbf{c}_i(t) \in \mathbf{C}$, where i is the index of this *control point*, and $\mathbf{C}$ denotes the set of all control points. When the user touches the multi-touch device at time t_0, the new control point, $\mathbf{c}_i(t_0)$, is set to the position of the nearest vertex on the mesh. The user drags their fingers across the multi-touch device to define a set of continuous spatio-temporal trajectories, $\mathbf{P}$, for the control points. Each of the trajectories, $\mathbf{p}_i \in \mathbf{P}$ specify where the corresponding control points must pass in the future frames. We represent each trajectory as a set of two-dimensional waypoints by dividing the trajectory into segments of a pre-defined length. For each timestep, the target location of each control point $\mathbf{p}_{i,j}$ is defined based on the jth waypoint in the corresponding user drawn trajectory $\mathbf{p}_i$. To govern the speed at which a control point can move we add a parameter, s, yielding the following update function:

$$\mathbf{c}_i(t+\Delta t) = \frac{\mathbf{p}_{i,j} - \mathbf{c}_i(t)}{\|\mathbf{p}_{i,j} - \mathbf{c}_i(t)\|} s\Delta t. \tag{3.1}$$

We pass the current location of each control point $\mathbf{c}_i(t)$ and the set of vertices from the previous formation mesh $\mathbf{F}_c(t-\Delta t)$ into an as-rigid-as-possible (ARAP) transformation solver [Igarashi *et al.* (2005)] to generate a user-defined deformed mesh, $\mathbf{F}_u(t)$, for that timestep. The ARAP deformation scheme is used as it is a well established algorithm for mesh manipulation and has been used successfully for crowd formation control [Kwon *et al.* (2008)]. The ARAP approach also permits global control over the mesh with only a few constraints. This helps to keep the number of required user inputs at a minimum, reducing the complexity of the control scheme. The combination of the moving control points and the ARAP solver causes the mesh to follow the user-defined trajectory and as such the crowd moves (Figure 3.2, Bottom). To allow the agents in the crowd to follow the mesh well, the speed of the control point, s, is set

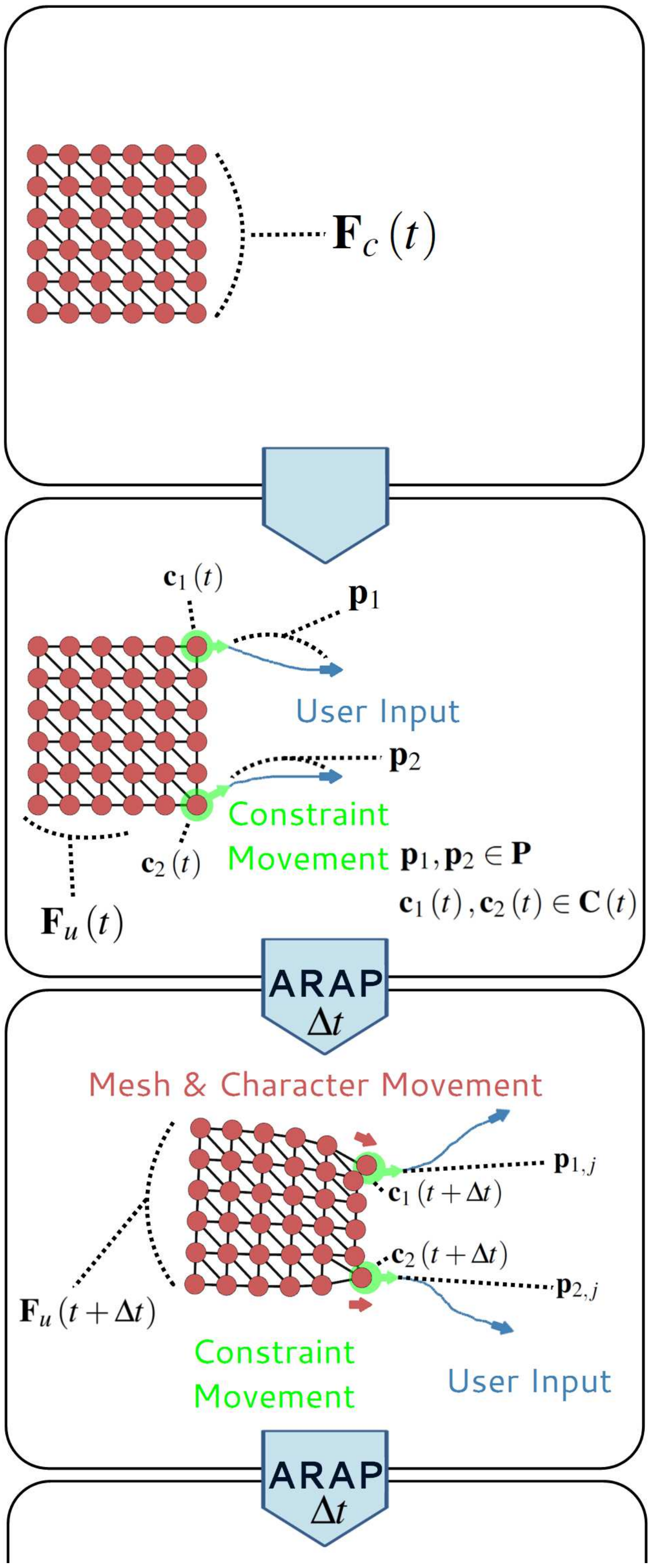


Figure 3.2: Application of the user input signals to control of the formation mesh. The formation of the crowd is represented by a 2D triangle mesh (Top). The user uses a multi-touch device to define trajectories of constraints applied to the mesh vertices (Middle). An as-rigid-as-possible (ARAP) solver [Igarashi *et al.* (2005)] is used to deform the mesh at each timestep according to the current position of the user's constraints (Bottom).

equal to the speed of the agents.

As soon as a control point is defined it immediately proceeds along the corresponding trajectory specified by the user. Since the movement of the control points deforms the formation mesh and causes it to move, the mesh, and subsequently the crowd, starts to move once a control signal is received. Whilst the user is providing input to the multitouch device the set of trajectories is updated but the control points do not track the current position of the user's touch inputs. Instead, the control points follow the history of the user's input signal (represented by the blue trajectories in Figure 3.2) at a speed determined by our system. In this way, the user does not always have to be interacting with the multitouch device in order to move the crowd; the user can define the trajectories for the mesh control points, remove their fingers from the multitouch device, and the crowd will continue to perform the specified movement as long as there are still trajectories to follow. By having the user provide input in this way, we can prevent unnatural fast movement of the crowd that would result if the formation followed the user's input directly and, for example, the user swiped their fingers across the multitouch device. Furthermore, if the target formation mesh were set to follow the current user's touch inputs, instead of using the current approach, any intermediate control, such as quickly expanding and then contracting the crowd, would not be maintained. At this point, $\mathbf{F}_u(t)$, known as the user formation mesh, represents the user's desired crowd formation at the current timestep; the formation's interaction with the environment is not considered. $\mathbf{F}_u(t)$ is subjected to deformation based on the environment in a later stage (Section 3.4).

3.3.2 Point, Line and Area Controls

Although a multi-touch device enables the use of several simultaneous user inputs there is a limit on how many of these inputs can be handled at any one time. This constraint comes either from the hardware (the majority of devices cannot handle more than four simultaneous touch inputs) or the limited number or dexterity of a user's fingers. As a result, not all of the vertices on our deformable mesh can be controlled explicitly with a user's touch inputs. In this section we propose different control schemes to overcome the limitation of multi-touch systems and produce a wider variety of control signals for manipulating the deformable mesh.

In our method a user applies their touch inputs directly onto the vertices of the deformable mesh. As a result, the resolution of the mesh affects the level of detail to which a user can specify the crowd's formation. It is straightforward to specify a shape on a low resolution mesh however the user is limited in the kinds of shapes that can be produced. Alternatively, a high resolution mesh provides flexibility for defining the crowd formation, but the number of touch points a user can manipulate with a multi-touch device is limited. As a result, the user can only directly control a subset of vertices on the mesh. Using the traditional point-based control system [Igarashi *et al.* (2005)], it is difficult for the user to control the rigidity when deforming the mesh. That is, when dragging a control point on a mesh, the system does not know how much the neighbour vertices should follow such a control point. Igarashi *et al.* (2005) solve the problem by allowing the user to predefine the rigidity of the mesh manually, however, this is not a plausible option for use in real-time control. We therefore present a set of controls, namely line and area controls, in addition to the point-based control scheme. These controls provide the user with the ability to manipulate the mesh with varying levels of rigidity.

The line-based control system (Figure 3.3) constrains the vertices of the control mesh that lie between two points specified by the user. When two control points $\mathbf{c}_0$ and $\mathbf{c}_1$ are defined, the vertices between them are sampled as supplementary control points. In our ARAP solver, these additional control points are applied as soft constraints as they are allowed to be affected by the environment in a similar way to the rest of the uncontrolled vertices on the mesh (see Section 3.4). When $\mathbf{c}_0$ and $\mathbf{c}_1$ are moved, the target location of the supplementary control points are computed by linearly interpolating the updated positions of $\mathbf{c}_0$ and $\mathbf{c}_1$. Defining multiple line constraints on the mesh allows the user to manipulate different sections of the mesh in different ways simultaneously.

We also propose an area-based control (Figure 3.4) that provides rigidity to a two-dimensional portion of the mesh. The section on which this control is applied is determined by the convex hull of three or more user-defined control points. When the area control is created, supplementary control points are sampled along the edges and inside its convex hull. They act as soft constraints on the mesh in the same way as in the line control. The mean value coordinates [Floater (2003)] of these supplementary control points are computed and are subsequently used to update their positions throughout the lifetime of the area control. In this way, the user can use only a few control points to manipulate varying proportions of the mesh.

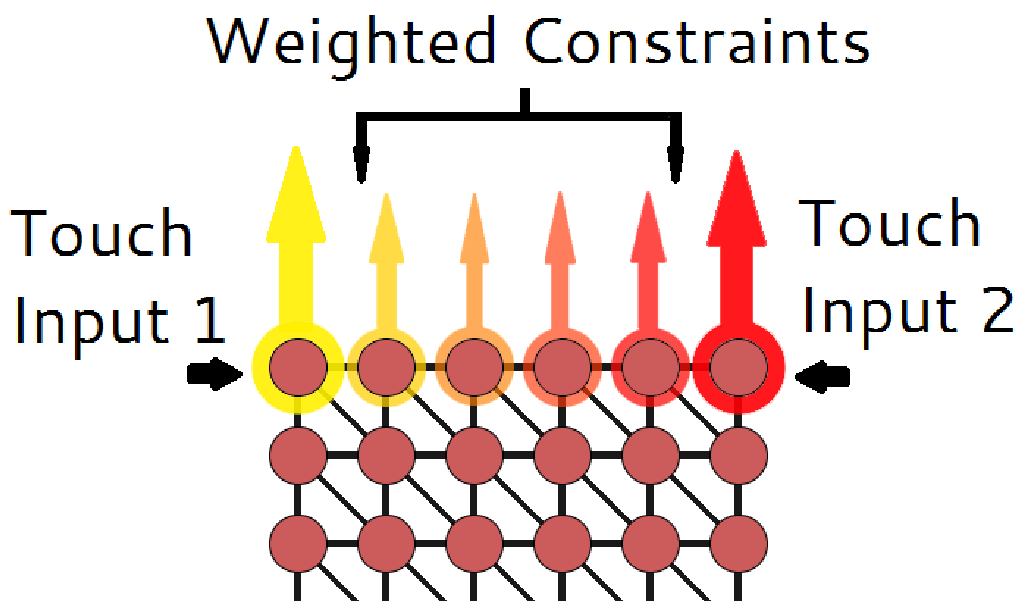


Figure 3.3: The proposed line-based control scheme for manipulating the formation mesh. The motion from the user's input constraints (large coloured arrows) is interpolated across a line in the mesh defined by the touch input positions.

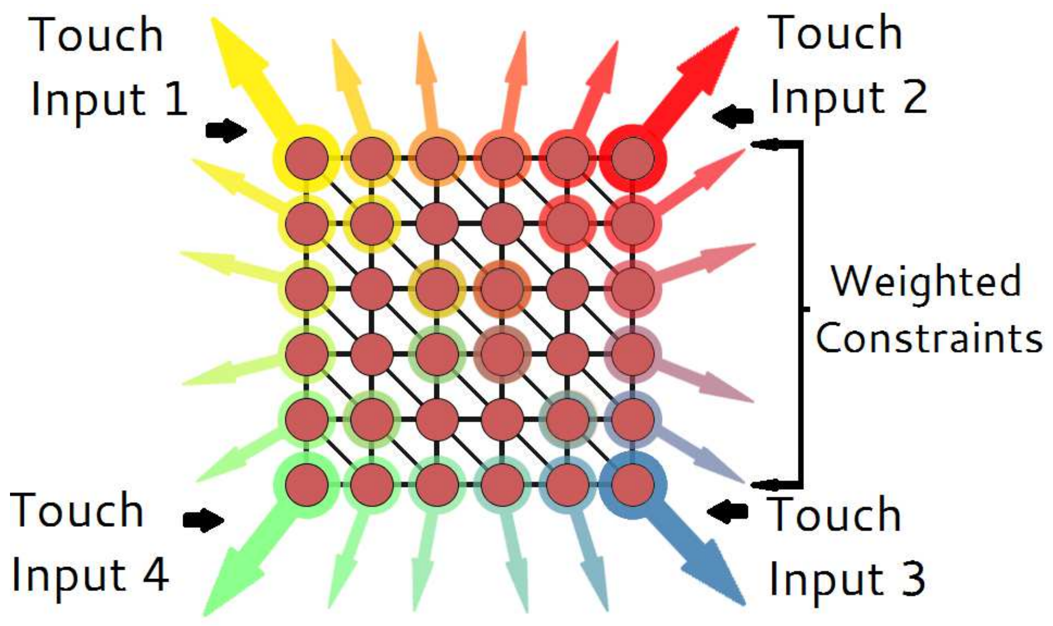


Figure 3.4: The proposed area-based control scheme for manipulating the formation mesh. The motion from the user's input constraints (large coloured arrows) is interpolated across the area of the mesh defined by the touch input positions.

The effects of using the three different control schemes; point, line, and area control, can be seen in Figure 3.5. Given the same control signal (Figure 3.5(a)) but different control schemes, the final formation is different (Figure 3.5(b) - (d)). The different forms of control confer varying levels of rigidity to the mesh, giving the user greater flexibility in the kinds of formations they can create with a small number of user inputs.

In order to identify the type of control that a user wants to apply, the timing that the fingers are placed on the multi-touch screen is examined. A single touch input gives basic point-based control, two simultaneous touch points indicates a line control, whilst three or more simultaneous touch points creates an area control. This scheme allows for different kinds of control to be applied simultaneously to different parts of the mesh. In Figure 3.6(a), we show an example where the user applies a line control at the left of a square and an area control on its right. The result when the user drags these areas is shown in Figure 3.6(b). It can be observed that the left half of the shape is deformed while the right part is kept rigid thanks to the two types of control used.

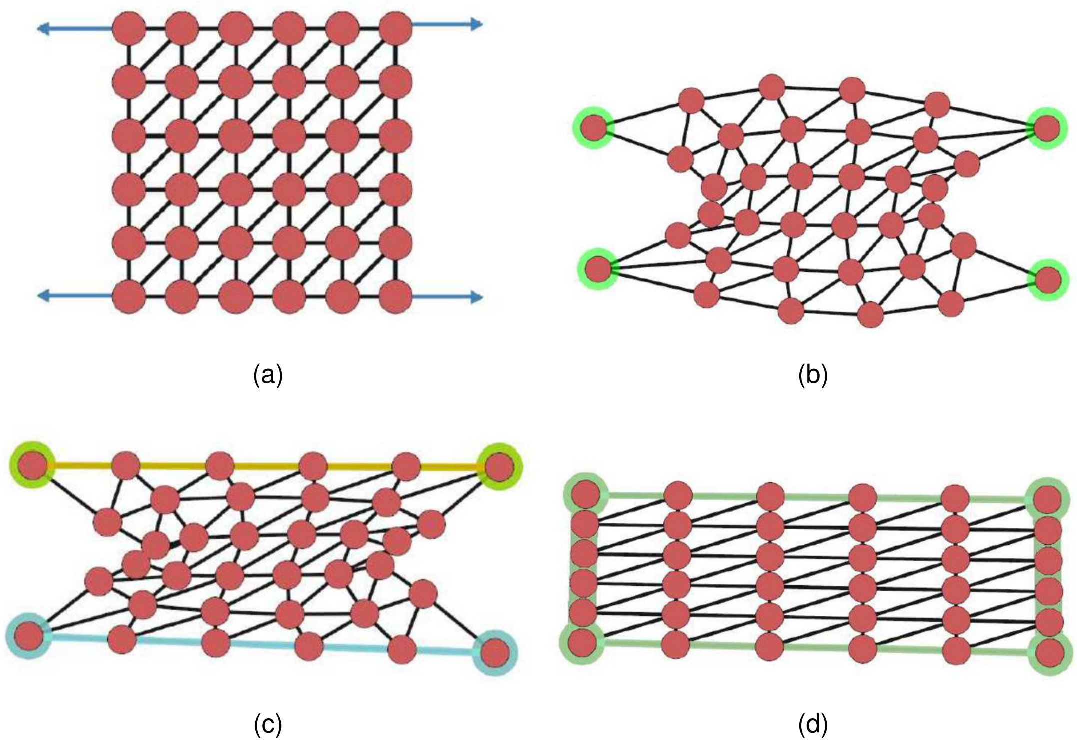


Figure 3.5: (a) Given the same movement of the user's touch inputs , a rectangular mesh will deform differently when using (b) point-based control, (c) line-based control, and (d) area-based control on the four corners.

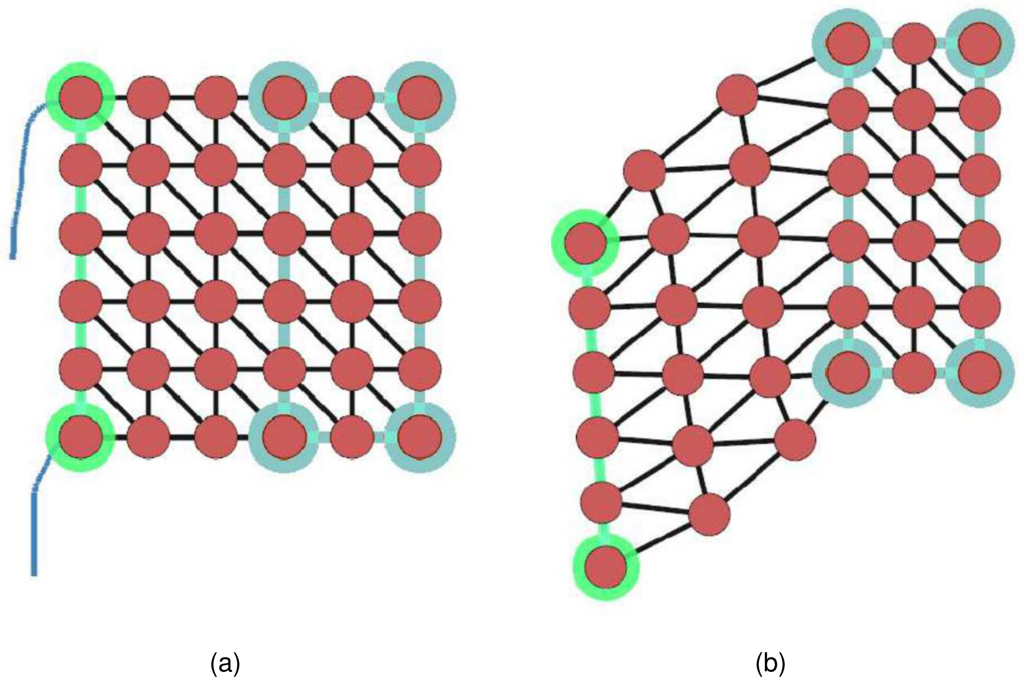


Figure 3.6: A user can apply different styles of control to the mesh simultaneously. (a) The user applies a line control and an area control onto the same mesh. (b) The resultant deformation.

3.4 Environment-Guided Mesh Deformation

In this section, we explain how we deform the control mesh of the crowd according to its interaction with the environment. This scheme is especially important for achieving effective obstacle avoidance in scenes where there are multiple obstacles, such as city scenes with several streets that are diverging and merging. The deformation of the control mesh is guided by a velocity field generated by the environment. By letting these low-level interactions be controlled by our system, we allow the user to concentrate on the higher level control of the crowd.

The environment is modelled with a set of objects, O. Each object, $o \in O$, generates a velocity field, ω_o, that will affect near-by vertices of the control mesh. Referring to Figure 3.7, the direction of the vector in the velocity field at point $\mathbf{x}$ is the unit vector from the centre of the object, $\mathbf{c}_o$, to the point $\mathbf{x}$. The magnitude of the vector in the velocity field is computed based on the distance, d, between the object's centre and the sample point:

$$f(d) = \begin{cases} 1 - \frac{d}{r_o} \; (0 < d < r_o) \\ 0 \; (r_o \leq d), \end{cases} \tag{3.2}$$

where r_o is a predefined range of influence for the object o. We divide the floor into grid cells of equal size, and compute the vector at each cell for the velocity field produced by all the objects in the environment:

$$\Omega(\mathbf{x}) = \sum_{o \in O} \omega_o(\mathbf{x}) = \sum_{o \in O} f(||\mathbf{x} - \mathbf{p}_o||) \frac{\mathbf{x} - \mathbf{c}_o}{||\mathbf{x} - \mathbf{c}_o||} \tag{3.3}$$

where $\mathbf{x}$ is the position at the centre of the cell, $\mathbf{p}_o$ is the closest point to $\mathbf{x}$ on the surface of obstacle o and $\omega_o(\mathbf{x})$ is the velocity field value for object o at $\mathbf{x}$. We use this approach because simply using the distance field can cause vertices to move slowly when there are long edges on the obstacle. The direction vector pointing away from the obstacle centre increases the tangent element of the vector field in such cases.

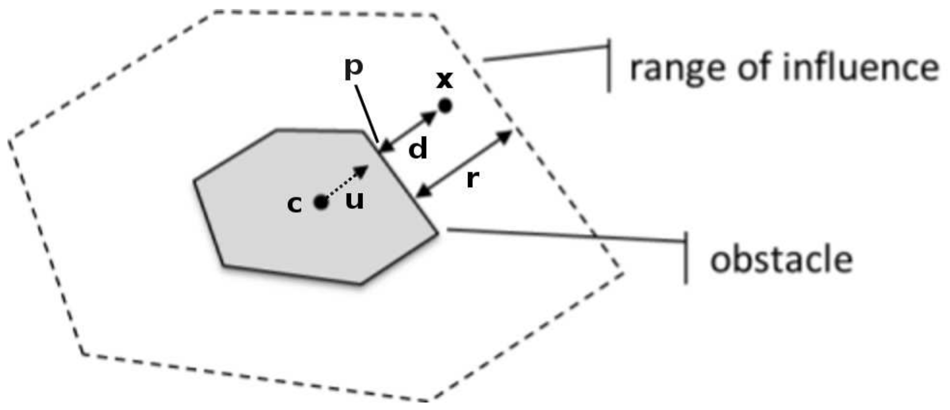


Figure 3.7: Calculating the velocity field produced by environment obstacles. The field produced by the obstacle at point $\mathbf{x}$ is dependent on the distance (d) of $\mathbf{x}$ from the obstacle's closest point ($\mathbf{p}$) relative to the range parameter (r), as well as $\mathbf{x}$'s relative direction, $\mathbf{u}$, from the obstacle centre ($\mathbf{c}$).

Given a user formation mesh from the previous stage, $\mathbf{F}_u(t)$, we examine the current position of each vertex of the control mesh, $\mathbf{v}_i(t) \in \mathbf{F}_u(t)$ and sum the contribution of each velocity field produced by all the obstacles at that position. We also monitor the collisions between the vertices and the obstacles, and push them out to the nearest point on the surface if they penetrate through the obstacle. The edges of the control mesh are allowed to pass through the environment. The vertex positions of $\mathbf{F}_u(t)$ are updated based on the field and the final formation mesh $\mathbf{F}_c(t)$ is then computed:

$$\mathbf{F}_c(t) = \mathbf{F}_u(t) + \Omega(t) = \mathbf{v}_i(t) + \Omega(\mathbf{v}_i(t)) \quad \forall_{\mathbf{v}_i \in \mathbf{F}_u}, \tag{3.4}$$

where $\Omega(t)$ is the state of Ω at time t as it updates according to the current state of the objects in the environment. In order to prevent the control mesh getting stuck in the environment, we limit the obstacles shapes to convex hulls, as well as the minimum distance between two obstacles.

3.5 Character Mapping

Once the configuration $\mathbf{F}_c(t)$ for the formation control mesh is decided we next have to determine which point on the mesh each agent will move to. To minimise obstructions between agents in the crowd during transition to the new formation it is necessary to assign an agent a target position based on their current configuration.

In order to assign each agent's goal position in the user-defined formation we use a formulation for solving the transportation problem, which is used to compute the Earth Mover's Distance [Rubner *et al.* (1998)]. The transportation problem is solved by minimising the amount of work to move objects from a set of source locations I to a set of target locations J. The solution to the problem consists of finding the amounts of the objects to be transported across all routes, henceforth referred to as "flows" $(f_{i,j})$, that minimises the overall cost of transportation between the two point sets. A set of flows can be evaluated using:

$$\sum_{i\in I}\sum_{j\in J} c_{i,j} f_{i,j}, \tag{3.5}$$

subject to the following constraints:

$$f_{i,j} \geq 0 \quad i \in I, j \in J \tag{3.6}$$

$$\sum_{i\in I} f_{ij} = y_i \quad j \in J \tag{3.7}$$

$$\sum_{j\in J} f_{ij} \leq x_i \quad i \in I, \tag{3.8}$$

where $c_{i,j}$ is the cost of travelling from point $i \in I$ to point $j \in J$, x_i is the total supply of source point i (the supplier) and y_j is the total capacity of target point j (the consumer). This cost is assessed for the full connectivity of the two point-sets. Readers are referred to Rubner *et al.* (1998) for further details. In this work, the source points correspond to the locations of the agents and the target points are the vertices of the formation mesh $\mathbf{F}_c(t)$ at the current simulation step. Each point $i \in I$ and $j \in J$ can be weighted to allow user-defined partial/full matching between the two point-sets. These weights act as the supply and demand signals from the source and goal points respectively. The flows that minimise Equation (3.5) satisfy these signals. For example, for the purposes of mass transport, low supply weight from a point $i \in I$ and high demand weights on points in J can produce solutions with many source points feeding to a single goal point. This is effective for controlling several agents in groups with fewer target locations in the formation, particularly if multiple agents can use the same cost-to-goal information that is computed once per target location (Section 5.3.1). In the current work it is desirable for each agent to be assigned to only one goal point and vice-versa. To achieve this we assign a weight of 1 to all source and target points to allow full mapping from current agent positions to candidate locations in the mesh. The Euclidean distance between an agent and a formation location on the mesh is used for the cost of travel, and the solution to the transportation problem provides a set of point-to-point correspondences between the agents and the target formation, which is recalculated at every time step.

Once a point in the mesh is assigned to an agent their route to the location is computed using grid-based A$*$ search on a binary occupancy grid with similar resolution to the velocity field grid. Each character considers other characters as obstacles and computes the optimal path. The characters then move to their corresponding target locations with a simple PD controller. A maximum speed is defined to avoid unnatural fast movement.

3.6 Experimental Results

In order to highlight the benefits of our approach, we have produced scenes showing a group of characters passing through different static environments including a gateway, corridors, woodlands, a city area, and a dynamic environment where cars are moving around. The formation of the characters is manipulated in some of the examples such that they can pass through narrow pathways or produce visual effects. We also show an experiment that presents the advantage of using the mass transport solver for mapping each character to a vertex of the control mesh. All the examples were produced starting from a uniform rectangular formation with 36 characters, except the city example, which consists of 100 characters.

3.6.1 Environments with Static Obstacles

We first show an example in which the characters pass through an environment produced by three long, rectangular obstacles. Three different ways to pass the crowd through the environment are produced: (1) fully expanding them such that they pass between the obstacles as well as outside them, (2) unevenly split them to make two groups pass through the two corridors produced by the obstacles, and (3) squeezing them into a single corridor by stretching the formation parallel to the obstacles. Some of the snapshots are shown in Figure 3.8(a) and (b). Notice that the movement of individual characters is not defined explicitly by the user. Instead, the user must only specify high-level properties of the simulation such as the general width and direction of the crowd's formation. Given the user's commands the system handles the low-level interaction of both the formation and the characters with the environment. As such, based on the formation defined, the characters fit into the appropriate pathways automatically. Experiments show how a general flow of the crowd can be achieved through a simple scene using only a single gesture (Figure 3.8(a)). Alternatively, the user can specify a more precise movement of the crowd in the scene by directing the formation mesh through a single corridor in the environment (Figure 3.8(b)). We also produce another example where a larger scale of crowd passes through an urban area with many buildings, cars and trees (see Figure 3.8(c)). Even in such a complex condition, the characters have no difficulty moving through the area.

3.6.2 Environments with Dynamic Obstacles

In this example, the characters are controlled to pass through an environment in which there are multiple dynamic obstacles (cars). In the open environment the formation mesh is defined as a result of the user's input signals and so the characters track the formation specified by the user. When the cars approach the formation mesh, the mesh deforms automatically according to the velocity field produced by the cars in order to avoid collisions (see Figure 3.8(d)). As such, the characters follow the deformed vertices and subsequently are able to move easily past the dynamic obstacles. We can tune the strength of the velocity field to adjust the distance at which a character starts to avoid the obstacles depending on the application. Even when the mesh is being deformed by the obstacles, there still exists a signal for the mesh to follow the user's controls thanks to the addition of the as-rigid-as-possible deformation scheme. This results in movement of the mesh that satisfies a user's direction as best as possible whilst still accounting for objects in the environment.

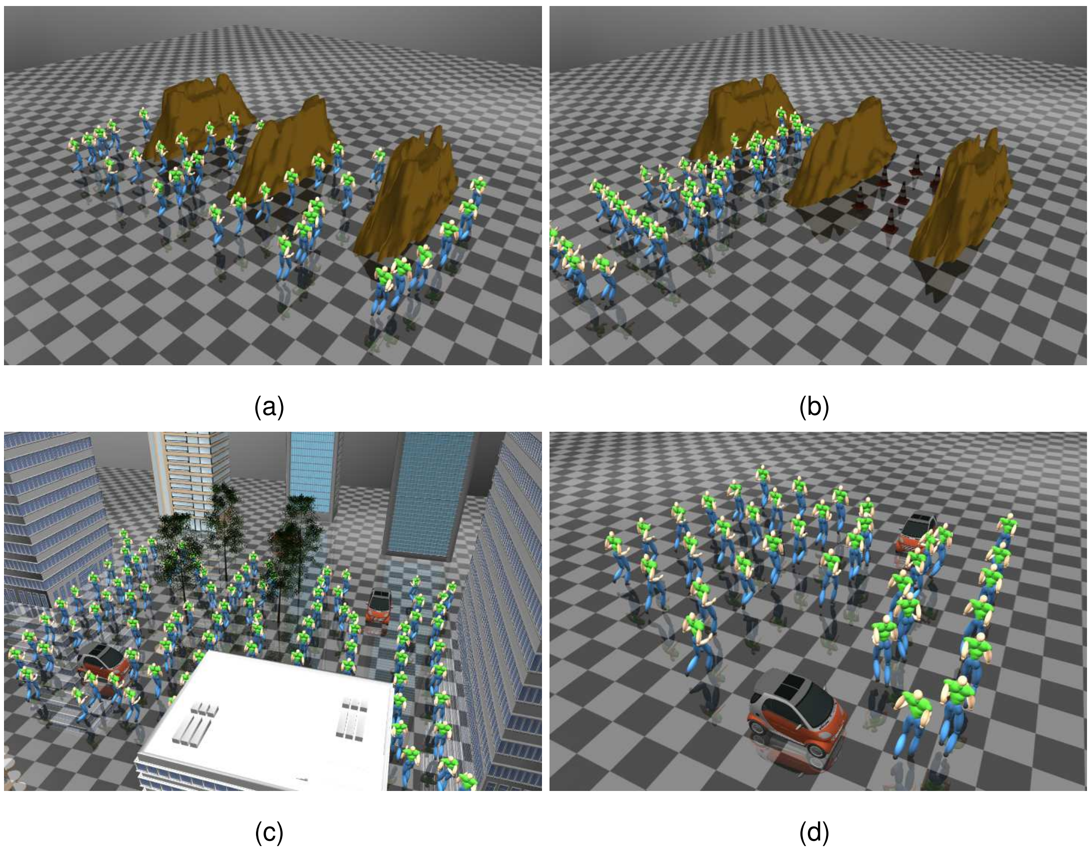


Figure 3.8: A crowd moving through (a) several corridors by spreading out, (b) a single corridor by squeezing into one row, (c) a city area, and (d) a crowd avoiding cars.

3.6.3 Formation Manipulation

Here we show examples of the different levels of formation manipulation in our system that can be used to produce different visual effects. We first describe how a user can use the current framework to define various formation shapes interactively. Examples are given showing the system's ability to interpolate between these different high-level crowd formations quickly and accurately. We then show how a user can control various aspects of these formations to carry out certain tasks. We synthesized formations including shapes of "Pacman", an arrow, a letter "L" and a star (Figure 3.9). To create these shapes interactively the controls presented in Section 3.3.2 are used. By employing two separate line controls on one side of the same mesh a rectangular formation can be bent into the mouth of the "Pacman" (Figure 3.9(a)). Area controls can be used to expand and shear the formation, as well as to maintain the rigidity of certain portions of the mesh whilst manipulating other areas. This can be seen with the "L" shape formation (Figure 3.9(b)). In this example, a line control is used to extend the lower portion of the formation whilst an area control keeps the rest of the formation rigid. Similarly, an arrow shape (Figure 3.9(c)) can be produced by using an area control for keeping the arrow head rigid while manipulating the tail with point and line controls. Point controls can be used to further refine certain parts of the formation. This can be seen in the star formation example (Figure 3.9(d)) where the points of the star are coarsely defined by initially squeezing certain parts of the formation using line controls. The formation is further refined using point controls to achieve the desired final shape.

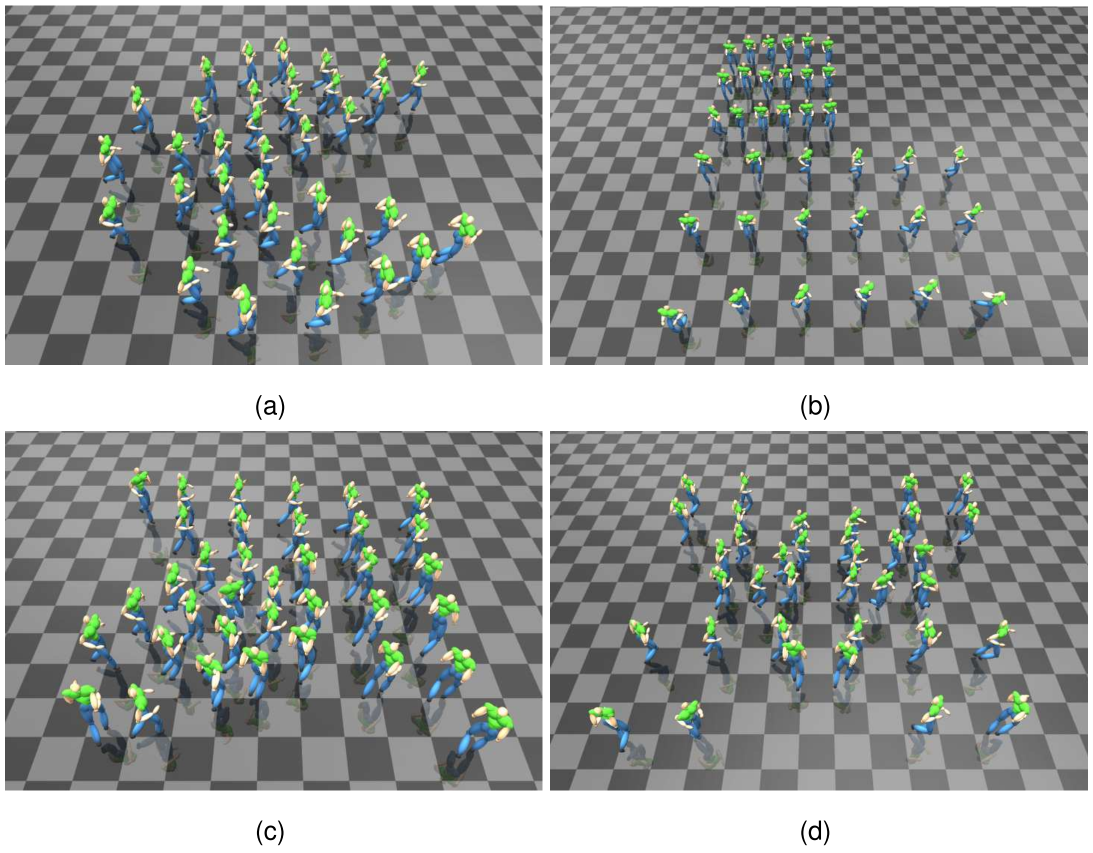
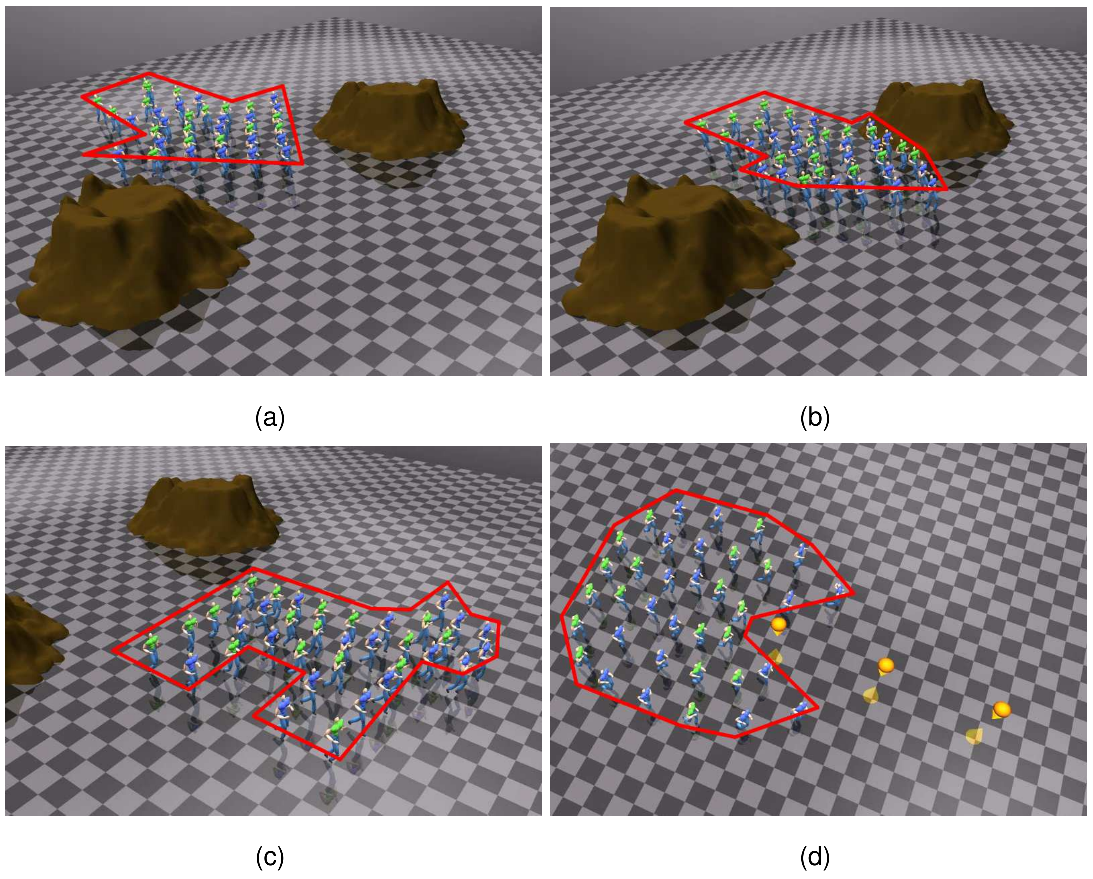


Figure 3.9: A crowd deformed into the shape of (a) "Pacman", (b) a letter "L", (c) an arrow and (d) a star.

As well as defining the crowd's formation on the fly, a set of mesh shapes can be registered to define formations that the crowd can switch between. Some examples of these formations can be seen in Figure 3.10. The user is able to specify the trajectory of the crowd using the multi-touch device and the crowd's formation can be switched depending on the environment. In the current demo, the formation is switched when the crowd passes over pre-defined checkpoints embedded in the environment, however, it would be reasonable to allow the user to handle this switching either through a basic button interface or a set of multi-touch gestures. Figure 3.10(a) shows the crowd in an arrow formation which is effective for passing through a narrow corridor. This formation can be directly switched to from the original square formation. The sides of the arrow deform temporarily when the formation is inside the corridor to ensure that the characters pass through without trouble (Figure 3.10(b)). Once through the corridor the crowd can switch back to their original formation or to another formation entirely, depending on the user's requirements (Figure 3.10(c)). By using the mass transport solver to assign character's goal positions, transitions between different formations occur quickly and with minimal congestion.



(a) (b) (c) (d)

Figure 3.10: Examples of high and low level control of crowd formation. A crowd can be seen adapting it's formation based on user signals and the environment: (a) An arrow formation is used to pass easily through the corridor, (b) The crowd formation is affected by the surrounding environment, (c) The crowd can easily transition to a variety of formations. (d) The current crowd formation can be easily manipulated to achieve certain tasks.

Not only does our system provide methods for defining high-level formation shapes, but a user is also able to manipulate individual formations directly in order to provide more detailed and intricate crowd motion. We show an example where the crowd formation resembles a ”pacman” character (Figure 3.10(d)). A set of simple gestures can provide interactive control over multiple aspects of the formation. In this case the user is able to translate the formation whilst simultaneously manipulating the ”pacman” character’s mouth. This helps to create visual effects as well as perform certain tasks, such as collecting items in the environment.

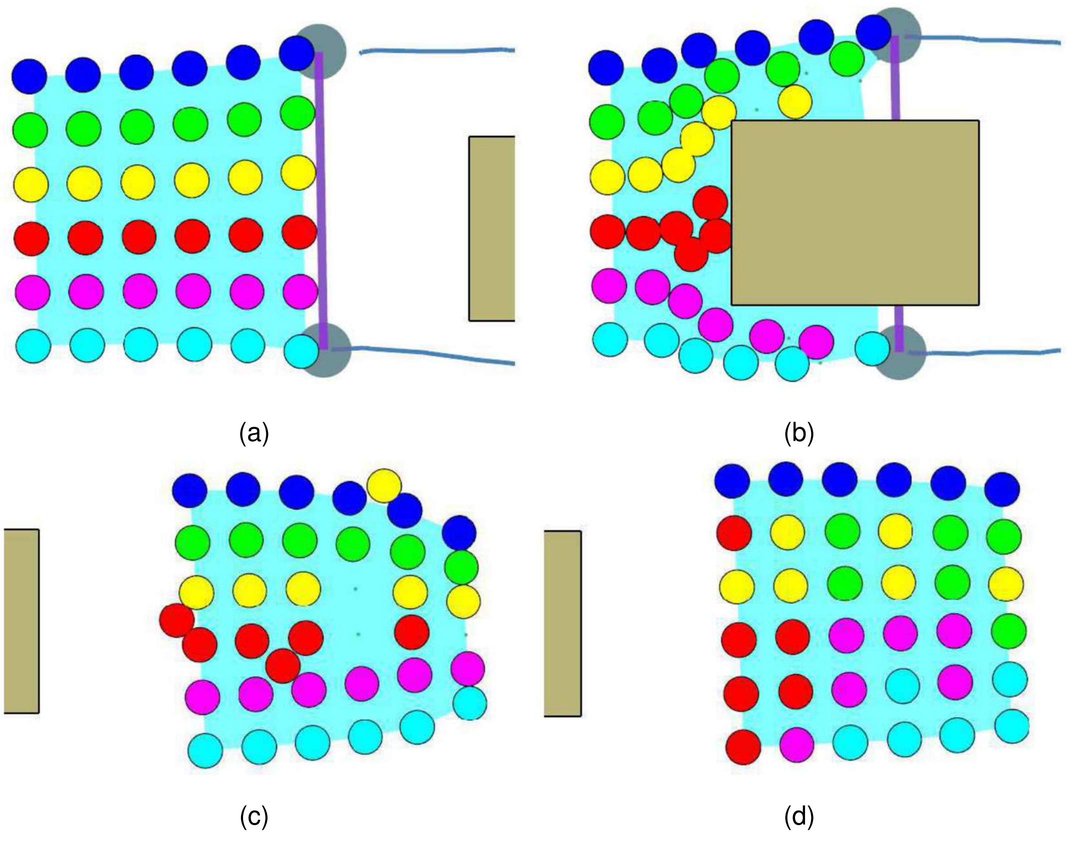


Figure 3.11: Effect of using the mass transport solver: (a) the crowd is controlled towards an obstacle, (b) characters collide with the obstacle during their motion, (c) the final state when the locations for the characters are fixed in the formation, and (d) final state when using the mass transport solver to compute the optimal final locations.

3.6.4 Mass Transport Solver

In the last experiment, we show examples that clarify the advantage of using the mass transport solver for guiding the characters (Figure 3.11). In the example, the characters in a square formation are supposed to pass around an obstacle and merge again (Figure 3.11(a)). Because some of the characters are prevented from moving by the surrounding characters and the obstacle for a while (see Figure 3.11(b)), they are late to arrive to the group once they have passed the obstacle. This sort of situation is quite common in interactive applications where a user wishes to move a crowd quite coarsely through an environment containing several obstacles. This increases the likelihood of collisions with obstacles or other characters that can perturb the relative configuration of the formation. In the case where the characters are required to return to their original position in the formation, they are blocked by the characters that filled in the row in advance (Figure 3.11(c)). This problem is particularly challenging in dense crowds where there is not enough space for the characters to pass through. In contrast, with our interpolation scheme based on the mass transport solver, the blocking character simply shifts into the formation to make room for the late arriver (see Figure 3.11(d)). Notice that the mapping of the characters to the mesh vertex in the final formation is different from the initial formation.

3.6.5 Computational Costs and 3D Rendering

The experiments are run on one core of a Core i7 2.67GHz CPU with 1GB of memory. For the multi-touch input we used an MTMini device [Sandler (2010)] along with the associated open-source Community Core Vision software [Sandler (2008)] for tracking touch points. The computation of the 2D trajectories that includes the deforming of the control mesh, reshaping it through its interaction with the environment, computing of the character destination by the mass transport solver, and updating their positions are all done in real-time at a rate of 60 frames per second. The final 3D scene involves computing the movements of each character. We created a simple locomotion database with running motions. Based on the planned movement trajectory, the characters select the optimal motions with a precomputed search tree [Lau & Kuffner (2005)]. We allow minor adjustments in the original motion to better fit the movement trajectory, and apply inverse kinematics to fix the supporting foot on the floor. The motion planning process is in real-time, but the rendering process is done offline due to the large number of characters and the lack of rendering optimization such as level-of-detail.

3.7 Discussion

In this research we present a novel method for effective user-guided control of crowd formation and motion in virtual environments. Currently, formation controls in interactive applications, especially computer games, are rather basic. In most cases, a group of characters is moved from one location to another by simple mouse control. As the dimensionality of the user control is limited, the only solution is to let low level character-character or character-environment interactions be handled by the system. The current work utilises this idea to allow more refined control over a crowd's formation whilst still keeping the necessary control signals relatively simple. Our method provides a more enriching user experience in real-time applications such as games. The method is particularly well suited to applications where group cohesiveness is important e.g. real-time strategy games or social group motion in crowds.

3.7.1 Multitouch control

Section 2.2 discussed how previous approaches to crowd control interfaces require a user to sequentially define multiple paths for a crowd to follow. In the work presented in this chapter, such trajectories are replaced by the trajectories of the user's fingers on the multi-touch device for real-time control. The subtle control signals from the fingers are used to deform the mesh and alter the way it interacts with obstacles and the environment. Our method combines formation tracking and obstacle avoidance into a single control signal. This is in contrast to Ho *et al.* (2010) who handle the impact of the environment by allowing agents to disengage formation tracking whilst avoiding obstacles. By providing a combined control signal we enable a crowd to continue to track the user's specified formation whilst avoiding obstacles in the environment. We have shown that the user can control the characters in various ways to move through the environment by subtly changing the way they control the formation via a multi-touch device.

3.7.2 Mass Transport Solver

Various methods exist for handling congestion in multi-agent path planning including, but not limited to, replanning [Koenig & Likhachev (2002)], space-time planning [Singh *et al.* (2011); Karamouzas *et al.* (2013)] and prioritised planning [Lau & Kuffner (2006); Sung *et al.* (2005); van den Berg & Overmars (2005)]. In each of these cases, goals are assigned sequentially making it possible that they are not assigned optimally. Centralised planners [Li & Chou (2003); Schwartz & Sharir (1983)] compute multi-agent motion simultaneously in order to find a good solution for all agents however, full plans must be computed for each agent, making these approaches costly. By using the MTS we can assign agent goals before any planning occurs, all that is needed is a distance from each agent to the target locations.

By using the MTS, our method enables a crowd to dynamically assign goal positions in the final formation leading to better tracking of the user-defined formation under perturbations. A similar approach is adopted in Alonso-Mora *et al.* (2011) however, there are a few key differences. Firstly, the authors use the *Hungarian algorithm* to solve the *assignment problem*. In this problem, the objective is to assign one agent to one and only one task. In this research, we solve the *transportation problem*, where multiple agents can be assigned to one task (or vice versa) by adjusting their weighting during optimisation. A one-to-one mapping can still be achieved by defining appropriate weights but the framework can also handle discrepancies between the number of agents and the number of goals, all in a single optimisation process. Secondly, we demonstrate the use of goal reassignment with larger numbers of agents and in the presence of static and dynamic obstacles, where perturbations are much greater. We show how this approach is effective for resolving congestion and enabling the crowd to follow the user-defined formation well. This kind of behaviour is simulated in Ho *et al.* (2012) however, the authors perform goal swapping by minimising travel distance in a locally pairwise fashion. This can mean that the final target positions for the agents are not optimal with regards to the group, causing oscillations or conflict between the movement of the agents. Finally, in later work we implement an environment-aware cost metric into our optimisation to account for the effect of complex environments on goal assignment (Section 5.3).

3.7.3 Scalability

With the current framework, the computational cost of the MTS imposes a bottleneck on the size of the crowd that can be controlled interactively. The time required to carry out the MTS is dependent on the number of characters in the simulation and subsequently the number of goals associated with these characters. Currently, the resolution of the mesh defines the number of characters that can be controlled by the system since each vertex of the mesh represents a goal position for a character. It is desirable to keep a relatively high resolution mesh to allow for interesting formation shapes to be defined, however, if this becomes too high the mesh can become uncontrollable and the computational cost of the as-rigid-as-possible solver can become prohibitive. Instead, it is appropriate to keep the resolution of the mesh fixed at a desired level for controllability and to define the character's goal positions in the formation parametrically by using, for example, barycentric coordinates. Additional speed up could be achieved by assigning the goal position for groups of characters as opposed to individuals in the formation. A hierarchical system could be used to assign varying sized groups to their appropriate positions in the formation, enabling simulation of much larger crowds.

3.7.4 Formation Representation and Crowd Movement

For this work we chose to represent the formation of the crowd using a two-dimensional mesh. This choice imposes certain limits on the type of crowd movement that can be produced using our system. Firstly, the rigid shape of the mesh makes it very easy for a user to generate structured crowd movement, where characters move as a cohesive unit and maintain the spatial layout of the crowd formation. In a lot of real crowds this kind of motion would seem a little unnatural. A lot of realistic scenarios involve more heterogeneous motion, where people would move around by themselves, sometimes clustering into groups but generally intermingling and displaying individuality in their behaviour. Our system does not focus on producing this kind of motion but is more suited for use when ordered crowd movement is preferable. This is often the case in video games, where a user is more concerned with the crowd following their instructions in order to achieve a certain task than with the crowd showing anisotropic movement. This is shown by our user study in Chapter 4, where control of agents in a more structured way is useful when guiding a team to achieve a goal, such as coverage and collection of objects in an environment. Furthermore, good control over the formation of a crowd of agents allows a user to create shapes that can be useful for storytelling and performances.

A further advantage of using a mesh to represent the crowd's formation is that there exist a number of approaches for mesh deformation [Igarashi *et al.* (2005); Weng *et al.* (2006); Wang *et al.* (2008); Yang *et al.* (2012)]. These methods permit a user to manipulate the shape of a mesh without significantly altering the underlying mesh configuration. This allows the spatial relationships specified in the formation to be well maintained. The mesh vertices also represent well-defined points of interaction for the user's touch input as well as the line and area gestures presented in Section 3.3.2. Conceivably, any mesh manipulation approach that permitted multiple simultaneous constraints could be applied in our algorithm. As noted by the authors in [Weng *et al.* (2006)], the number of iterations to solve in their approach can vary significantly depending on factors such as the shape and the amount of deformation to the mesh. This therefore could have a negative impact on interactive manipulation of the crowd shape. Similarly in Wang *et al.* (2008), an iterative solver is used for rigid shape matching for squares, akin to Igarashi *et al.* (2005)'s work for triangles. Yang *et al.* (2012)'s work is concerned with multi-element 2D meshes where shapes overlap one another so is not intended for the simpler meshes used in this work. Alternative shape manipulation techniques such as Green coordinates [Lipman *et al.* (2008)] and Mean Value coordinates [Floater (2003)] could be used though these are most effective when there is a well defined cage for containing the positions to be manipulated. As mentioned in Section 3.3.1, the as-rigid-as-possible scheme was chosen because of its effective use in previous research for group shape control [Kwon *et al.* (2008)] and in terms of computation it has the benefit of being based on an efficient linear system formulation. Furthermore, the approach is fairly straightforward to implement, with a C++ implementation readily available [Schmidt (2009)].

3.7.5 Character-environment interaction

Even though we have only presented actions such as running and avoiding in our examples, it is possible to add more complex interactions such as characters crawling under obstacles, side stepping along a narrow space, jumping over ditches and climbing up ladders. Such effects will be easier to produce by embedding the specific movements in the environment using patch-based approaches [Yersin *et al.* (2009); Lee *et al.* (2006)]. A patch-based approach would also enable us to simulate scenes of character-character interactions, such as two armies fighting [Shum *et al.* (2008)]. We present an approach to incorporating motion data into a crowd's motion planning and formation tracking using a simplified patch-based approach in Chapter 5.

3.7.6 Formation tracking

For the user to have good control over the behaviour of the crowd it is important that the characters are able to track the formation specified by the user throughout the simulation. In the current work, characters are able to maintain a close proximity to their goal position in simple environments. With increasing numbers of obstacles there arise cases in which the distance between a character and its assigned formation position becomes quite large. This can happen due to the character being unable to travel through the environment quickly enough to keep up with the formation, for example, the character may become temporarily stuck whilst passing by an obstacle or congestion may occur if several characters are attempting to pass through the same narrow area. One improvement to the current approach would be providing a feedback signal to the movement of the control mesh. This signal could be based on how well the agents are tracking the mesh or even the state of the mesh vertices, that is, whether they are interacting with an obstacle or not. This would help when a vertex or agent takes particularly long to negotiate an obstacle or large deformation of the mesh occurs as a result of the user forcing the mesh to collide with large obstacles. The system could handle these situations by adjusting the movement of the mesh accordingly. In Chapter 5 we explore a method for introducing a simple feedback loop between the formation mesh and the characters that enables better tracking of the formation, particularly in complex environments.

Additionally, the current method does not account for the future motion of dynamic obstacles when planning the mesh movement. This choice was made to give greater control of the mesh to the user. Having the mesh follow an optimal path rather than that specified by the user may make the user feel less in control of the crowd and thus degrade their experience. That being said, rather than using our simplified potential field for avoiding obstacles, an RVO-like obstacle avoidance mechanism [van den Berg *et al.* (2008a)] would provide greater intelligence to the mesh motion, for example preventing it from passing in front of moving cars, and consequently produce smoother motion. Replacing our simple A$*$ path planner for characters in static environments with one that can account for dynamic obstacles would also mean that characters themselves can avoid unrealistic movements when tracking the mesh. This would also help to minimise any discrepancies between the formation mesh and the characters' current positions.

3.8 Summary

Real-time crowd control has become an important research topic due to the recent advancement in console game quality and hardware processing capability. The degrees of freedom of a crowd is much higher than that provided by a standard user input device. As a result most crowd control systems require the user to design the crowd movements through multiple passes, such as first specifying the crowd's start and goal points, then providing the agent trajectories with intermediate waypoints or paths. Such a multi-pass control would spoil the responsiveness and excitement of real-time games. In this chapter, we proposed a new, single-pass algorithm to control crowds using a deformable mesh. When controlling crowds, we observe that most of the low level details are related to passive interactions between the crowd and the environment, such as obstacle avoidance and diverging/merging at cross points. Therefore, we simplify the crowd control problem by representing the crowd with a deformable mesh that passively reacts to the environment. As a result, the user can focus on high level control that is more important for context delivery. Our algorithm provides an efficient crowd control framework while maintaining the quality of the simulation, which is useful for real-time applications such as strategy games. We evaluate the effectiveness of our multitouch approach for user control in Chapter 4.

Chapter 4

User Evaluation of Multitouch Control

In this chapter we present an in-depth user study to appropriately evaluate the system described in Chapter 3. We analyze the capability of our user interface for interactively moving and defining crowd shape for task completion, and provide a comparison to traditional mouse-based controllers. Results show that our multitouch-based crowd controller enables a user to complete a basic item-collection task in a shorter amount of time and with fewer unique inputs as compared to the mouse-based controller. In the rest of this chapter we present the methodology used for the study (Section 4.2) followed by an analysis of the participants' ability to complete an item collection task (Section 4.3.1), and the general usage and feedback with regards to our multitouch controller and a common mouse-based controller (Section 4.3.3).

4.1 Contributions

- An evaluation of the usability of the previously proposed multi-touch crowd-control framework (Chapter 3) as compared to traditional mouse-based control methods. We provide a comparison of each scheme's ability to control and manipulate a crowd's motion and formation in an interactive application.

4.2 Method

In the study we had a total of 15 participants, consisting largely of postgraduate students all aged between 20 and 35. To compare our system to other user-control approaches we implemented a mouse controller based on those found in current real-time strategy games. This controller included a basic mouse control interface (Figure 4.1) and the movement of the characters was determined using the approach in Treuille *et al.* (2006).

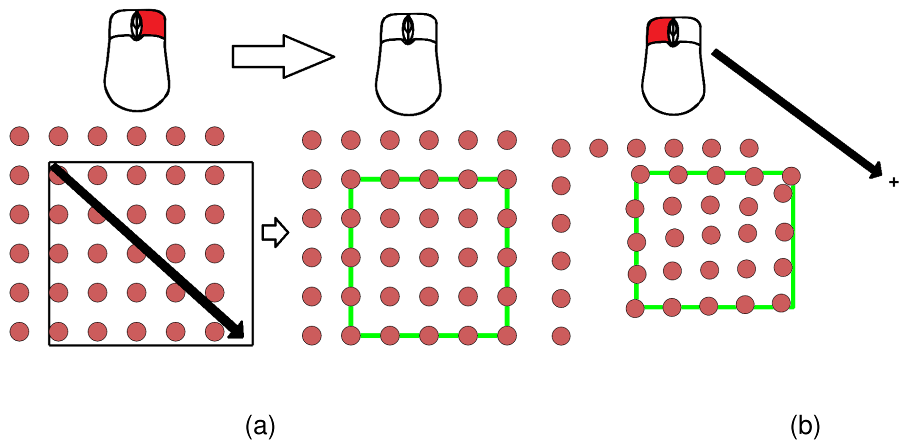


Figure 4.1: The mouse control interface used in the user study. (a) The user could select the characters by right-clicking and dragging across them in the scene. (b) The characters could then be given a goal point by left-clicking in any open space.

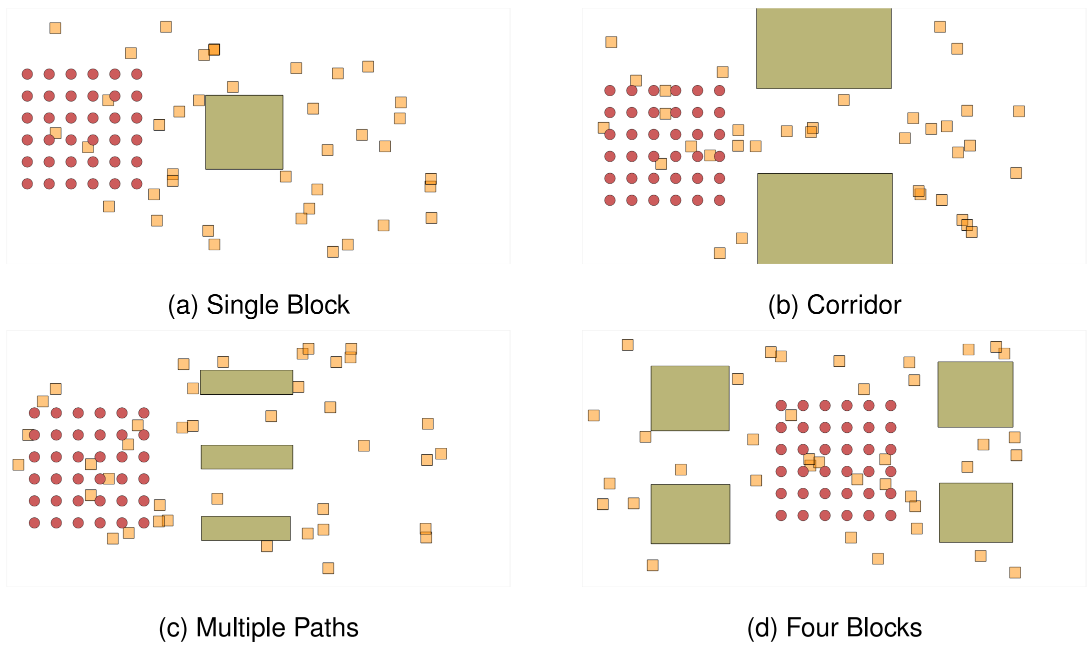


Figure 4.2: The initial setup for environments used in the user study. The positions of the items to collect (small orange blocks) were randomised for each trial.

Participants were given some practice time to get comfortable with using the mouse scheme and our proposed multitouch control scheme. In general, participants spent 1-2 minutes practicing with each control scheme. Once happy with each scheme, participants were asked to carry out a number of tasks to test them. In each task users were presented with one of 4 different environments (Figure 4.2). Each of the environments contained a set of obstacles as well as a number of collectible items. Users were instructed to guide a set of characters to collect the items in the environment in as little time as possible. The position of the collectible items in the scene was randomised at the start of each task to prevent any experimenter bias from their placement. Each environment was presented twice to the user: once for each control scheme, and the order in which users tested the control schemes was switched to prevent any bias from task experience.

4.3 Results

4.3.1 Task Completion

In most scenarios, the multitouch controller enables more efficient collection of items by allowing a user to move the crowd and manipulate its shape simultaneously. A comparison of the times taken to complete each task with the different controllers is shown in Figure 4.3. It can be seen that the time for task completion is reduced in three of the four scenes when using our multitouch controller. In fact, for the "Single Block", "Corridor", and "Multiple Paths" environments (Figure 4.2(a)-(c)) the multitouch controller shows a 35%, 16%, and 20% decrease in the median completion time respectively, compared to the mouse controller.

	P-value			
Data	*Single Block*	*Corridor*	*Multiple Paths*	*Four Blocks*
Completion Time	0.0016	0.1571	0.1696	0.0015
Number of Inputs	0.0001	0.0016	0.0103	0.0126
Question Scores	0.6839	0.1777	0.4288	0.7275

Table 4.1: P-values from a two-tailed paired t-test for significance between results collected using the mouse and multitouch control schemes. P-values are shown to 4 decimal places. For all significantly different data ($p < 0.05$) the "Four Blocks" completion time is the only one in favour of the mouse control scheme.

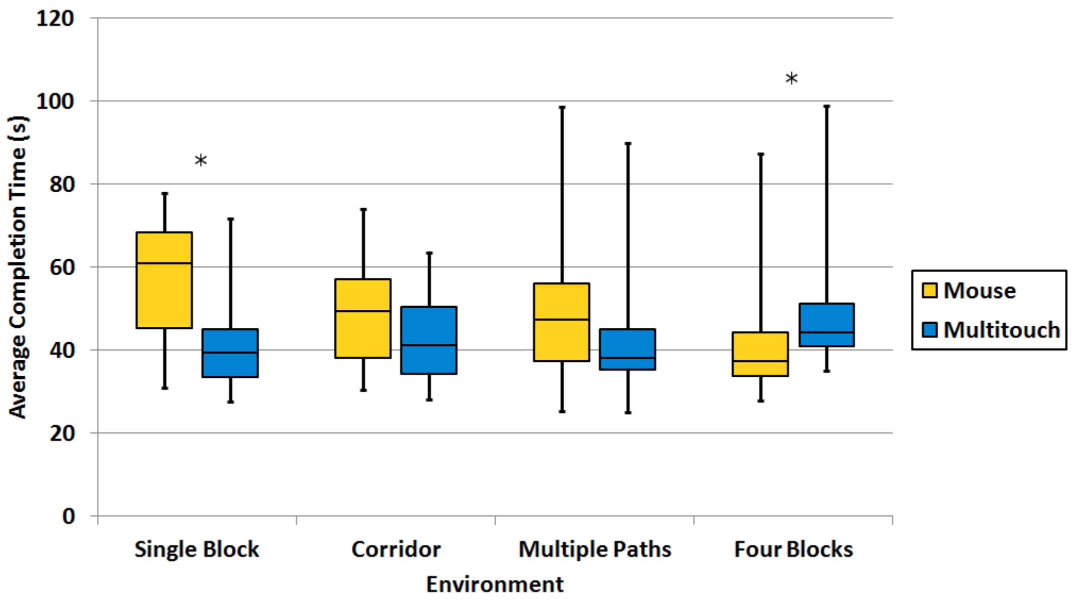


Figure 4.3: Box-whisker plot of item collection task completion time for multitouch and mouse-based controllers in the four different environments shown in Figure 4.2. An asterisk indicates that the data is significantly different (p < 0.05 in paired two-tailed t-test).

We performed a paired two-tailed t-test to check for differences between the multitouch and mouse controllers. The generated p-values can be seen in table 4.1. The task completion times for both the "Single Block" and "Four Blocks" scenario are shown to be significantly different ($p < 0.05$) with the former favouring the multitouch controller and the latter the mouse controller.

4.3.2 Required User Input

Figure 4.4 shows a box-whisker plot for the counts of inputs provided by a user in completion of the box collection tasks. For the mouse-based controller, a single input constitutes all events received from the mouse between a button-down and corresponding button-up event (i.e. one input is either a right-click and drag to select characters or a left-click to set the selected agents' goal). In terms of our multitouch controller a single input is regarded as the events received from the moment the first touch-down signal is registered until the last finger is removed from the multitouch device (this therefore represents a single *interaction period* between the user and the screen).

From Figure 4.4 we can see that the multitouch controller requires far fewer unique inputs for completion of all of the tasks as compared to the use of the mouse-based controller. There are still some cases in which the multitouch controller required a larger number of inputs such as in the “Multiple Paths” scenario but overall the multitouch showed a 64%, 57%, 53%, and 54% decrease in the average number of required inputs for the “Single Block”, “Corridor”, “Multiple Paths”, and “Four Blocks” environments respectively. With a paired two-tailed t-test the p-values indicate that in all scenarios this reduction in number of inputs is significant ($p < 0.05$, see table 4.1 for the values).

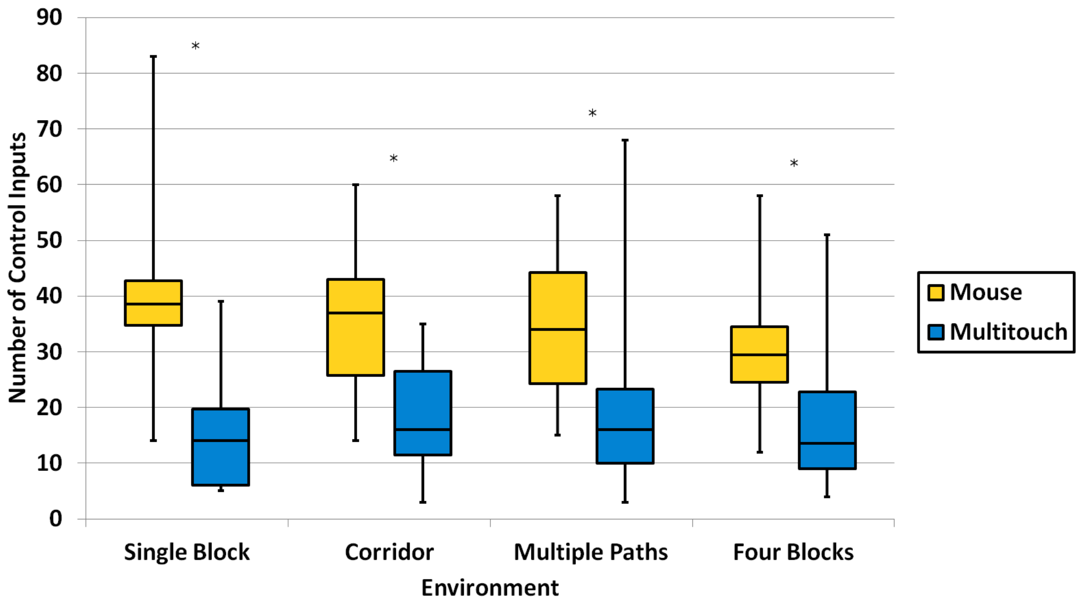


Figure 4.4: Box-whisker plot of the number of control inputs provided by the user for completion of an item collection task in four different environments using the multitouch and mouse-based controllers. An asterisk indicates that the data is significantly different (p < 0.05 in paired two-tailed t-test).

4.3.3 User Feedback

In addition to completing the above tasks, participants were also asked a set of questions with regards to their experience of each control mechanism. Figure 4.5 shows the average scores given by participants for each of the questions outlined in Figure 4.6. In all cases both the mouse and the multitouch control scheme averaged a score of between 3 and 4, with the multitouch control showing a better score in the question concerning participants’ overall view of the control scheme. The slightly better score for the mouse control scheme in questions 1-3 may in part be due to the familiarity of the participants with using a mouse device. A number of the participants commented that their experience of using a mouse device in real-time strategy as well as other games may have meant that they favoured this device implicitly, through what may be referred to as a “mouse prior”. The multitouch device, despite being comparably less common than the mouse, still showed strong scores in response to the questions and the differences between the scores for the multitouch and mouse controllers was not shown to be significant (p-values > 0.05, see table 4.1). This suggests that such devices are appealing as a method for interactive crowd control.

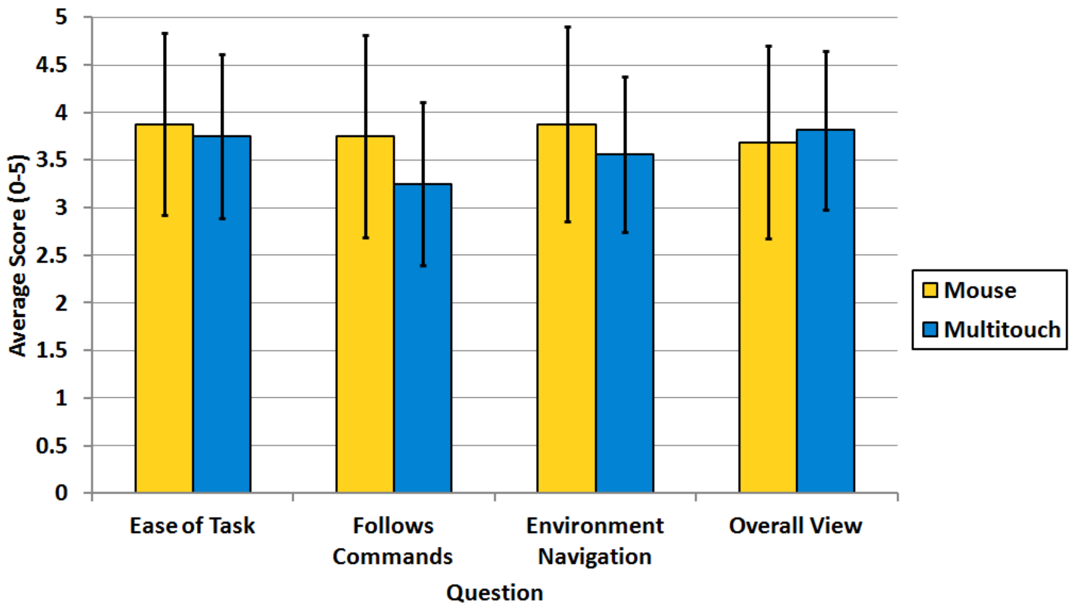


Figure 4.5: Average scores and their standard deviations for multitouch and mouse-based controllers given in response to user study questions (see Figure 4.6 for the full questions).

Question 1: *Please rate how you found it to complete the tasks using this control mechanism. (0 = Very Hard, 5 = Very Easy)*

Question 2: *Please rate how well you felt the characters followed your commands. (0 = Not At All, 5 = Very Much So)*

Question 3: *Please rate how you found it to navigate your characters in the various environments using this control mechanism. (0 = Very Hard, 5 = Very Easy)*

Question 4: *Please rate your overall experience of this control mechanism. (0 = Very Poor, 5 = Very Good)*

Figure 4.6: The questions presented to each user after completion of the tasks using either the mouse or the multitouch controller.

4.3.4 Use of Multitouch

To assess the tendency for a user to employ multitouch control we took a count of the occurrences of different numbers of touch signals used in an input. These can be seen in Figure 4.7. The skew shown in the counts towards a single touch input may also be indicative of the aforementioned "mouse prior", where users are more inclined to use a single finger for interaction as this is what they are used to. Despite this, Figure 4.7 shows that users did incorporate multitouch inputs, including some using as many as 6 touch signals. The greater propensity for 2-touch gestures suggests that users were more comfortable using fewer fingers and it is likely that they used the line control gesture presented in Section 3.3.2. Equally, an increase in the frequency of 4-touch inputs compared to others indicates that users made use of the area control gesture for expanding or contracting the formation whilst moving the crowd around the scene.

The greater amount of open space in the "Single Block" environment allowed users to take advantage of the multitouch controller's simultaneous movement and shape control capability. In a number of cases the users were able to expand and contract the group formation whilst guiding the agents around the scene resulting in a large improvement in task completion time. In the "Multiple Paths" scene, participants utilised the mesh interaction with the environment to divide the crowd into several pathways at once. This allowed the user to cover much of the environment with minimal gestures and complete the task more efficiently when compared to using the mouse.

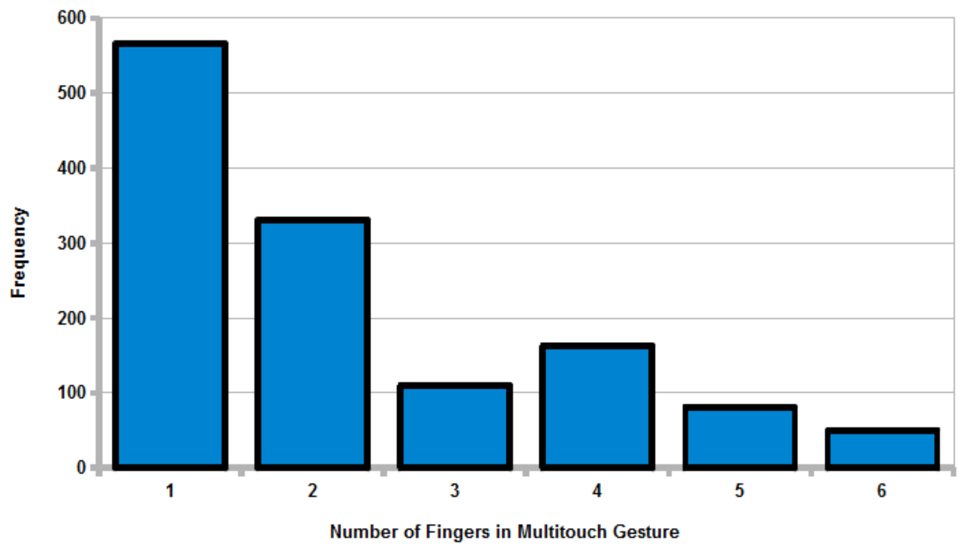


Figure 4.7: Count of the number of fingers used in users' control signals for manipulating the deformable mesh.

4.4 Discussion

The results of our user study highlight the benefits of using a multitouch device for controlling crowd motion. In particular, users are able to use our control scheme to guide a crowd of virtual agents to complete a task in a shorter time and using fewer inputs as compared to a standard mouse-based controller. On top of this, participants answered positively when questioned on the usability, intuitive nature, and general experience of the multitouch controller, providing similar scores to those given for the mouse-based controller. Some users highlighted the fact that the common usage of a mouse device may have biased their answers in its favour, suggesting that a multitouch-based controller is still a viable option for controlling crowd simulations.

4.4.1 User Compatibility

In Section 3.7.4 we highlighted that, because of the use of a two-dimensional mesh, the current multitouch control scheme imposes certain limits on the type of crowd motion that can be produced. The result of this is that the overall realism of the crowd is reduced as the agent's movement conforms to the rigid mesh shape. However, this rigidity enables a user to control a crowd to perform certain tasks more effectively than when using other control schemes. In the current chapter we have shown that such a scheme can be used to significantly improve task completion times in certain scenarios, whilst showing some improvement over the mouse controller in others. The multitouch scheme conferred a particular advantage in environments where the user is able to expand the crowd's formation and control their shape to cover the environment more effectively. Furthermore, in all scenarios the total input count given by the user to complete the task is significantly lower for the multitouch controller. While the current multitouch control scheme is unable to provide fully realistic crowd movement, the results of our user study show that it is able to produce sufficient motion to complete basic tasks and can be controlled to do so with significantly lower amount of user input. As a result, our multitouch control scheme provides a method for faster, simpler control of a group of agents at the cost of reduced realism in their movement.

Further inspection of the participants usage of the multitouch controller shows a large bias towards single or two-touch control. This could be attributed to the greater comfort level when performing these types of control, both in terms of what the participants are already used to (a mouse and other devices are generally controlled by a single finger) or in terms of the physical comfort of applying fingers to the touch screen. A reasonable number of 3 and 4-touch controls present in the data suggests that the first instance is the more likely.

4.4.2 Flexibility

In the “Four Blocks” scene (Figure 4.2(d)) the mouse controller gives a lower average time for task completion. This highlights a limitation of the current approach: the crowd must remain as a cohesive whole. With the central placement of the crowd in the “Four Blocks” scene the best strategy to collect items is to split up the crowd and send them to different corners of the environment simultaneously, something the mouse controller is able to do more effectively than our multitouch controller. In this case, the implicit group cohesion caused by the use of a deformable mesh in our method limits the crowd’s ability to multi-task. A future development of this work would consider approaches for splitting and merging of the crowd and user specification of subgroups. Additionally, alternative methods for shape manipulation through the multitouch device may provide fruitful avenues for such research. In Chapter 6 we present preliminary work towards this goal. In this we consider the association between a user’s touch input configuration and the formation of the crowd to generate new crowd motion based on a novel user input, without the restrictions of mesh-based control.

4.4.3 Other Crowd Simulation Control

We observe that other formation control interfaces, for example sketching-based [Gu & Deng (2011b, 2013)], utilise a control scheme similar to the mouse controller. Such controllers have limited responsiveness due to requiring multiple passes to direct a crowd. This suggests that the multitouch controller would produce lower task completion times compared to such interfaces, particularly given the advantages of the multitouch’s single-pass control shown in the current study. This would be interesting to perform as a follow-up to this study in future work. The user study presented in this chapter shows the advantage of using our method for navigating a crowd through a given environment and suggests that a multitouch device is a promising medium through which to provide user control of virtual crowds.

Chapter 5

Interaction with the Environment

In scenes involving a crowd of characters, the interaction between the individual members of a crowd and their environment is important in conveying realism and creating interesting visual and interactive experiences in computer games and animation. A simulation where characters alter their motion according to their surroundings, such as crawling/crouching under or climbing over objects, is much more akin to real crowd motion than when characters simply avoid or bump into an object with no effect. This is particularly the case when simulating scenes of panic, when a character may wish to traverse an obstacle rather than run around it as this would allow them to escape danger more quickly. Previous crowd simulation systems largely focus on the use of simple running and avoiding motion to guide a set of characters through a virtual environment. In these cases, consideration of the environment manifests as collision avoidance, both at an individual or group level. To enhance the simulation, we wish for agents to show a wider variety of character motions and to be able to interact directly with objects in the environment. In this chapter we present a method for incorporating such interactions into crowd simulation and control.

Gu & Deng (2011a) looked at enhancing the diversity of agent motion in crowds. However, motions that involve a character interacting with the environment around them were not considered. Choi *et al.* (2011) achieved effective path planning involving climbing and crawling actions in cluttered environments for an individual agent whilst Lee *et al.* (2006) generated scenes of multiple characters interacting with a virtual environment. These last two techniques utilised a *motion patch* approach which is able to incorporate real motion data into the scene. However, these approaches do not provide a lot of flexibility in terms of user control and, although there are several characters in the scene, the generated interactions involve only a single character at a time.

Kim *et al.* (2013) incorporated heuristics-based “pushing” and “pulling”-style interactions as constraints in the ORCA framework to recreate physically plausible behaviour in crowd simulation. Their method was able to couple physical forces generated by agents and the environment into a single system, allowing multiple characters to influence the movement of obstacles simultaneously. However, this approach does not consider how agents may interact with the environment in other ways, such as traversal of an object, and the effect this would have on their motion.

In this chapter we present an extension to the work in Chapter 3 to allow user control of characters in complex scenes, where the agents’ motion is directly affected by their interaction with the environment. In Chapter 3, the user-defined formation is maintained under perturbations by assigning the agents’ goal positions via minimisation of their collective movement. Since many interesting scenarios consist of a rich set of actions involving direct character-environment interaction, we wish to consider the effect of such motions on a character’s path planning and therefore their ability to reach a position in the final crowd formation. We suggest the use of patch-based approaches in order to achieve this. Such patches can be used to represent the motion required to interact with a particular part of the environment. Previous work on embedding motion data in virtual scenes involves either placement of specific patches in a regular tiled grid [Yersin *et al.* (2009); Lee *et al.* (2006)], or stitching of irregular shaped patches in a highly constrained fashion [Hyun *et al.* (2013); Shum *et al.* (2008)]. While these methods are effective for producing large scale crowd scenes the relatively inflexible nature of the patches makes them difficult to apply directly to interactive control of a crowd. We wish for the characters to be able to enter and exit a given patch based largely on the directions given by the user. We therefore propose a simplified version of motion data patches that permits flexibility in the simulation whilst still conveying relevant information on character-environment interactions.

Compared to the approach in our previous work in Chapter 3, we have enhanced our system such that the complexity of the environment is considered when assigning final positions in the crowd formation. In particular, our system now takes into account areas of the environment that require characters to conduct special actions such as crawling, jumping, climbing and swimming. This is achieved by applying a modified version of the distance metric based on the Eikonal function [Treuille *et al.* (2006)], instead of using a simple Euclidean distance metric when solving for the characters’ goal positions (Section 5.3). By including information on the impact of the environment on an agent’s path the system can produce realistic and efficient formation control even in very crowded and complex environments, which could have resulted in congestion in our previous system. Furthermore, an additional feedback loop, based on the characters' distance from the user's input formation, has been added to the system from Chapter 3 (Section 5.4.4). The feedback loop is used to adjust the progress of the deformable mesh along the constraints defined by the user. This prevents the user-defined formation from separating from the characters if they become heavily perturbed. As a result, the characters remain closer to the mesh throughout the simulation and the final crowd trajectories are more representative of the user-defined crowd motion. Experimental results show scenes of crowds interacting with static and dynamic obstacles whilst following formation and motion signals defined by a user's input (Section 5.4). Our system is best applied to real-time crowd control applications involving formation changes and environment interactions such as real-time strategy games. Our approach can also be used for interactive animation creation to generate scenes including city-scale crowd flow.

5.1 Contributions

- An environment-aware metric for evaluating the cost of travel for an agent to reach a given goal. The metric accounts for the effect various traversable objects in the environment have on the cost of an agent's path as well as the traditional consideration of the cost to travel through open areas or impassable obstacles. This enables effective path planning to be performed in a wider variety of virtual environments and generates realistic biasing of an agent's movement through areas of more easily traversable terrain.

- An approach for incorporating motion capture data information into the proposed environment-aware cost metric. As such, the cost for the execution of the motion data is incorporated into an agent's path planning.

- A feedback loop between the desired crowd state and an agent's current state in the simulation. The incorporation of this loop creates a better coupling between the user's commands and the actual movement of the crowd in the simulation. This results in better formation tracking and a more appropriate final animation.

5.2 Method Overview

Here we provide a general overview of the methods presented in this chapter. Since the work in this chapter is used to extend the work in Chapter 3, the following outline is given with respect to that provided in Section 3.2.

0. **Preprocessing and Embedding of Motion Data Patches** - Collect, clean up and extract average speed data from motion clips of character-environment interactions. Normalise the motion data patches and apply to scenes using polygon-drawing interface (Section 5.3.4)

1. **User input** - Convert user input to signals for manipulating the crowd formation (see Section 3.3.2).

2. **Mesh Deformation and Environment Interaction** - Deformation of the crowd formation according to user constraints (Section 3.3.1). The motion of the current user-defined crowd formation is adjusted using a feedback signal based on the state of the crowd and the deformable mesh (Section 5.3.5). Further alteration of the crowd formation mesh to account for interaction with the environment (Section 3.4).

3. **Goal Assignment and Individual Character Movement** - Assign agents' goal positions in the final crowd formation using an environment-aware cost metric based on obstacles and embedded motion data patches (Section 5.3) and the mass transport solver. Use the generated cost field to create the paths for agents to reach their goals once they have been assigned.

5.3 Improved Character to Formation Mapping

Given the configuration $\mathbf{F}_c(t)$ for the control mesh, computed as described in Section 3.3 and Section 3.4, we next have to determine the target point on the mesh for each agent. To minimise obstructions between agents in the crowd during transition to the new formation it is necessary to assign an agent a target position based on their current configuration. In Section 3.5 we described how we determine the goal position of each agent by employing a solution to the mass transportation problem with an Euclidean distance metric. Here we discuss how we use a potential field construction to incorporate an environment-aware metric in our mass transport solver (Section 5.3.1). This metric not only accounts for obstacles and other agents in the environment (Section 5.3.2) but also considers the motion data that will be used in the final render of the scene (Section 5.3.3-Section 5.3.4). By doing this we are able to encode motion data information implicitly in an agent's planning. This results in better goal assignment and subsequently more efficient movement of agents to satisfy a user's input formation.

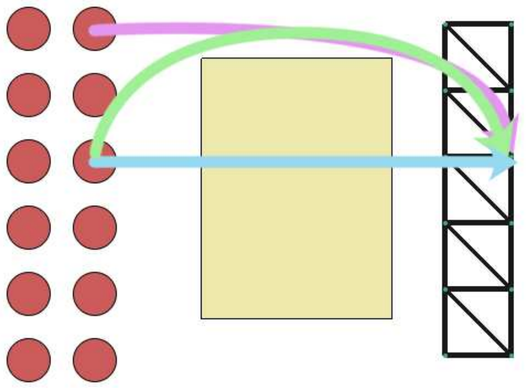


Figure 5.1: Limitations of an Euclidean distance metric for assigning a character's goal position. Assigning formation goal points based on Euclidean distance (blue arrow) fails to consider the true length of the agent's path in the presence of obstacles (green arrow). It is more efficient to assign this goal point to an agent whose true distance to travel is smaller (pink arrow). This can be achieved using a cost metric based on geodesic distance (or equivalent) in the mass transport solver.

5.3.1 Environment-Aware Metric for Goal Assignment

Appropriate assignment of agents to the vertices of the formation mesh, $\mathbf{F}_c(t)$, is achieved by employing a suitable cost metric, $c_{i,j}$, in the mass transport solver. In Chapter 3 a Euclidean distance metric was used to solve the transportation problem for an agent's target location. However, this is not optimal, particularly in environments with large obstacles. Consider a situation where there is an obstacle between the crowd and their target formation as in Figure 5.1. The best solution would have the agents on the outside of the crowd move to locations in the middle of the target formation as they travel a shorter distance and reach these points earlier (Figure 5.1, pink arrow). By using the Euclidean distance metric (Figure 5.1, blue arrow) the cost provided to the mass transport solver does not reflect the route the agent must take to reach the formation (Figure 5.1, green arrow). The true shortest distance that takes into account the obstacles must be used to obtain the best assignment of agent goal positions in the formation. In addition to the actual distance an agent must travel, assignment must also consider how long it may take for an agent to follow a given route. This can be done by taking into account the speed with which an agent can move through given parts of the environment. For example, in Figure 5.2 it may take an agent a longer time to travel across the obstacles which, while traversable, will slow them down. In the next section we describe how we incorporate this information into the mass transport solver and in Section 5.4.3 we show the effects this information has on the agents' path planning.

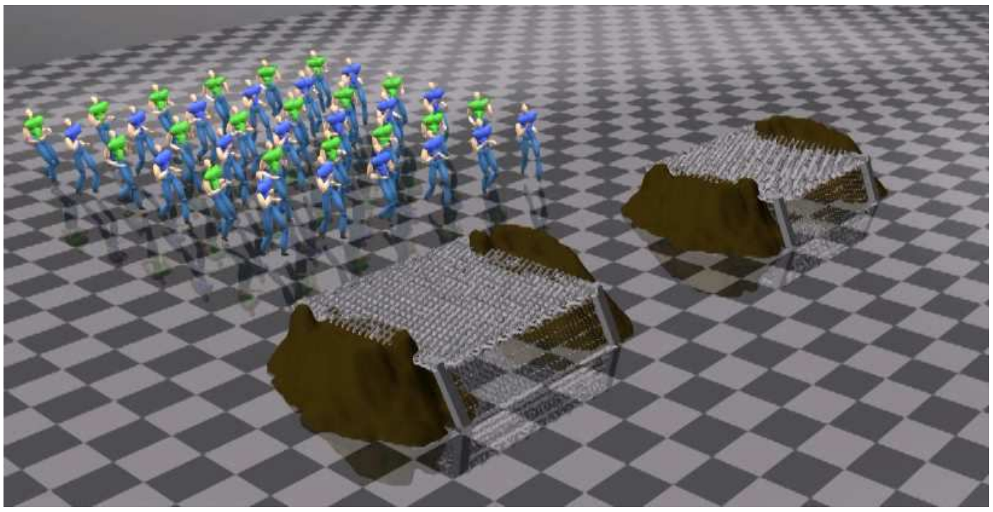


Figure 5.2: Example of a crowd's choice to move through different types of environment. A crowd encounters a set of passable obstacles in the environment. In this case each agent in the crowd must choose which route is quicker: passing through the central pathway or climbing over the obstacles either side of it. When the central pathway is too congested the outer paths over the obstacles become preferable.

5.3.2 Evaluating Cost to the Goal

In continuum-based crowd simulation [Treuille *et al.* (2006)] the cost for an agent to travel to its goal is given by an approximation of the Eikonal equation. This approximation uses a cost metric that accounts for the environment as well as other agents. We construct a potential field to determine the cost to travel to the vertices of the formation mesh, $\mathbf{F}_c(t)$, for a given point in the environment. In Treuille *et al.* (2006) the overall cost for an agent to travel to its destination is provided by a combination of the length of the path to the goal, the time taken, and a discomfort field based on obstacles and other agents in the environment:

$$\underbrace{\alpha \int_P 1 ds}_{\text{Path Length}} + \underbrace{\beta \int_P 1 dt}_{\text{Time}} + \underbrace{\gamma \int_P g dt}_{\text{Discomfort}}, \tag{5.1}$$

where α, β, and γ are weights; g is the value of discomfort at a given point in the environment; and dt and ds indicate that the integral is taken with respect to time or path length respectively. Readers are referred to Treuille *et al.* (2006) for more details on how the individual values for path length and discomfort in Equation (5.1) are calculated. Using the equality $ds = f dt$ where f is the speed of an agent, Equation (5.1) can be rewritten and simplified to

$$\int_P C ds, \text{ where } C \equiv \frac{\alpha f + \beta + \gamma g}{f}. \tag{5.2}$$

By applying Equation (5.2) to a two-dimensional grid of the environment we can compute a *unit cost field* for a given scene. In the current work, this unit cost field represents the evaluation of Equation (5.5) for travelling across the edge of each grid square termed the *cost-to-go*, i.e. the cost to move from the current grid cell to a neighbouring grid cell. To produce the final potential field, ϕ, we employ the same approach as Treuille *et al.* (2006) of using the fast marching method [Tsitsiklis (1995)] to approximate the *Eikonal equation*:

$$\|\nabla\phi(\mathbf{x})\| = C, \tag{5.3}$$

where $\phi(\mathbf{x})$ is the value of the field, ϕ, at a given point $\mathbf{x}$ in the environment. $\|\nabla\phi(\mathbf{x})\|$ represents the gradient of this field at point $\mathbf{x}$ which is equivalent to the cost, C, to travel through this point. The fast marching method is a one-pass, non-iterative algorithm for the numerical solution of Equation (5.3). The algorithm presented in Tsitsiklis (1995) provides a solution in time $O(n \log n)$ where n is the number of grid points. A potential field $\phi_\mathbf{i}$ is constructed for each vertex $\mathbf{v}_i$ in the current user-defined control mesh $\mathbf{F}_c(t)$. In each case, $\phi_\mathbf{i} = 0$ in the cell containing $\mathbf{v}_i$ and everywhere else $\phi_\mathbf{i}$ satisfies Equation (5.3). This means that the total number of cost fields to calculate is determined only by the number of formation target positions in $\mathbf{F}_c(t)$ and is independent of the number of agents in the simulation. Each agent can reuse the cost field associated with a target position to obtain their cost to travel to it. This means that a larger number of agents could be controlled by the same number of target positions with a relatively small increase in the computation time. An example potential field for a single vertex in $\mathbf{F}_c(t)$ can be seen in Figure 5.3. Given the position $\mathbf{x}$ of an agent $a \in \mathbf{A}$, where $\mathbf{A}$ is the set of all agents, we can retrieve the cost for the agent to travel to each vertex, $\mathbf{v}_i$, in $\mathbf{F}_c(t)$, by taking the value at that location in the appropriate potential field:

$$C_{a_\mathbf{x} \to \mathbf{v}_i} = \phi_\mathbf{i}(a_\mathbf{x}), \tag{5.4}$$

where $C_{a_\mathbf{x} \to \mathbf{v}_i}$ is the cost for agent a at position $\mathbf{x}$ to travel to $\mathbf{v}_i$.

Due to the discrete nature of the two-dimensional grid we use bilinear interpolation on the values associated with the edges of the grid square containing $a_\mathbf{x}$ to achieve a more accurate reading from ϕ_i. These values can be passed into our mass transport solver in order to assign agents an appropriate goal position in the final formation. Once a point in the mesh is assigned to an agent their route to the location is computed using gradient descent on the field associated with their assigned goal vertex. The characters then move to their corresponding target locations with a simple PD controller. A maximum speed is defined to avoid unnatural fast movement.

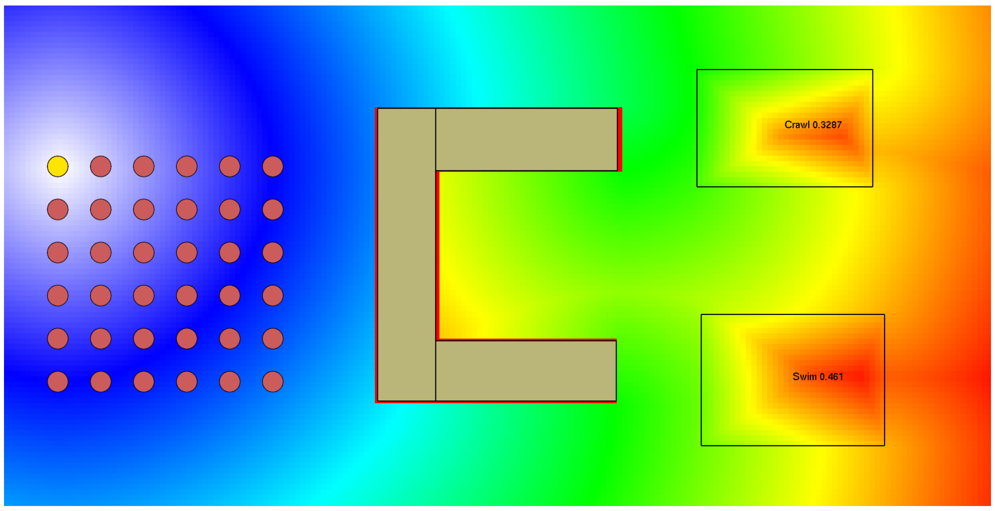


Figure 5.3: An example of a potential field produced for a single vertex in the user-defined control mesh in an environment containing obstacles and motion data patches. Red indicates a high cost to travel and white indicates a low cost. The field shown is with respect to the formation position of the top-left agent (highlighted in yellow).

5.3.3 Representing Environment Interactions in the Cost Metric

In the majority of previous crowd simulation research, environments consist solely of traversable “free” space or impassable obstacles. However, in a number of real-life environments there are certain objects that, while traversable, will affect a person’s speed of travel across them. Examples of these include low areas through which a person must crawl or fences/walls over which people must jump or climb. In this section, we describe how we apply such an idea to agent planning to produce appropriate motion in various environments.

The fast marching method [Tsitsiklis (1995)], used to compute the cost field for the environment-aware metric (Section 5.3.2), is an approach that can be used to track a moving boundary expanding outwards from a source point. The concept of a *speed field* is used to define the rate at which the front of the moving boundary propagates.

In this way, low values in the speed field can be used to represent obstacles, whilst high values can signify open areas in a scene. We can therefore think of this value as representing the speed at which an agent can travel through a given point, $\mathbf{x}$, in the environment. For impassable areas the travel speed can be considered to be zero (creating infinitely high cost values) whilst for open areas, in which an agent can move freely, the travel speed can be considered to have a value of 1, allowing an agent to move at their desired speed. Areas that do not permit an agent to move freely but are still traversable can be assigned an intermediate value for travel speed. It is also possible that certain areas of an environment, such as moving walkways, allow an agent to travel faster than their normal running speed. We incorporate this idea into the current framework by rewriting C in Equation (5.2) to be:

$$C \equiv \frac{\alpha f + \beta + \gamma g}{f \times S(\mathbf{x})}, \tag{5.5}$$

where $S(\mathbf{x})$ represents the speed at which an agent can travel through a given point, $\mathbf{x}$, in the environment. By formulating our cost function in this way, areas of the environment that do not affect an agent's motion can be assigned a travel speed of 1, essentially having no effect on the original cost metric from Treuille *et al.* (2006). Areas with travel speeds less than 1, that therefore slow the movement of the agents, induce a high value for C. Objects in the environment that actually decrease the time to move through a given area are assigned a travel speed greater than 1, making the value computed for C much smaller. This can occur when an agent performs an interaction with an object in the environment that actually speeds up the motion of the agent, for example when travelling on an escalator or conveyor belt.

5.3.4 Embedding Motion Data in the Environment

By considering environment travel speed in the cost function, we can easily incorporate a variety of different motions into agent planning by extracting velocity information from motion data in a preprocessing stage and later embedding this information into the crowd scene. Our method only relies on information from the 2D trajectory of a given motion, so it works equally well with both video tracking or full-body motion capture data.

Preprocessing Motion Data: In the current work we use trajectories extracted from full-body motions of characters traversing various environmental obstacles. Given a trajectory we use finite (forward) differencing to calculate the velocity, $\mathbf{x}'$, of the recorded motion at a given time t in the data:

$$\mathbf{x}'(t) = \frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t}, \tag{5.6}$$

where $\mathbf{x}(t)$ is the two-dimensional position of the centre (root position) of the character at the time t and Δt is the timestep between consecutive frames in the motion data. The magnitude of this set of velocities represents the per-frame speed at which the character is travelling in the motion data example. To calculate a speed value, S, to be used in Equation (5.5) we take the average magnitude of the character's velocity across the n frames of the motion data:

$$S = \frac{\sum_{i=1}^{n} \|\mathbf{x}_i'\|}{n}. \tag{5.7}$$

Given the entire set of motion data clips, M, to be used in the crowd scene, we calculate the values of S for each individual clip $m \in M$ using Equation (5.7). We define the normal travelling speed of an agent, $S(O)$, as their speed when unaffected by objects in the environment i.e. they are running in open space. We normalise each motion clip's average speed values by adjusting them according to $S(O)$:

$$\hat{S}_m = S_m / S(O) \quad \forall_{m \in M}, \tag{5.8}$$

where S_m represents the original average speed of the motion data clip m and $\hat{S}_m$ is the adjusted value of motion data clip m relative to movement in the open space. In our experiments $S(O)$ is the speed associated with a cycle of running motion data and, when applied to Equation (5.8), it is in turn normalised with itself. As a result, Equation (5.8) gives a value of 1 to the motion used for the character moving in open space and a value relative to this motion for all other data. Table 5.1 shows the values extracted from the motion data used in our experiments. Note that, with this method, at no point is a value of zero assigned to a motion data example as this is used to represent impassable obstacles. This would only occur if the motion data clip being processed showed no movement at all.

Adding Data to the Scene: This information can be applied to the scene by creating "patches". These patches are represented as 2D polygons in the scene and contain

Motion Type	Average Travel Speed
Run	1.0
Crawl	0.33
Duck	0.30
Swim	0.46
Jump	1.24
Climb Up	0.50
Climb Down	0.48

Table 5.1: The average speeds extracted from motion data used in the experiments

the motion data file name, ID, and average speed of the motion. Entrance to a patch is detected when the agent's current motion ID switches from the ID for the running data, used in the open environment, to the ID given by the patch. When an agent enters a patch, the motion data ID determines the motion for the agent to carry out and the average travel speed determines the speed of the agent as it passes through the patch. The average travel speed also determines the speed field value in any grid cells that contain any part of the patch, so that it can be used in Equation (5.5) to compute the unit cost field during the planning phase. As described in section 5.3.2, examples of these patches, and how they affect the overall cost field, can be seen as the two transparent blocks on the right side of Figure 5.3. It can be seen that the value of the cost field increases within patches that slow the movement of an agent, in this case a crawling and a swimming motion. For cyclic motions such as crawling, ducking, and swimming, patches can be of arbitrary shape and size, however, for single-step motions such as jumping and climbing, the length of the patch is constrained by the length of the motion data's trajectory. This prevents these actions from inappropriately executing multiple times in the final render. Additionally, for the crawling, ducking, and swimming motions, an appropriate transition motion is applied when it is detected that the agent enters or leaves the patch.

5.3.5 Coupling of User Input and Crowd Motion

In the system in Chapter 3, the movement of the deformable mesh is governed by the constrained vertices' progress along a user's input trajectories. To enable the agents to follow the deforming mesh, these constraints move at the same speed as an agent. If the agents remain unperturbed then they are able to keep up with the mesh and follow the crowd motion defined by the user. If however the motion of the agent is hindered, either through collision with an obstacle or slowing down as they pass through a motion data patch, then the coupling between the mesh and the crowd is decreased and the final crowd motion does not closely fit the user's input. In Section 3.6.4 we discussed how, in interactive applications with high chance of collisions, it is important for the agents to not be fixed to the mesh movement so as to prevent conflict when following a user's input in complex environments. In light of this, we employ a feedback term to the movement of the deformable mesh that accounts for the current state of the crowd. The key idea is to prevent the mesh from continuing to follow the constraints when the crowd is not tracking it well i.e. when the crowd is a significant distance from the position of the mesh. Equation (3.1) defines the movement of the control points used to update $\mathbf{F}_c$ according to the user's input. In order to adjust this movement we add an additional feedback signal to our control point update, Equation (3.1), that adjusts the velocity of the control point according to the average distance of agents from their target points on the formation:

$$\mathbf{c}_i\left(t+\Delta t\right) = \frac{\mathbf{p}_{i,j}-\mathbf{c}_i\left(t\right)}{\|\mathbf{p}_{i,j}-\mathbf{c}_i\left(t\right)\|}\left(s-\min\left(s,e^{D(\mathbf{F}_c(t),\mathbf{A}(t))}\right)\right)\Delta t, \tag{5.9}$$

where s is the speed of an agent, and $D\left(\mathbf{F},\mathbf{A}\right)$ is based on the distance between a set of agents and their target formation:

$$D\left(\mathbf{F},\mathbf{A}\right) = \frac{\sum_{i=1}^{n}\|\mathbf{v}_i - a_{i,\mathbf{x}}\|}{r}, \tag{5.10}$$

where n is the total number of agents in $\mathbf{A}$, r is the radius of an agent, and $a_{i,\mathbf{x}}$ and $\mathbf{v}_i$ are the position of the ith agent and its corresponding goal point in the formation $\mathbf{F}$ respectively. Note that if the agents in the crowd are all at their respective goals in the formation then Equation (5.9) reduces to Equation (3.1) and no feedback signal is applied. We use an exponential function for the feedback signal, with the distance normalised by the radius of an agent. This is so that the signal is quite weak when the agents are tracking the formation well but quickly becomes stronger as the agents are perturbed from their goal positions. We apply the min operator with the agents' speed so that the feedback signal is never negative and the formation does not move backwards. This has the effect that when the crowd is not tracking the formation well the mesh appears to stop and "wait" for the agents to catch up with it.

5.4 Experimental Results

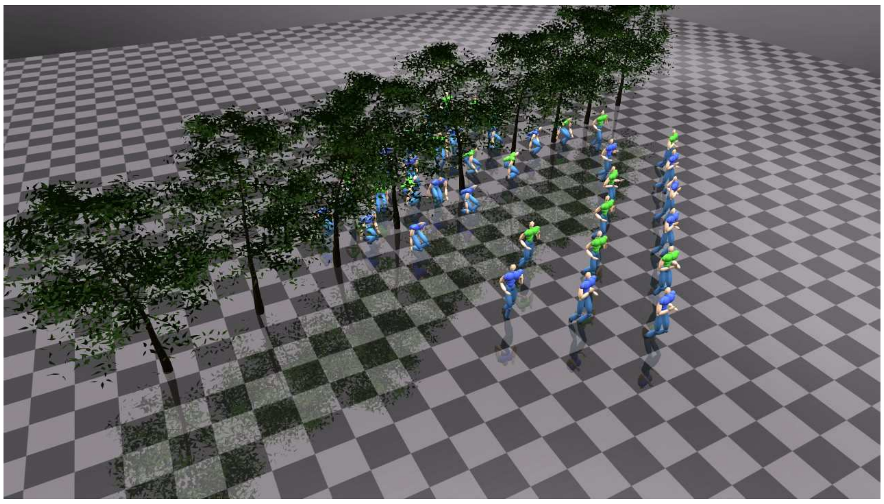
We have produced scenes showing a group of characters passing through different static environments including corridors, woodlands, an obstacle course and a dynamic environment with moving cars. Each scene contains open space and obstacles as well as various objects that the characters can traverse in order to reach their goal. The formation of the characters is manipulated in some of the examples such that they can pass through narrow pathways or interact directly with certain parts of the environment. All the examples were produced starting from a uniform rectangular formation with 36 characters, except the obstacle course example, which consists of 100 characters.

5.4.1 Handling Motion Data Patches

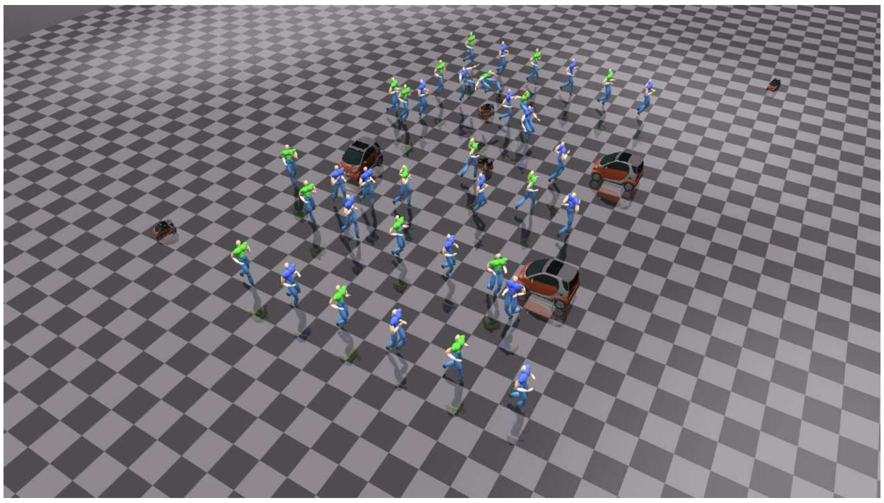
Here we show how characters in the crowd are affected by motion data patches in different environments and how the crowd can handle an arbitrary number of static or dynamic patches. Our first example involves a crowd moving through dense woodland, incorporating several trees modelled by small obstacles. The branches of the trees overlap with one another and as a result, characters must duck to pass through (see Figure 5.4(a)). This motion is represented in the scene by applying appropriately sized motion data patches around the base of each tree. These patches contain the average travel speed from the ducking motion data used in the final render of the scene. The characters are able to pass through the small gaps between each tree while exhibiting a slower travel speed resembling the required ducking motion.

In the next example, the characters are controlled to pass through an environment containing multiple dynamic obstacles (cars) and dynamic patches (mice). When the characters approach the cars they automatically avoid them according to the user control and the potential field produced by the cars (see Figure 5.4(b)). We can tune the strength of the potential field to adjust the distance at which a character starts to avoid the obstacles. Furthermore, in the simulation the characters react to the mice with a jump motion when they pass through the crowd. This reaction is produced by the movement of the dynamic patches and the effect on the character's speed when they interact.

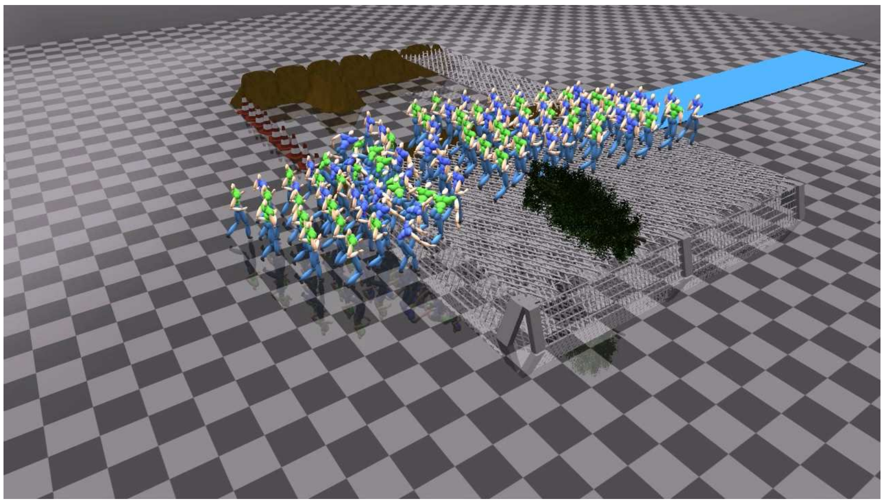
We also produce an example where a larger crowd passes through an obstacle course with narrow corridors, walls to climb/jump over, netting to crawl under, and a pool to swim through (see Figure 5.4(c)). This scene contains different motion data patches that allow the characters to interact appropriately with the environment. Even in such a complex condition, the characters have no difficulty moving through the area.



(a)



(b)



(c)

Figure 5.4: A crowd (a) moving through dense woodland, (b) avoiding large and small moving cars, and (c) passing through a complex obstacle course area. The crowd is able to interact with many arbitrarily placed objects in the environment by responding to motion data patches in the scene.

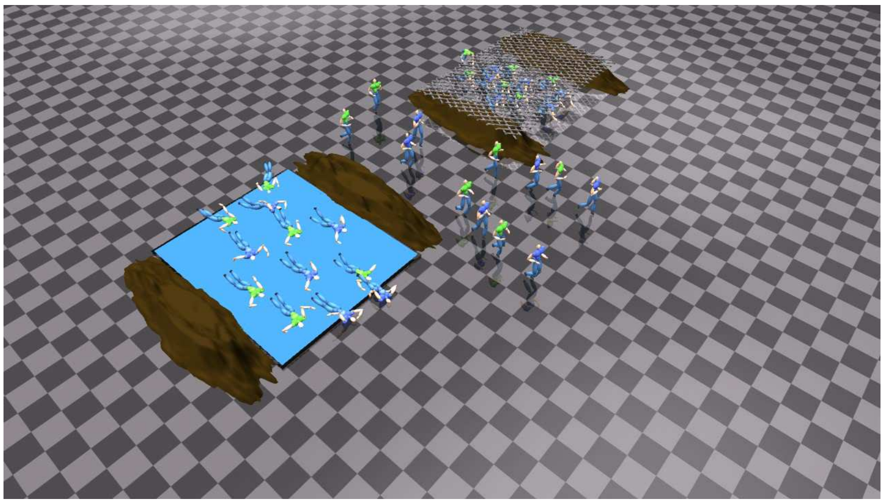
5.4.2 Defining Crowd Trajectories

In these examples we show how a user is able to control the overall motion of the crowd while the low-level interactions of the crowd are handled by the system. We first show an example in which characters pass through an environment containing three separate pathways. Each pathway contains a different kind of terrain, from top-to-bottom: a net to crawl under, open space to run through, and a pool to swim through. A snapshot can be seen in Figure 5.5(a). The user is able to specify the subsets of the crowd that pass through each individual pathway using a basic multi-touch gesture. The movement of individual characters is not defined explicitly by the user, instead, based on the formation defined, the characters fit into the pathways automatically. In each pathway the characters move appropriately according to the type of terrain they are passing through.

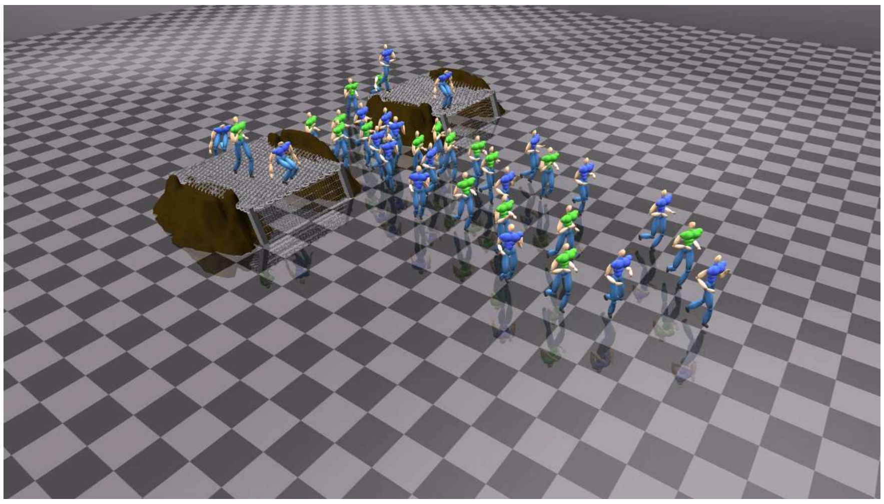
In addition to this, we show a similar example in which characters have multiple paths through which they can travel (see Figure 5.5(b)). This time however the pathways are narrower than before. The central pathway contains open space and the two pathways either side have objects that the characters must climb over. When the user performs a gesture to move the entire crowd through the central pathway it becomes too congested. As a result, the characters choose the slower side pathways rather than waiting for the central pathway to clear. The side pathways have become a better choice for following the formation specified by the user. This example shows the ability for the system to adapt the character's trajectories to best follow the user's commands.

5.4.3 Choosing an Appropriate Path

In this section we further demonstrate the effects of incorporating information on character-environment interactions into the planning and movement of the crowd. Figure 5.6 shows a situation where a crowd must reach a goal formation on the other side of the scene with an obstacle in between (beige box). The agents have a choice of two routes to the goal formation, one of which contains a motion data patch that will slow the movement of the characters. If this effect is not considered in the agents' planning (i.e. the cost is evaluated using Equation (5.2)) then the agents split into two separate groups moving via the upper and lower routes (Figure 5.6(a)). If however the travel speed for the motion data patch is considered in the cost (Equation (5.5)) then all of the agents move via the lower path (Figure 5.6(d)). As the simulation progresses we can see that not considering the effect of the character-environment interactions causes the group travelling via the open lower route to move ahead of the group following the upper route (Figure 5.6(b)) and to reach their destination well before (Figure 5.6(c)). In the second instance, by avoiding the slower upper route the whole crowd can reach the goal formation at the same time (Figure 5.6(f)).



(a)



(b)

Figure 5.5: A crowd (a) moving through several pathways containing various terrain, and (b) choosing slower pathways in response to congestion. The crowd plans an optimal trajectory to follow the instructions of the user whilst also considering the environment.

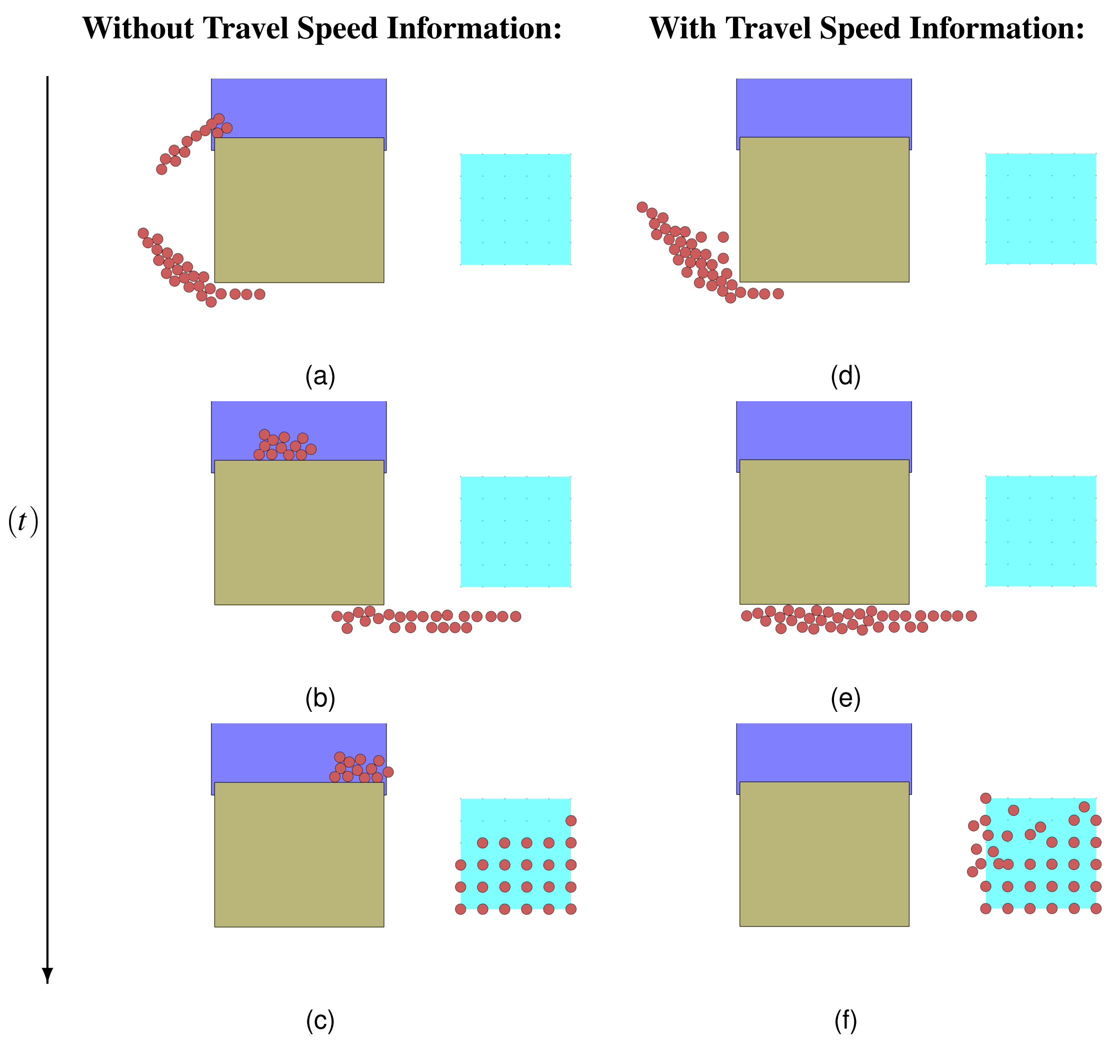

Without Travel Speed Information:
With Travel Speed Information:
(t)
(a)
(b)
(c)
(d)
(e)
(f)

Figure 5.6: A crowd must reach a goal formation (light blue square) on the far side of the environment where one route contains an obstacle to crawl under (represented by the purple area indicating a motion data patch). (a)-(c) The crowd moves without considering the slowing of their movement by the obstacle causing them to split into two groups, with one reaching the goal well before the other. When the effect of the obstacle is considered (d)-(f) the agents choose the more efficient lower route allowing them to reach the desired formation quickly and simultaneously.

5.4.4 Formation Tracking

Here we show the benefit of adding a feedback signal to the update of the control points that deform the formation mesh (Equation (5.9)) based on the current state of the target formation and the crowd. In Figure 5.7(a) the user provides a control signal to move the crowd through a gap between a set of obstacles. This gap contains a motion data patch representing a crawl motion that will slow down the movement of the agents. Without the feedback signal, as the agents are moving through this crawling area, the formation mesh progresses beyond the obstacles and into the open area (Figure 5.7(b)). Even though the user has specified that the crowd should be thin while passing through the gap, without the feedback signal the agents spread out more than they should. Furthermore, the user has specified that the crowd should smoothly return to their original square configuration after passing through the obstacles. In the absence of feedback the deformable mesh has reached the end of the user's input before the crowd has moved out from between the obstacles. This results in the agents moving toward the formation mesh and then changing direction quite abruptly and the final trajectories reflect this (Figure 5.7(c)).

By incorporating a feedback term into the update of the mesh's movement the position of the target formation and the crowd shows much better coupling. In Figure 5.7(d) it can be seen that the mesh remains much closer to the position of the crowd even though they have been slowed down by the crawling motion patch. This allows the crowd to replicate the user's signal for the crowd to be thin as they move through the obstacles. In turn, the final trajectories of the crowd follow the signal from the user to remain thin and then smoothly transition back to the square on the other side of the obstacles (Figure 5.7(e)).

5.4.5 Computational Costs and 3D Rendering

The experiments are run on one core of a Core i7 2.67GHz CPU with 1GB of memory. For the multi-touch input we used a G4 multitouch overlay from PQ labs, attached to a 24" Acer S240HL LCD monitor. The computation of the 2D trajectories including the deforming of the control mesh, reshaping it through its interaction with the environment, computing of the character destination by the mass transport solver, and updating their positions are all done in real-time at a rate of $\sim$32 frames per second. We found that with the current unoptimised implementation, framerates reduced to around 8-10fps at a crowd size of $>$160 characters which still allows for interactive control of the crowd.

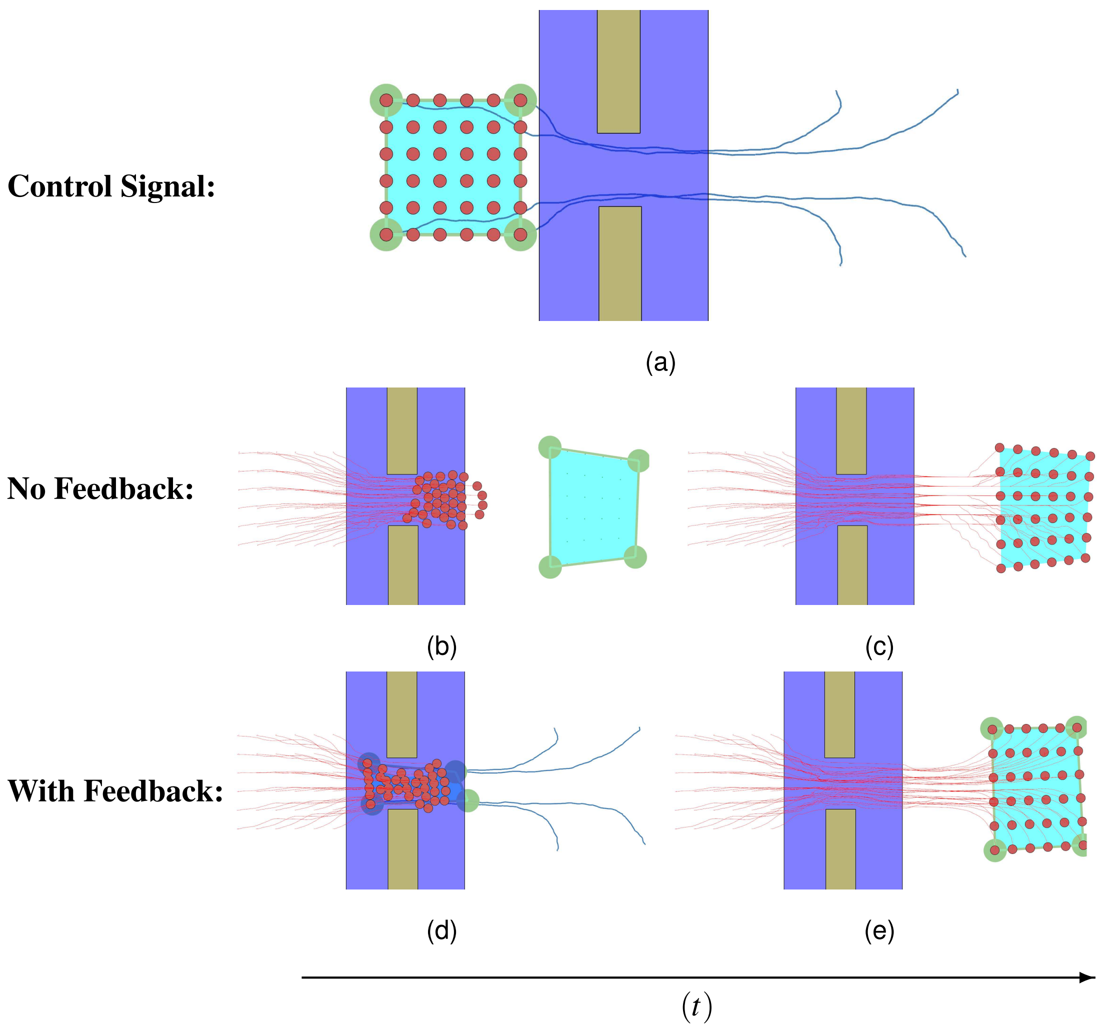


Figure 5.7: (a) The input signal provided by the user. Without a feedback signal the crowd becomes easily separated from the motion of the user-defined deformabe mesh, (b), and the resulting crowd trajectories (red lines), do not reflect the smoothness of the user control, (c). By adding the feedback signal there is better coupling between the crowd and the movement of the deformable formation mesh,(d), generating agent trajectories that better fit the user's input, (e).

5.5 Discussion

5.5.1 Character-Environment Interactions

In this chapter we have presented a method for including character-environment interactions into the multitouch crowd formation control framework presented in Chapter 3. This method provides the system with more information on the effects that objects in the environment can have on an agent's movement speed. This allows our planner to assign character goals in a user-defined formation based on an appropriate approximation of each character's ability to reach their goal. This leads to a better assignment of goals which in turn results in more effective user-defined formation tracking.

The inclusion of a feedback signal based on the state of the agents and their target formation creates a better coupling between the crowd and the user-defined motion when agents are undergoing the interactions with the environment. By employing a feedback signal to achieve this coupling, instead of fixing agents' positions in the target formation, we allow our system to keep the flexibility provided by characters being able to switch positions in the formation. The combination of the feedback signal and the use of the mass transport solver results in greater adherence of the characters' trajectories to a user's input signal and better resolution of any perturbations in the crowd's motion.

5.5.2 Scalability

For the environment-aware cost metric to be incorporated into the mass transport solver a cost field must be created for each goal position in the target formation. This significantly increases the time to compute agent goal assignment compared to using a basic Euclidean distance metric, particularly since we wish to keep this information up to date with regards to the state of the environment.

Alternative representations of the environment, including roadmaps and navigation graphs, could be used instead of our grid-based cost field to only compute a cost for areas of the environment rather than individual grid cells. However, such representations would only be able to incorporate the information from our motion data patches in a coarse fashion. For example, the travel speed from a motion data patch would only affect the cost to travel along the edge of a graph or roadmap that is associated with the area in which it is placed. This means that graphs or roadmaps could not as accurately capture the effect of smaller motion data patches such as those used in our experiments

(Section 5.4.1). Furthermore, the costs associated with roadmaps or graphs are more discrete than from a grid-based cost field. As a result, multiple goal positions are more likely to be judged to have the same cost for an agent to reach. This can cause unnecessary switching in the assignment of agents' goal positions when using these values in the mass transport solver. Decreasing the resolution of the grid is a reasonable compromise between accuracy and speed of computation. Fewer grid squares improves the computation time of the Fast Marching Method whilst the values taken from a grid square can still be interpolated using bilinear interpolation for a more accurate result for an individual agent.

In the current framework there exists one goal position for every agent in the crowd. With small numbers of agents, computation of the unit cost field and approximation of the Eikonal equation by the Fast Marching Method can be performed on the fly. However, as the number of agents increases this approach can become computationally prohibitive. A simple solution can be to only update a subset of the cost fields at each timestep. This will reduce the computational load but mean there is no guarantee that all of the cost fields are representative of the current state of the environment. Since we reassign agents' goal positions at each frame this can have knock-on effects in the planning stage of our system and ultimately the tracking of the user-defined formation. Alternatively, a reduced number of goal positions compared to agents can be used, as suggested in Section 3.7.

5.5.3 Environment-Aware Cost Metric

The environment-aware cost metric presented in Equation (5.5) is able to account for the travel speed of an agent passing through a given point in the environment. However, under the current framework the travel speed value is restricted to being positive. This value therefore cannot consider objects that might have a net negative effect on the agents motion i.e. objects that don't just stop a character but also force them backwards. This is the kind of effect that a moving obstacle will have on the agent. Without this information, an agent will not actively move away from any object that has this effect, ultimately hindering their movement and their ability to track the user's input. Furthermore, the travel speed value does not consider any direction of travel that the object may impose upon an agent. This kind of information is important when considering objects such as ticket gates that only allow a person to pass through in one direction.

5.5.4 Motion Data Patches

The motion data patches presented in this chapter are an effective way to add real motion capture data to the planning of characters in the crowd. Not only do they affect the trajectories and motion of the crowd according to real character-environment interactions but they are a straightforward way to tell which motion data clip should be used in the final render of the scene. Overall, using the average speed of the motion data provides values that implement the effects of executing such actions on both the planning and the eventual movement of the characters in simulation. However, the average speed may not be reflective of the true character-environment interaction. This is true of the non-cyclic motions, such as jumping, that have variable speeds throughout the course of the motion. Applying the speed values across all frames of the motion data would be more representative though this makes the use of such data less flexible and therefore less appropriate in our interactive system. In this work we have presented actions that either passively or actively interact with the environment (such as running and avoiding, or crawling and climbing respectively). We would like to extend this approach by considering motion data involving character-character interactions. A modified version of the patch-based approach could enable us to simulate scenes involving dynamic interactions between characters, such as two armies fighting [Shum *et al.* (2008)].

5.6 Summary

Interactions between characters and their environment play an important part in conveying realistic and immersive experiences in animation and interactive applications. In a similar way to previous research, [Yersin *et al.* (2009); Lee *et al.* (2006); Hyun *et al.* (2013); Shum *et al.* (2008)], we have enhanced the motion of characters in our simulations by embedding motion data into the environment in the form of patches. Whilst previous patch-based approaches require strict constraints to be satisfied, our patches are more flexible, allowing a user to have greater control over the motion and formation of the crowd and still have influence over character-environment interactions. Motion capture data of such interactions is incorporated into a character's path planning by including information about the motion in the cost metric. This allows for better following of a user's input signals in complex environments. As a result, our method provides a more enriching user experience in real-time applications such as games.

Chapter 6

Flexible Multi-agent Motion Control

In previous chapters we have shown how effective a multitouch device can be for controlling the motion of a group of agents in a virtual environment. A clear advantage of a multitouch device over other input mechanisms in this case is the ability to register several simultaneous control inputs from a user. This fact means such devices offer the opportunity for users to interact with an application in an expressive manner. We believe that this expressibility translates well for control of crowds in interactive applications and creation of animation.

A major difficulty when designing multitouch control comes from the trade-off between keeping the control scheme simple and still being able to harness the expressive nature of touch devices. It is also important that a control scheme is able to handle variations in a user's performance of a gesture. Rekik *et al.* (2013) found that users often articulate gestures with one or both hands, using multiple fingers when performing similar tasks. Furthermore, users showed similar variations in their gestures when asked to provide control for the movement of robot groups [Micire *et al.* (2009)].

Often, to enable a user to perform complex actions, control schemes can place restrictions on the way in which a user interacts via the touch device, such as in our work in chapters 3 & 5. In that work we defined a two-dimensional mesh as an intermediary between a user's input and the movement of the crowd. In doing so the control scheme forces the user to provide input in a certain way. In this case, users can only interact by applying touch inputs directly to vertices on the mesh. As such, the configuration of the mesh affects where a user can apply their fingers to the touch device and subsequently articulate their control signal. Our previous control scheme also requires a user to manipulate the crowd using a certain number of fingers in order to elicit motions such as expanding and contracting or splitting of the crowd. Limiting the methods for providing input can affect the user's experience with the application, including how easily they can provide their intended control signal and how comfortable they are when attempting to do so.

As well as affecting how a user interacts with a touch device to provide their input, control schemes can limit the types of motion that can be produced by a user. For example, the use of an intermediate mesh constrains the crowd to move in a very specific, sometimes unnatural, manner. In particular, the natural interaction between characters in a crowd can be lost because the movement of the crowd is governed by the relative positioning of the vertices on the mesh. This approach is appropriate for controlling a crowd to perform motion whilst maintaining a formation but to create more free-form, natural crowd motion we need something more flexible. Instead of restricting how input can be provided, we wish for the user to be able to control the crowd in a way that is intuitive to them. Additionally, we wish to be able to generate motion for crowds of arbitrary size and shape and for this motion to be defined directly by variations in the user's input.

In the current chapter we present a data-driven approach that allows a user to perform control gestures in a flexible and intuitive manner. Our system uses gesture examples taken from real users to infer the intended control signal from a new user input and to produce a crowd motion accordingly. We believe that simultaneous inputs offer greater ability for a user to express their intent to the application. In our system, a user can provide touch input by employing a variable number of fingers using one or two hands, placed on the touch device at various orientations. To achieve this we apply a method for recognising the style of a user's gesture based on a set of features that are independent of common variations in a user's touch input. These features can not only handle different numbers of inputs but they also retain information on the sequence of touch inputs (stroke) that the user provides. This information is often missing in previous gesture recognition research [Rekik *et al.* (2014); Jiang *et al.* (2012); Vatavu *et al.* (2012)].

Our system uses our proposed features to detect the desired control signal in a continuous manner by identifying similar gestures from the examples in our data set. This is in contrast to previous work on multitouch gesture recognition (Section 2.3), whose goal is often to determine which gesture type, from a discrete list, a user's input most closely resembles. Once the type is determined, the specific action associated with that gesture is performed. This is a valid approach when the intention is for a user to perform a discrete set of gestures. However, in the current work, we wish for a user to be able to perform a continuous set of gestures such that the resulting actions i.e. the crowd motion, will vary with the user's input. This allows our system to produce new crowd motions that are not part of our original data set but reflect the control given by the user. Our system maps from a new user input to its corresponding crowd motion using the crowd motion data associated with the similar gesture examples identified in the classification step. The resulting crowd motion can be produced for an arbitrary number of agents using statistical models of the crowd's trajectories that maintain the original style of the raw motion data.

6.1 Contributions

1. We offer a data-driven method for inferring appropriate crowd motion based on a user's input signal. To control the crowd, our approach is not restricted by a set of prespecified crowd formations nor does it require an explicit set of control points in order to manipulate the crowd's movement. We allow a user to interact naturally with the touch device and generate the final crowd motion using this information.

2. We present a set of features that are invariant to variability in users' preferred touch input style. We show how these features can be used for recognising different properties of a user's multitouch input, allowing us to distinguish between a variety of control signals.

3. We use input controls and crowd motion pairs collected from real users to define the concept of a *gesture space* and its associated *crowd motion space*. We describe a mapping between these spaces that can be used to convert a new user input into its corresponding crowd motion.

4. We show how new sets of agent trajectories can be produced from models of crowd motion data that we term *crowd motion primitives*. We describe a method to combine a number of these primitives to generate new models capable of producing crowd motion not seen in the original data set.

6.2 Method Overview

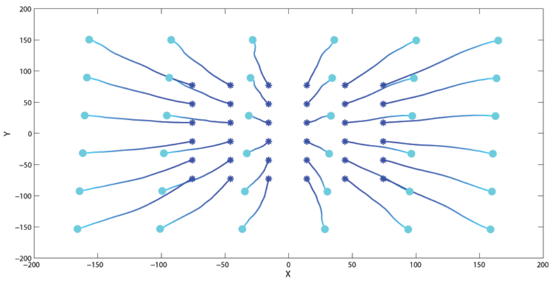
The approach presented in this chapter consists of two main stages: an offline stage, where data is collected and placed into data structures that describe the mapping from a user's input to a given crowd motion style, and an online stage, where our system receives new user input and converts this into a new crowd motion appropriate for the agents in the simulation. Here we present an overview of these stages.

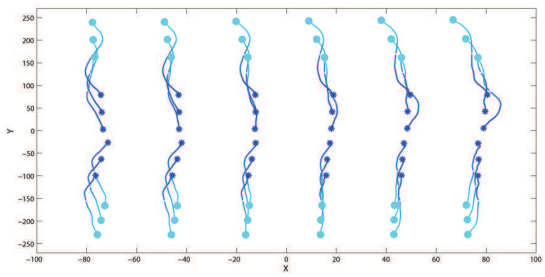
Offline - User-defined gestures for controlling different types of crowd motion are collected as described in Section 6.3. The inputs provided by users are normalised (Section 6.4.1) and a set of features is computed for each example Section 6.4.2. These gestures form data points in a *gesture space* that describes the possible types of user input that can be received by our system (Section 6.4). Each motion data example used to collect the gestures is modelled as a *crowd motion primitive* that represents a data point in a corresponding *crowd motion space*. These primitives can be used to produce new crowd motion of a similar style (Section 6.5.1). The pairing of example user gestures and *crowd motion primitives* forms a correspondence between *gesture space* and *crowd motion space* that can be used in our online phase to map from a new user gesture to its associated crowd motion.

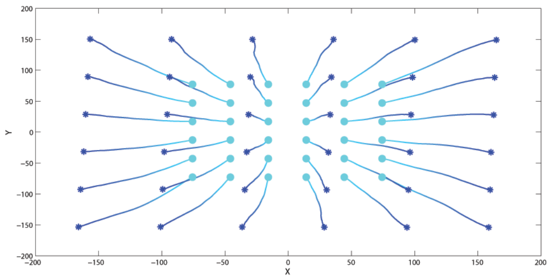
Online - Our online phase consists of a number of steps for converting a user's input into an appropriate crowd motion:

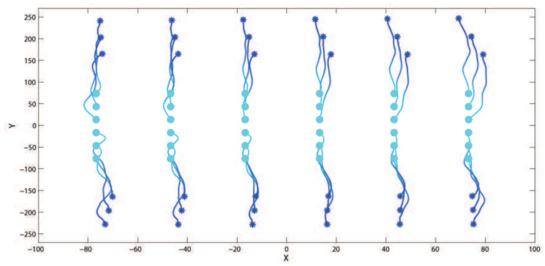
1. *Recording and processing of user input* - When a new input is received via a multitouch device it is first converted into a normalised gesture (Section 6.4.1) and a set of features is computed (Section 6.4.2) so that it can be compared to the user gesture examples in our data set.
2. *Continuous classification of the user gesture* - The representation of the new user gesture is determined by finding its neighbourhood in *gesture space* based on its set of features (Section 6.4.3).
3. *Mapping from gesture space to crowd motion space* - A new *crowd motion primitive* is generated by mapping from the point in *gesture space* represented by the new user gesture to a corresponding point in *crowd motion space*. This mapping is achieved by using the implicit relationship defined by our user gesture and *crowd motion primitive* pairs along with the neighbourhood information from step 2 (Section 6.5.2).

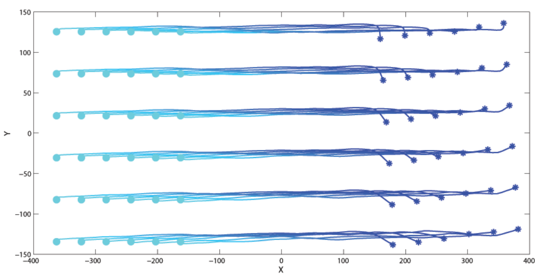
4. *Creating and applying new crowd motion* - A set of trajectories is generated for the crowd in the simulation using the *crowd motion primitive* created in the previous step. Each of these trajectories is assigned to an agent in the crowd based on their current positions (Section 6.5.3).


(a) Crowd “contract” motion


(b) Crowd “converge” motion


(c) Crowd “expand” motion


(d) Crowd “split” motion


(e) Crowd “straight” motion

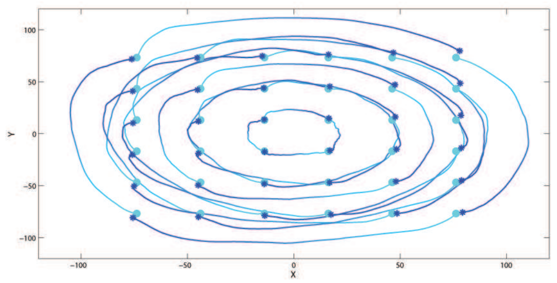

(f) Crowd “twist” motion

Figure 6.1: Examples of basic crowd motion shown to users to collect their control gestures. The light blue colour indicates the start of the motion and the dark blue indicates the end.

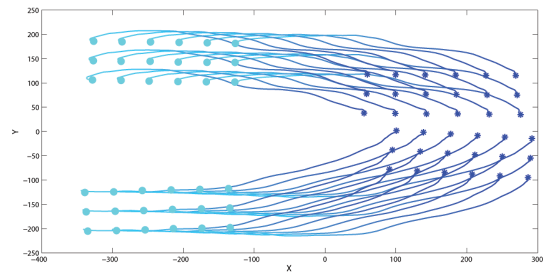
6.3 Data Collection

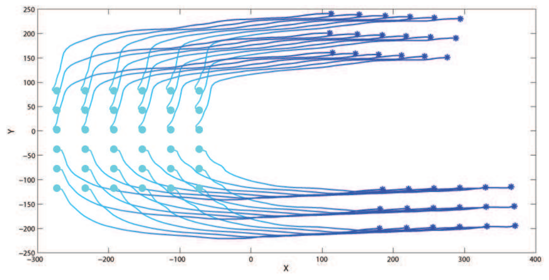
In this section we explain our process for collecting data on user-defined gestures for controlling a variety of crowd motions. First we ascertained six key components of crowd movement that we believe can define a number of different crowd manoeuvres

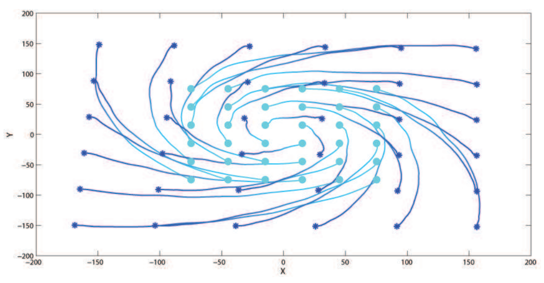
(Figure 6.1). These components capture the basic movement of a crowd of agents from a start point to a goal point (Figure 6.1(e)), the idea that the movement of agents in a crowd can increase or decrease its overall density (Figure 6.1(a) & (c) respectively), and the fact that a crowd can move in a circular motion around a given point (Figure 6.1(f)). This last "twist" example is also relevant to the movement of a crowd around corners, where the trajectory arc of agents closer to the corner is smaller than those further away i.e. on the outside edge of the crowd. The proposed motion components also account for instances where a crowd can form from, or split into, subgroups (Figure 6.1(b) & (d)).

The chosen motions by themselves are quite basic but in combination with one another they help to describe a variety of different crowd manoeuvres. As an example, the straight movement of the crowd can be combined with the splitting or merging motions to generate a scene where two groups of characters converge to a point on the other side of the environment or where a crowd splits apart whilst moving past an obstacle. These movements are useful for performing simple tasks and generating collective crowd movement. It is also plausible that a user could guide one crowd to split whilst moving another crowd between them using these kinds of movement. The choice of these crowd motion examples allows a user to provide basic gestures for their control which can then be layered to generate more interesting scenes. The motions are clear, and it is straightforward for a user to understand what the intention of the motion is. However, these motion examples make it difficult to generate more complex crowd movement, such as multiple characters crossing and moving past one another around a common centre point.

Examples of the basic styles of motion in Figure 6.1, along with hybrid motions incorporating two of these basic motions' styles (Figure 6.2), were created using the system presented in chapters 3 & 5. Ten participants, aged between 20 and 50, were presented with a number of these motion examples and asked to provide a gesture to control the crowd motion using two or more fingers. The gestures were recorded on a G4 multitouch overlay from PQ labs, attached to a 24" Acer S240HL LCD monitor. The orientation of the crowd motion on the screen was varied to prevent any bias in terms of the positioning of the hands when recording the gesture. Postprocessing of the user's inputs was performed to clean up any noise caused by the switching of touch input ID when the fingers became too close on the screen. This cleanup was minimal and the inputs remained untouched otherwise. As a result, 80 input-to-crowd-motion pairs were generated for the basic motions and 70 pairs for the hybrid motions. These


(a) Crowd “converge & move” motion


(b) Crowd “split & move” motion


(c) Crowd “twist & expand” motion

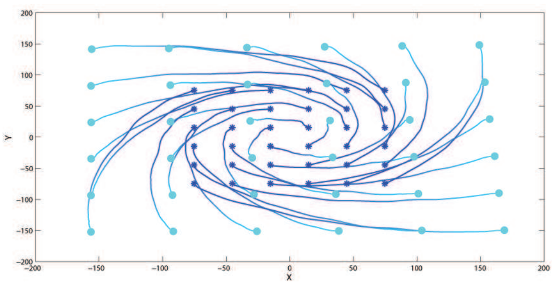

(d) Crowd “twist & contract” motion

Figure 6.2: Examples of hybrid crowd motion shown to users to collect their control gestures. These motions incorporate combinations of the styles of motions seen in Figure 6.1. The light blue colour indicates the start of the motion and the dark blue indicates the end.

pairings provided for the basic crowd motions form the basis for the *gesture space* and *crowd motion space* described in sections 6.4 & 6.5. The total set of provided user inputs for each basic motion style can be seen in Section 6.6.1 along with an analysis. The gestures provided by users for the hybrid crowd motions were used as inputs for our experiments on generating new crowd motions presented in section 6.5.2.

6.4 Gesture space

In previous work on multitouch gesture recognition, users’ inputs are defined in terms of a discrete set of gesture templates that map directly to a discrete set of actions to perform in an application. In the current work we wish for a user input to be defined in a continuous space, allowing each unique gesture to represent a different point in this space and therefore a different input to the application. Here we define a *gesture space*; a representation of all the possible gestures that a user can provide to our system. The continuous nature of this space means that variations in a user’s input can be used to generate variations in the final crowd motion (Section 6.5.2). The coverage of this *gesture space* is limited by the underlying gesture data used. As mentioned in Section 6.3, the crowd motions used in this work, and their related gestures, cover a number of different scenarios for control of crowd movement but do not comprehensively cover all possible user gestures. Instead, they are intended to cover continuous control of crowd movement by user input.

Our concept of a *gesture space* is similar to the idea of *Motion Fields*, described by Lee *et al.* (2014)). In this work the authors proposed a high-dimensional continuous space that incorporated the set of all possible motion states in character motion. Unlike character motion however, where the state of a character is well described by a consistent set of joint positions and velocities, the way in which a user performs a particular gesture can vary significantly from person to person. For example, a user can use a variable number of fingers to perform the same intended gesture, meaning a direct comparison of the touch inputs is not appropriate. We therefore propose a set of features that are independent of this variation but still capture the intent of the input, allowing us to effectively distinguish between different types of user gesture. In the rest of this section we describe our concept of *gesture space*, starting with how we define a gesture in terms of a user's input on a multitouch device (Section 6.4.1). We then outline the features that we use to differentiate various kinds of gesture (Section 6.4.2) and finally we show how these features can be used to define a gesture's location in the *gesture space* (Section 6.4.3).

6.4.1 Creating a Gesture

In the current work, a gesture is described by the set of trajectories corresponding to the distinct inputs provided by a user on a touch device. Since a touch device will record the positions of a user's input at discrete time intervals we can define the trajectory, g_n, of touch input n as a set of points over time:

$$g_n = \{\mathbf{g}_n(t) \mid t = 1, \ldots, T_n\}, \tag{6.1}$$

where T_n is the time at which the finger describing input n is removed from the touch screen device. Each point of the trajectory, $\mathbf{g}_n(t) \in \mathbb{R}^2$, is a vector describing the x and

y screen coordinates of touch input n at time t:

$$\mathbf{g}_n(t) = \begin{bmatrix} g_n^x(t) \\ g_n^y(t) \end{bmatrix}, \text{ for } t = 1,\ldots,T, \text{ where } T = \max_n(T_n). \tag{6.2}$$

Therefore, we can describe a gesture, G, as a set of N trajectories:

$$G = \{g_n \mid n = 1,\ldots,N\}, \tag{6.3}$$

where N is the number of distinct touch inputs provided in the user's gesture. For ease of notation we define a point cloud P containing N points, and its mean value, $\bar{P}$, as:

$$P = \{\mathbf{p}_n \mid n = 1,\ldots,N\}, \tag{6.4}$$

$$\bar{P} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{p}_n,$$

$$\mathbf{p}_n \in \mathbb{R}^2.$$

In terms of a gesture, G, $G^{P(t)}$ is defined as the point cloud created by the positions of the N touch inputs at timestep t such that:

$$G^{P(t)} = \{\mathbf{g}_n(t) \mid n = 1,\ldots,N\}, \tag{6.5}$$

From this we can further define the point cloud for the entire gesture G as:

$$G^{\mathbf{P}} = \{G^{P(t)} \mid t = 1,\ldots,T\}. \tag{6.6}$$

Variations in sampling rate can occur due to different sensing hardware and input software, and as a result the value for T can be affected by the movement speed of the user's touch inputs. To compare different gestures even when they are provided at different movement speeds we resample G such that each touch input trajectory, $g_n \in G$, defined by the original T points is defined by T' points instead. For this purpose we use a cubic Hermite spline to interpolate the original trajectories of the user's touch inputs at the new sampling resolution. A function for resampling a two-dimensional trajectory, z, from T data points to T' data points can be defined as follows:

$$S(z,T,T') = \{H\left(z(t_0),m(t_0),z(t_0+1),m(t_0+1),s(t',T')T-t_0\right) \mid t'=1\ldots T'\}, \quad (6.7)$$

$$\begin{aligned}\text{where } s(x,y) &= \frac{x}{y},\ 1 \le x \le y,\ y>0,\\ t_0 &= \lfloor s\left(t',T'\right)T\rfloor,\\ m(t) &= \frac{1}{2}(z(t+1)-z(t-1)),\end{aligned}$$

$$\begin{aligned}\text{and } H(\mathbf{p}_0,\mathbf{m}_0,\mathbf{p}_1,\mathbf{m}_1,r) &= \left(2r^3-3r^2+1\right)\mathbf{p}_0 + \left(r^3-2r^2+r\right)\mathbf{m}_0\\ &\quad + \left(-2r^3+3r^2\right)\mathbf{p}_1 + \left(r^3-r^2\right)\mathbf{m}_1,\\ \text{for } &\mathbf{p}_0,\mathbf{m}_0,\mathbf{p}_1,\mathbf{m}_1 \in \mathbb{R}^2,\\ &r \in [0,1],\end{aligned}$$

where H is the Hermite spline function. The resampled gesture G' is subsequently defined as:

$$G' = \{S(g_n,T,T') \mid n = 1,\ldots,N\}. \quad (6.8)$$

The choice of value for T' is important since undersampling would remove too much information from the original gesture but oversampling would add unnecessary detail and increase computational overhead in later stages. Wobbrock *et al.* (2007) suggested that $T' = 64$ was an effective value for resampling user gestures and we found this worked well in our experiments. Given the resampled gesture, G', we produce the final, normalised gesture, $\hat{G}$, by translating the gesture's point cloud, $G'^{\mathbf{P}}$, so that its centroid is at the origin and each point is scaled into unit range $[0,1]\times[0,1]$ whilst preserving the gesture's shape. If we define P_x and P_y as the set of x and y positions for all the points in $G'^{\mathbf{P}}$ we can use the following equation to achieve this:

$$\hat{G} = \{\frac{\mathbf{g}'_n(t) - \bar{G}'^{\mathbf{P}}}{\lambda} \mid n = 1,\ldots,N,\ \ t = 1,\ldots,T'\}, \quad (6.9)$$

$$\text{where } \lambda = \max\{\max_{p_{x_1},p_{x_2}\in P_x}(|p_{x_1}-p_{x_2}|),\ \max_{p_{y_1},p_{y_2}\in P_y}(|p_{y_1}-p_{y_2}|)\}.$$

In this case, λ acts as a scaling factor based on the axis-aligned bounding box of the set of points $G'^{\mathbf{P}}$. This approach of gesture preprocessing is common throughout gesture recognition research [Wobbrock *et al.* (2007); Anthony & Wobbrock (2010); Vatavu *et al.* (2012); Rekik *et al.* (2014)]. For ease of notation, in the rest of this chapter we will assume that G refers to the normalised gesture, $\hat{G}$, and $T = T'$ unless explicitly stated.

6.4.2 Gesture Features

As with *Motion Fields* [Lee *et al.* (2014)], the concept of similarity between gestures is key to our definition of a *gesture space*. Here we define a set of features for an arbitrary gesture, G, that we believe can distinguish well between different types of user input.

Centroid Feature

The centroid feature represents the average position of the user's touch inputs (centroid) over time. Given equation 6.5, we can define this feature as a function of the input gesture G, by producing an array of the average value of the gesture's point clouds for $t = 1,\dots,T$:

$$C(G) = [\bar{G}^{P(1)}\ \bar{G}^{P(2)}\ \cdots\ \bar{G}^{P(T)}]^T. \tag{6.10}$$

Distance to Centroid Feature

The distance to centroid feature represents the average distance of each touch input relative to their centroid over time. For a given point set, P, the average centroid distance, $l(P)$, is calculated as:

$$l(P) = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{p}_n - \bar{P}\|,\ \ \mathbf{p}_n \in P. \tag{6.11}$$

We compute this feature for the input gesture G by applying equation 6.11 to each of its point clouds for $t = 1,\dots,T$:

$$L(G) = [l(G^{P(1)})\ \ l(G^{P(2)})\ \ \cdots\ \ l(G^{P(T)})]^T. \tag{6.12}$$

Rotation Feature

The rotation feature represents the average cumulative change in rotation over time of each touch input around the centroid of all the inputs. For two point sets, P_0 and P_1, the average change in rotation of corresponding points in the point sets is given by:

$$r(P_0,P_1) = \frac{1}{N}\sum_{n=1}^{N} \theta(\mathbf{p}_{n0} - \bar{P}_0, \mathbf{p}_{n1} - \bar{P}_1),\ \ \mathbf{p}_{n0} \in P_0,\ \ \mathbf{p}_{n1} \in P_1 \tag{6.13}$$

$$\text{where } \theta(\mathbf{v}_1,\mathbf{v}_2) = \operatorname{atan2}(\mathbf{v}_2 \times \mathbf{v}_1, \mathbf{v}_2 \cdot \mathbf{v}_1),\ \ \mathbf{v}_1,\mathbf{v}_2 \in \mathbb{R}^2 \text{ and}$$

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\frac{y}{x} & x > 0 \\ \arctan\frac{y}{x} + \pi & y \geq 0, x < 0 \\ \arctan\frac{y}{x} - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

We compute the final feature by using r on a gesture, G's point clouds for $t = 1,\dots,T$:

$$R(G) = \mathbf{R} = [0, r(G^{P(2)}, G^{P(1)}) + \mathbf{R}(1)\ \ \cdots\ \ r(G^{P(T)}, G^{P(T-1)}) + \mathbf{R}(T-1)]^T. \tag{6.14}$$

Minimum Oriented Bounding Box Feature

This feature records the minimum and maximum dimension (henceforth referred to as width and height respectively) of the minimum oriented bounding box (MOBB) for the positions of the touch inputs at each timestep. Given a function, MOBB(P), that returns a set of points, ordered counterclockwise, that define the MOBB of P using the rotating calipers method [Toussaint (1983)], we define this feature as:

$$b(P) = \Big[w(P), h(P)\Big], \tag{6.15}$$

$$\begin{aligned} \text{where } w(P) &= \min_{\mathbf{p}_n,\mathbf{p}_{n+1} \in \text{MOBB}(P)} (\|\mathbf{p}_n - \mathbf{p}_{n+1}\|), \\ h(P) &= \max_{\mathbf{p}_n,\mathbf{p}_{n+1} \in \text{MOBB}(P)} (\|\mathbf{p}_n - \mathbf{p}_{n+1}\|), \end{aligned}$$

As with the other gesture features, we produce an array of the values from function b for the input gesture G:

$$B(G) = [b(G^{P(1)})\ \ b(G^{P(2)})\ \ \cdots\ \ b(G^{P(T)})]^T. \tag{6.16}$$

Distance Between Gestures

Given this set of features we can define a function for computing the distance between two gestures, G_0 and G_1:

$$\begin{aligned} D(G_0,G_1) &= \alpha \text{DTW}(C(G_0), C(G_1)) + \\ &\quad \beta \text{DTW}(L(G_0), L(G_1)) + \\ &\quad \gamma \text{DTW}(R(G_0), R(G_1)) + \\ &\quad \delta \text{DTW}(B(G_0), B(G_1)), \end{aligned} \tag{6.17}$$

where DTW provides a distance between two n-dimensional vectors using dynamic time warping [Berndt & Clifford (1994)], and α, β, γ, and δ, are weights for each feature. In the current research we found $\alpha = 0.04$, $\beta = 0.36$, $\gamma = 0.36$, and $\delta = 0.24$ to work well for defining the distance between gestures in our dataset.

6.4.3 Forming Gesture Space

Generally speaking, a *gesture space* constitutes the set of all possible gestures that a user can perform, however, in practice it is represented by a finite collection of gesture data examples. In the current work, our *gesture space* is defined by the set of user-provided gestures, $\mathbf{G}_U$, for different styles of crowd motion (Section 6.3). Given our distance function in equation 6.17, a gesture G's location in *gesture space* can be described by its neigbourhood, $\mathbf{G}_N$ and a set of similarity weights, w, that describe how close G is to other gestures in its neighbourhood. Here we define $\mathbf{G}_N$ to be the set of K-nearest gestures in $\mathbf{G}_U$:

$$\mathbf{G}_N = \{G_i \mid i = 1, \ldots, K\}, \;\; G_i \in \mathbf{G}_U, \tag{6.18}$$

where G_i is the ith closest gesture in $\mathbf{G}_U$, and the similarity weights are computed as:

$$\begin{aligned} w &= \{w_i \mid i = 1, \ldots, K\}, \qquad (6.19) \\ w_i &= \frac{1}{\mu D(G, G_i)}, \;\; G_i \in \mathbf{G}_N, \\ \mu &= \sum_{i=1}^{K} D(G, G_i). \end{aligned}$$

Here, μ acts as a normalisation factor to make sure that all the weights sum to 1. In our experiments we found $K = 10$ to be appropriate.

6.5 Crowd Motion Space

Here we present our formulation of a *crowd motion space* which is conceptually similar to a *gesture space*. We represent a crowd motion in a similar way to a gesture (Section 6.4.1), such that a crowd motion, M, consisting of the movement on a two-dimensional plane of N agents for T timesteps is defined as:

$$M = \{m_n \mid n = 1, \ldots, N\}, \tag{6.20}$$

$$\text{where } m_n = \{\mathbf{m}_n(t) \mid t = 1, \ldots, T\},$$
$$\mathbf{m}_n(t) = \begin{bmatrix} m_n^x(t) \\ m_n^y(t) \end{bmatrix},$$

where $m_n^x(t)$ and $m_n^y(t)$ are the x and y position respectively of the nth agent at time t. We define the point set, $M^{P(t)}$ to be the positions of the N agents at timestep t such that:

$$M^{P(t)} = \{\mathbf{m}_n(t) \mid n = 1, \ldots, N\}, \tag{6.21}$$

and the point set for the entire crowd motion, M as:

$$M^{\mathbf{P}} = \{M^{P(t)} \mid t = 1, \ldots, T\}. \tag{6.22}$$

To provide consistency between crowd motion examples we resample the trajectories of each crowd motion to T' points using equation 6.7 and subtract the centroid of the motion so that it is at the origin:

$$M' = \{S(m_n, T, T') \mid n = 1, \ldots, N\}, \tag{6.23}$$

$$\hat{M} = \{\mathbf{m}'_n(t) - \bar{M}'^{\mathbf{P}} \mid n = 1, \ldots, N, \; t = 1, \ldots, T'\}.$$

As with gestures, for ease of notation we assume that M refers to a normalised $\hat{M}$ and $T = T'$ for the rest of this chapter unless otherwise stated.

To generate our crowd motion space we use the set of basic motions, $\mathbf{M}_U$, presented to users during the data collection stage (Section 6.3). Unlike in a *gesture space* we wish to produce crowd motion for crowds consisting of any number of agents. Furthermore, we wish to produce new styles of motion based on a user's gesture and not just on the types of motion in our data set. This is not achievable by simple interpolation of the original data examples and becomes more difficult if we want to incorporate motion examples that include groups of arbitrary size. For these reasons we create *crowd motion primitives* from the motions in $\mathbf{M}_U$ that are similar in form to *morphable motion primitives* [Min *et al.* (2009); Min & Chai (2012)]. In Section 6.5.1 we describe how we generate our *crowd motion primitives* and how we can blend between multiple primitives to produce new crowd motion that is appropriate to a user's input gesture.

6.5.1 Generating Motion Models

From a given crowd motion example, $M_U \in \mathbf{M}_U$, we construct an $N \times 2T$ matrix, E^{M_U}, where the nth row consists of a sequential concatenation of the position of the nth trajectory, m_n, in the crowd example:

$$E^{M_U} = \begin{pmatrix} m_1^x(1) & m_1^y(1) & \cdots & m_1^x(T) & m_1^y(T) \\ m_2^x(1) & m_2^y(1) & \cdots & m_2^x(T) & m_2^y(T) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m_N^x(1) & m_N^y(1) & \cdots & m_N^x(T) & m_N^y(T) \end{pmatrix} \tag{6.24}$$

We apply principle component analysis (PCA) on the matrix, E^{M_U}, to produce a set of eigenvectors, $E_V^{M_U}$, that describe the modes of the crowd motion data, and a set of eigenscores, $E_S^{M_U}$, that represent the projection of the original data into the space defined by $E_V^{M_U}$. The nth row in the original data matrix, $E_{[n]}^{M_U}$, can be recovered by using the mean trajectory from the data, $\bar{m}^{M_U}$, the nth row of eigenscores, $E_{S[n]}^{M_U}$, and the set of eigenvectors:

$$E_{[n]}^{M_U} = \bar{m} + E_{S[n]}^{M_U} E_V^{M_U T} \tag{6.25}$$

In this case, the eigenvectors describe the main axes of variation for the example trajectories, and the eigenscores denote the weights to apply to these axes to reproduce the trajectories in the crowd motion example. By modelling the distribution of the eigenscores we can sample new scores that will produce trajectories similar to those in the original example. We model this probability distribution, $\phi(E_S^{M_U})$ using a Gaussian mixture model (GMM) whose parameters are estimated automatically using an Expectation-Maximization algorithm (Bishop (1996)). With GMMs, we can define a set number of Gaussian components that constitute the overall probability distribution. This property allows us to model any clustering of eigenscores that might be present in crowd motion examples. This is particularly relevant to our basic “split” and “converge” motions which involve the movement of two separate sets of agents. These examples produce two aggregated sets of eigenscores, one per subgroup of agents. With a single Gaussian we would not be able to capture this property and thus be unable to effectively reproduce these motion styles. For other motion types that do not show this clustering, such as for “twist” examples, GMMs produce an appropriate mixture of their constituent Gaussians to model this. We found that a general rule of thumb is to set the number of GMM components equal to the maximum number of subgroups present in the set of crowd motion examples. This allows any clustering property of the crowd motion to be maintained. Should a motion example contain fewer subgroups than this number of components then they will blend together to form an appropriate distribution for the model. In the current work, using two Gaussian components allowed us to model the different styles in our crowd motion data set well. Since users only use two hands to control the crowd this number of components is appropriate for the current application.

With PCA we can also reduce the dimensionality of the crowd motion data by altering the number of modes we use for recovery in Equation (6.25). This helps us to reduce computation time when creating new motion data, and also to remove noise from the original data caused by tiny variations in an agent's trajectory. We found that with as little as 15 modes we could retrieve $> 99\%$ of the variations for each of the original crowd motion data examples in $\mathbf{M}_U$. We can define a *crowd motion primitive* for a motion example, M_U as a combination of the motion data's mean, eigenvector matrix, and GMM: $\mathcal{M}(M_U) = \{\bar{m}^{M_U}, E_V^{M_U}, \phi(E_S^{M_U})\}$. We create our *crowd motion space* not from the set of crowd motion examples, $\mathbf{M}_U$, but from their corresponding *crowd motion primitives*, $\{\mathcal{M}(M_U) \mid M_U \in \mathbf{M}_U\}$.

6.5.2 Generating a New Crowd Motion Model

From our data collection, each user-defined gesture, G_U, corresponds directly to a motion example, M_U such that there is an implicit mapping $\psi : G_U \to M_U$. As such, we can map directly from each gesture, G_i, in the set of neighbour gestures determined in Section 6.4.3, to their corresponding *crowd motion primitives* in *crowd motion space*:

$$\mathcal{M}(M_i) = \mathcal{M}(\psi(G_i)), \;\; G_i \in \mathbf{G}_N, \;\; M_i \in \mathbf{M}_U. \tag{6.26}$$

As a result, $\mathcal{M}(M_i)$ represents the ith neighbour to the, as yet undetermined, *crowd motion primitive* corresponding to the new user gesture. To produce the final *crowd motion primitive* we create a model based on the weighted sum of each neighbour model using the similarity weights, w, calculated in Section 6.4.3:

$$\mathcal{M}(\hat{M}) = \sum_{i=1}^{K} w_i \mathcal{M}(M_i) = \sum_{i=1}^{K} \{w_i \bar{m}^{M_i}, w_i E_V^{M_i}, w_i \phi(E_S^{M_i})\} \tag{6.27}$$

Since the trajectories in each motion example have been resampled, combining the mean trajectories in this way is simply a matter of summing the weighted trajectory vectors:

$$\bar{m}^{\hat{M}} = \sum_{i=1}^{K} w_i \bar{m}^{M_i}. \tag{6.28}$$

We also combine the eigenvectors for the models in the same way:

$$E_V^{\bar{M}} = \sum_{i=1}^{K} w_i E_V^{M_i}. \tag{6.29}$$

To ensure orthonormalisation of the new eigenvectors we apply the modified Gramm-Schmidt method [Cheney & Kincaid (2008)], and we enforce each eigenvector to be positive so that they do not cancel each other out when combined.

To produce an appropriate GMM for the new *crowd motion primitive*, a naive combination of the neighbour's GMM parameters is not sufficient. If this approach is taken, cross fading of the Gaussian components can occur if the mean and variance of the *n*th component in each model do not match well. To alleviate this, we first establish the correspondence between the GMM components of each neighbour motion primitive with those of the closest neighbour. We do this using the mass transport solver approach described in Section 3.5, where the target points are the means of the closest neighbour's GMM components, and the source points are the GMM component means of the neighbour currently being considered. The weight for each GMM component is set to its weighting in the GMM mixture. This approach is similar to the work on displacement interpolation presented by Bonneel *et al.* (2011). Once a correspondence is established we produce a weighted sum of the parameters of matching components and use this information to produce the final GMM distribution. We find this technique is able to match components well between GMMs and prevent the issue of cross fading.

6.5.3 Applying the Generated Motion Model

To produce new motion we sample N scores from our new GMM, where N is the number of agents in the crowd for which we are creating the motion. These scores are then applied to the interpolated eigenvectors and mean trajectory of $\mathcal{M}(\hat{M})$ to generate a new set of trajectories using Equation (6.25). We assign these new trajectories to individual agents based on the current configuration of the crowd. The start points of each trajectory are used as the set of goal points and the agents' positions as the source points and we solve for the assignment using the mass transport solver as in Section 3.5. The agents then follow their provided trajectory to produce the motion of the crowd in response to the user's gesture.

6.6 Experimental Results

In this section we provide experimental results that demonstrate our system's ability to recognise the style of a user's gesture and to produce a crowd motion that replicates this. We start with an analysis of the gestures for controlling crowd motion collected from users (Section 6.6.1) and the effectiveness of our proposed features to distinguish between different types of user input (Section 6.6.2). We then show examples of the motions produced by our system in response to a variety of user gestures, including those that are not present in the original dataset (Section 6.6.3).

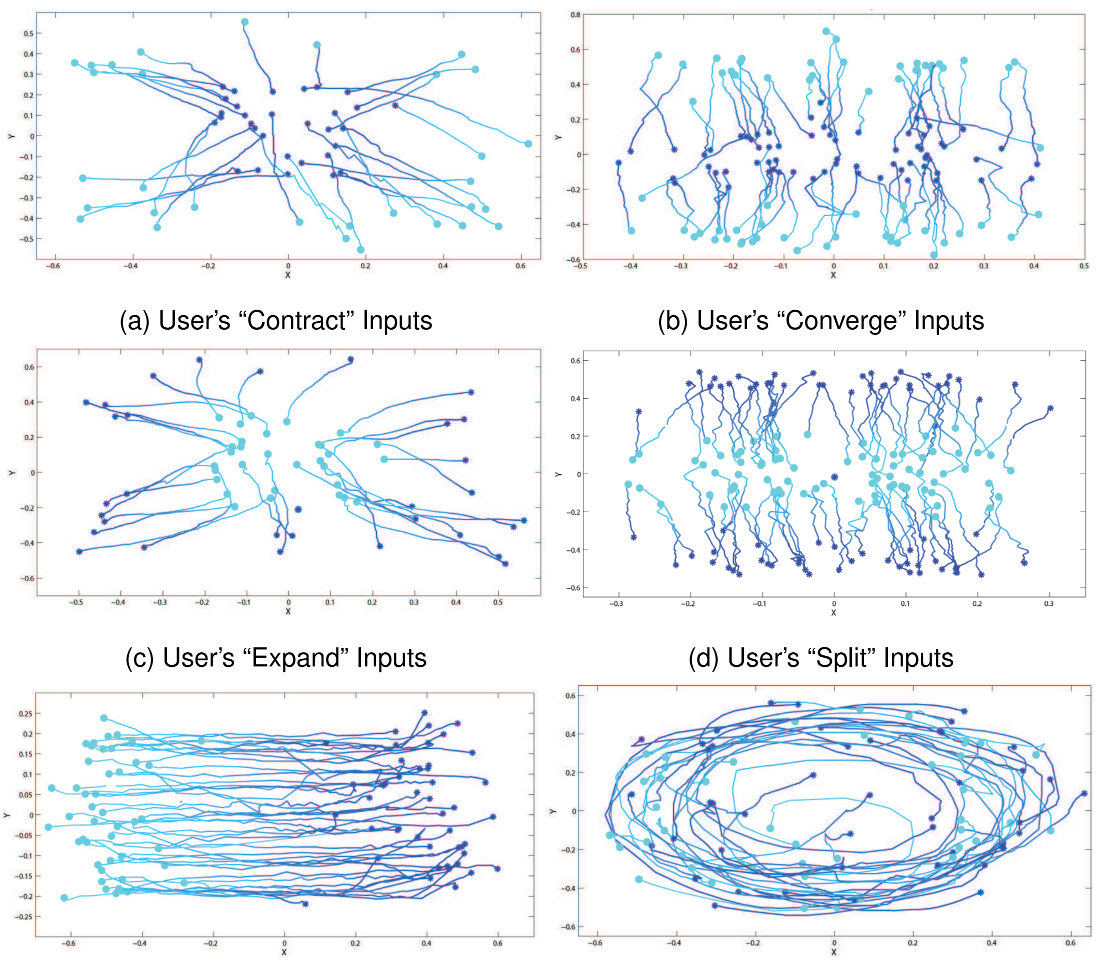
6.6.1 User Input Analysis

In this section we present an analysis of the gestures collected from users for controlling the basic crowd motions shown in Figure 6.1. We show how our features described in Section 6.4.2 are expressive enough to distinguish the styles of user gesture provided for the different types of basic crowd motion.

Figure 6.3 shows plots for all of the user inputs provided per crowd motion type. There is a clear distinction between the different sets of user inputs, showing that user's alter how they choose to control the crowd based on the overall crowd movement. For those crowd motions that occur opposite to one another i.e. expand and contract motions or split and converge motions, it can be seen that user's provide equally opposing gestures that signify this difference (Figure 6.3(a) - (d)). It is also clear that there is a difference in the style of gesture provided for conveying motions where the crowd expands or contracts (Figure 6.3(a) and Figure 6.3(c)) and for motions where the crowd splits or merges (Figure 6.3(b) and Figure 6.3(d)). Generally speaking these gestures closely resemble the true crowd motion.

Centroid Feature

The position of the centroid for the duration of different types of user gesture can be seen in Figure 6.4. As would be expected, the gestures for straight crowd movement show the greatest movement of the centroid from the start of the gesture. The expanding and contracting classes both show a significant change in the position of the centroid along the y axis. Since the feature for distance to the centroid shows a decrease in distance between the fingers in these two gestures, this indicates that the user prefers to move a finger (or multiple fingers) less than others whilst performing the gesture. This would explain why this shift in the mean position of the fingers occurs.


(a) User's "Contract" Inputs

(b) User's "Converge" Inputs

(c) User's "Expand" Inputs

(d) User's "Split" Inputs

(e) User's "Straight" Inputs

(f) User's "Twist" Inputs

Figure 6.3: User's inputs provided for controlling various crowd motions. The light blue colour indicates the start of the input and the dark blue colour indicates the end.

Distance to Centroid Feature

Figure 6.5 shows the average distance to centroid feature computed on the gestures provided for different classes of crowd motion across all input examples. It is clear that this feature is able to distinguish between three different subsets of the crowd motion gestures: 1) Gestures provided for the crowd's movement towards each other (contract and converge movement), 2) Gestures provided for the crowd's movement away from each other (split and expand movement), and 3) Gestures for motion where the crowd remain a similar distance away from one another (twist and straight motion). Not only are these three subsets distinct from one another, this property is consistent with the corresponding crowd motion for each type of gesture.As a consequence, this feature is unable to separate the inputs within these subsets but this is accounted for

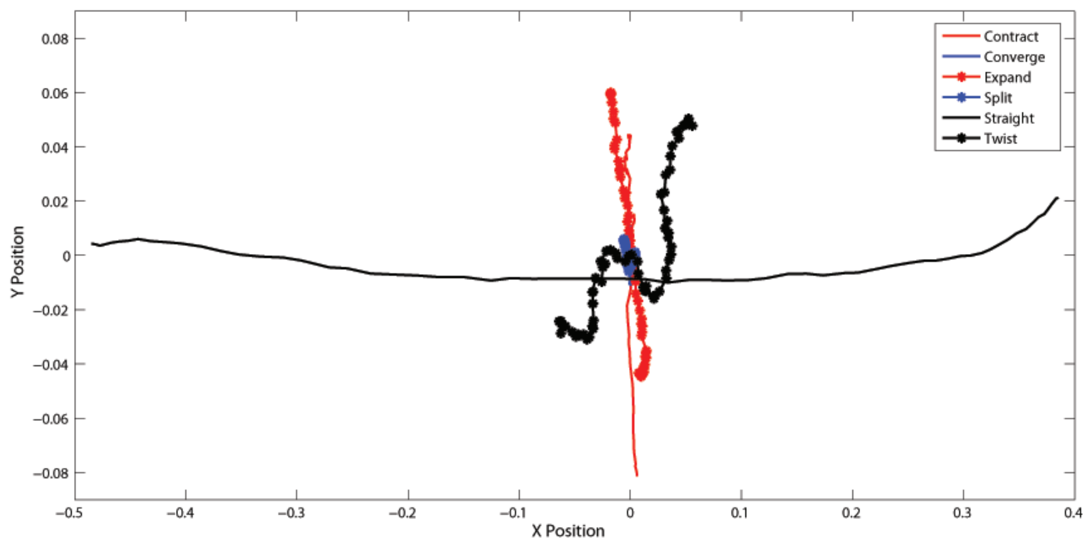


Figure 6.4: Centroid feature for different user gesture classes. The user gestures for controlling the “straight” crowd motion show the most significant change in the centroid of the user’s touch inputs.

using the other proposed gesture features.

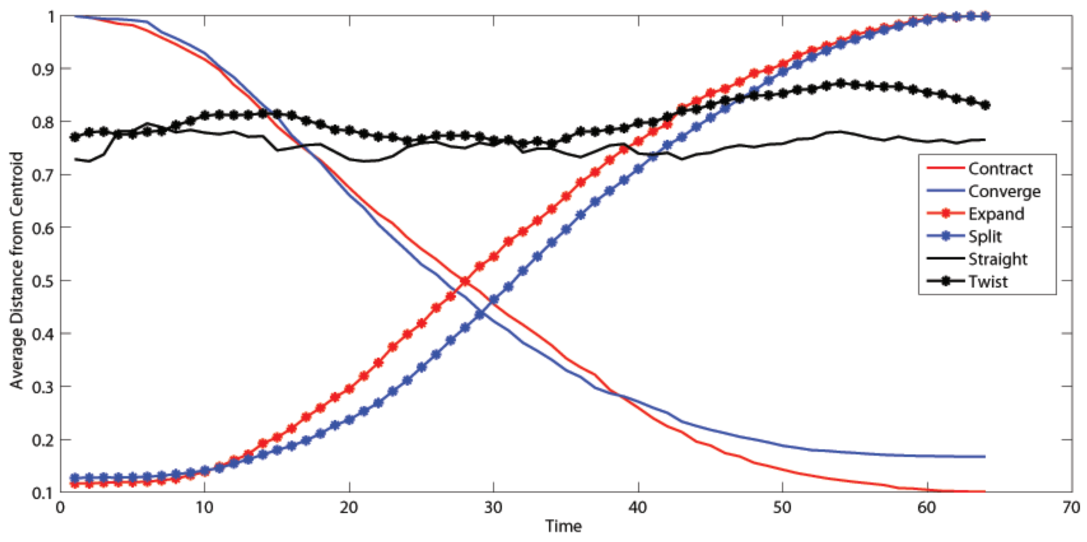


Figure 6.5: Distance to Centroid Feature for different user gesture classes. This feature clearly shows that it can distinguish between three significant subsets of gesture styles, coupling the “converge” & “contract”, “split” & “expand”, and the “straight” & “twist” types of gesture.

Rotational Feature

The difference in the rotational feature between different classes of crowd motion can be seen in Figure 6.6. This figure shows the average of this feature for each of the different types of motion. The most rotational of the crowd motions (the twist motion) also shows the greatest total rotation in the user's gestures. In some gestures for straight motion the user provided a slightly curved trajectory that is likely to account for the greater rotational feature shown towards the middle of the gesture. In all other cases the total rotation is low, indicating that this feature is appropriate for establishing the presence of a twist style command in a user's gesture and for separating this from the presence of other styles of gesture that might be provided as input.

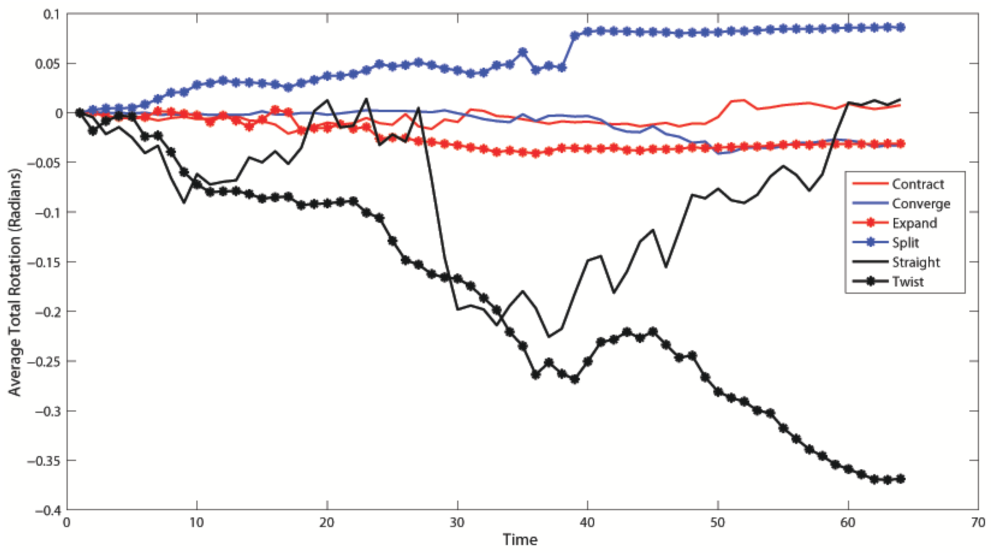


Figure 6.6: Average total rotation of user's inputs for different user gesture classes. Here we can see that most of the gesture types show little to no total rotation of the user's touch inputs. However, there is a significant total rotation shown by the "twist" (and to some extent the "straight") gesture type showing how this feature can help to identify the presence of this control in a user gesture.

Minimum Oriented Bounding Box Feature

For this feature we track the minimum oriented bounding box of the touch inputs at each time step and encode the feature as a 2D function using the values for the minimum and maximum dimension of the box (henceforth referred to as width and height respectively). Figure 6.7 shows these 2D functions for each of our classes of crowd motion gesture, with the width plotted on the x axis and the height plotted on the y axis. We can see that for both contract and expand gestures there is a strong positive correlation between the width and height of the minimum bounding box. The converge, and particularly the split gestures do not show as significant a correlation between the dimensions of their minimum oriented bounding box.

By looking at Figure 6.8, the values for the width and height of the bounding box in contract and expand gestures can be seen to decrease and increase respectively, highlighting the difference between the two motions, despite their similar shape in Figure 6.7. This shows that for these gestures the minimum bounding box shrinks or grows appropriately but its shape remains relatively square throughout. In contrast, the split and converge gestures show a similar, consistently small, value for the width of the bounding box but an increase in the height of the bounding box. This reflects the fact that the gestures show touch inputs moving apart from one another but only along one axis. In the user's gesture, this results in the bounding box looking like a rectangle being stretched (in the case of split) or compacted (in the case of converge).

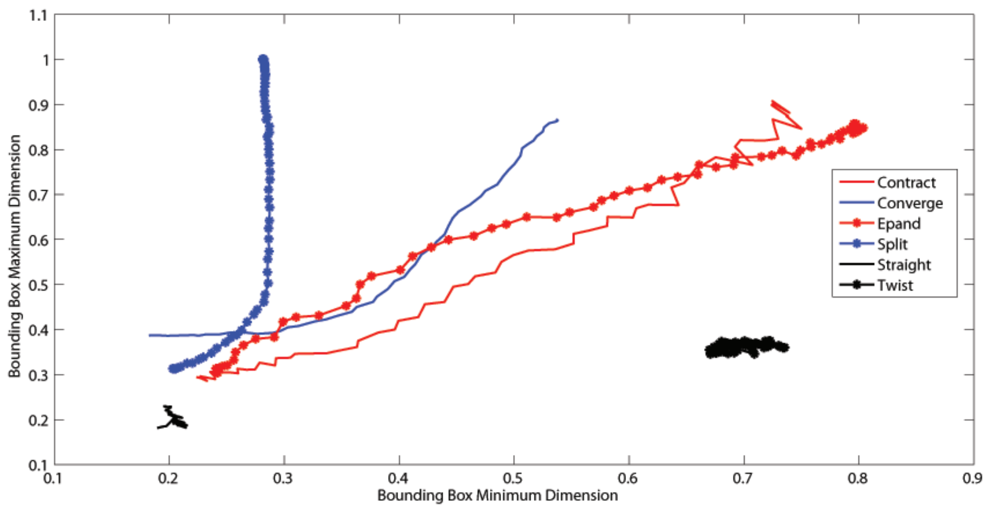


Figure 6.7: Combined minimum oriented bounding box feature (2D) for different user gesture classes. There is a clear separation between the "contract" & "expand" gestures from the "split" & "converge" gestures, something that is not seen in the basic Distance to Centroid feature shown in Figure 6.5

For the split and converge gestures, the curvature of the 2D functions in Figure 6.7 occurs as a result of a switch between the sides of the bounding box that represent the minimum and maximum dimensions. In the case of a converge gesture, this switch occurs as the fingers become closer towards the end of the gesture. This can be seen in Figure 6.8, where the values for each dimension become closer as time increases. The opposite is true of the split gesture. Finally, for Figure 6.7 we see that twist and straight gestures show consistent values for both the width and height of the bounding box throughout the gesture. For the twist gesture, Figure 6.8 shows that the height of the bounding box is much greater than the width of the bounding box but for the straight gestures these are always at a similar value. This indicates that a user either places their fingers further apart during a twist gesture as compared with a straight gesture or that this placement remains more colinear (forming a thin rectangular bounding box) over time.

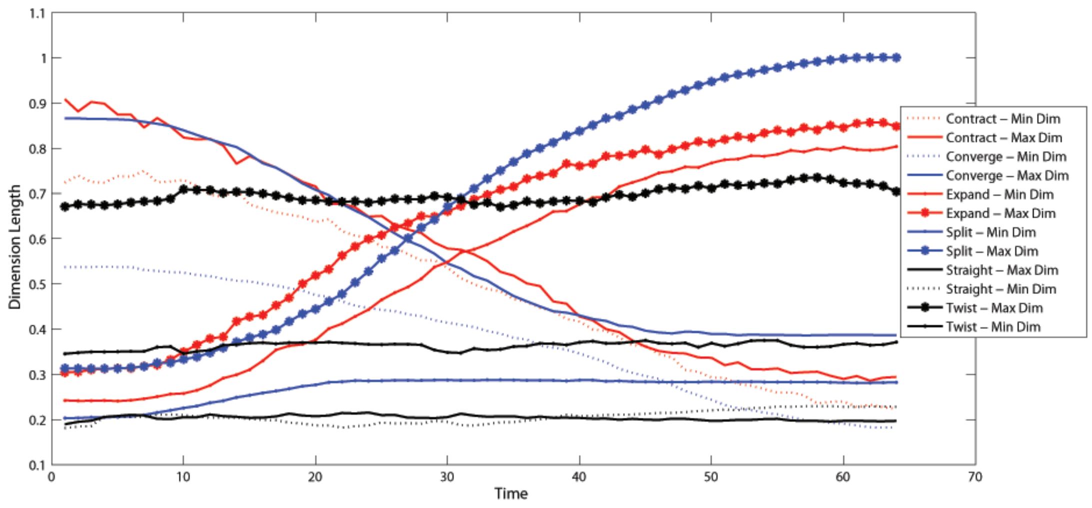


Figure 6.8: Change in dimensions over time for the minimum oriented bounding box feature in different user gesture classes. For “converge” (“split”) gestures the values for these dimensions become closer together (further apart) over time. This is different to “contract” (“expand”) gestures where the values for dimensions remain in a similar proportion, despite changing absolute value.

6.6.2 Classification

For testing the usefulness of the suggested features for classification we ran leave-one-out cross validation on our basic gesture data set. We classified each user input in turn using all other user input examples as the training set. To classify we used weighted nearest neighbour voting, where the class with the highest total weight was chosen as the classification.

Figure 6.9 shows a confusion matrix with the results of this classification test. This matrix shows a strong ability for the features presented here to classify each class of input gesture. All the classes (with the exception of contract) were classified correctly more than 80% of the time. Although not as strong as the rest of the classes, the contract class could still be classified correctly more than 60% of the time using the current set of features. The incorrect classification of the contract gestures as converge types is likely due to the similarity of their rotational and distance to centroid features.

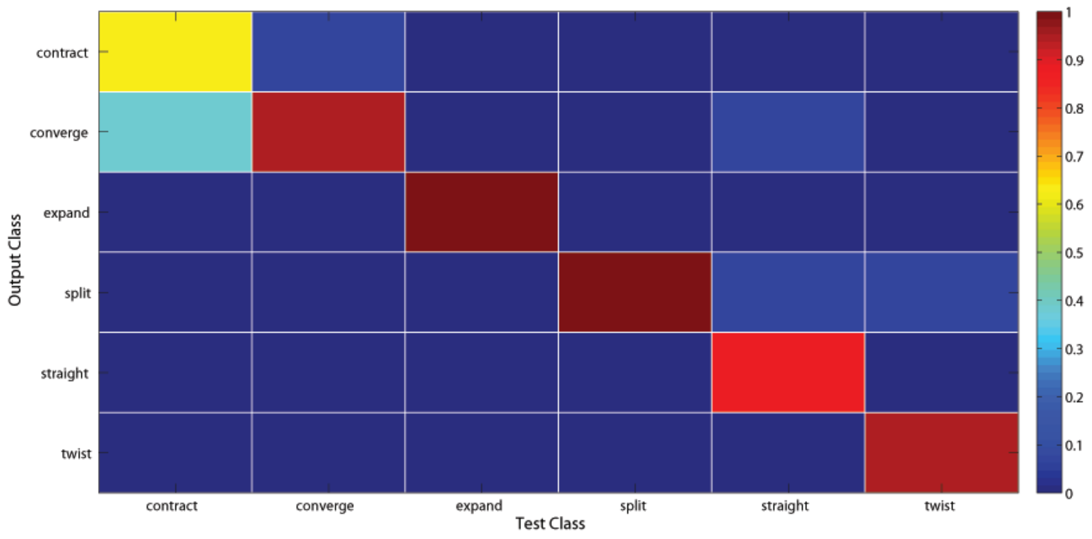
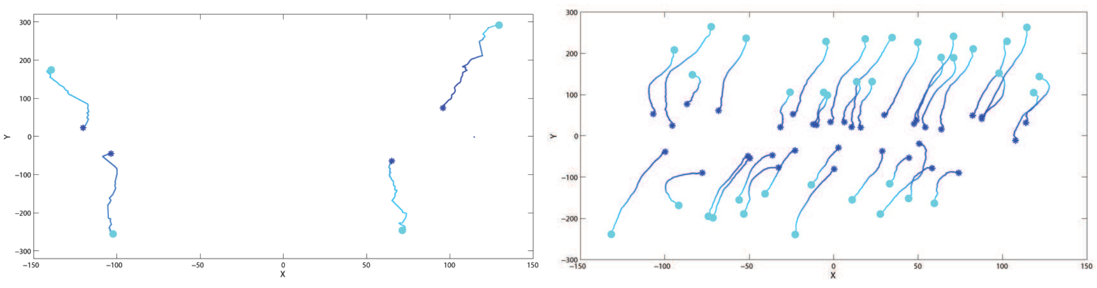


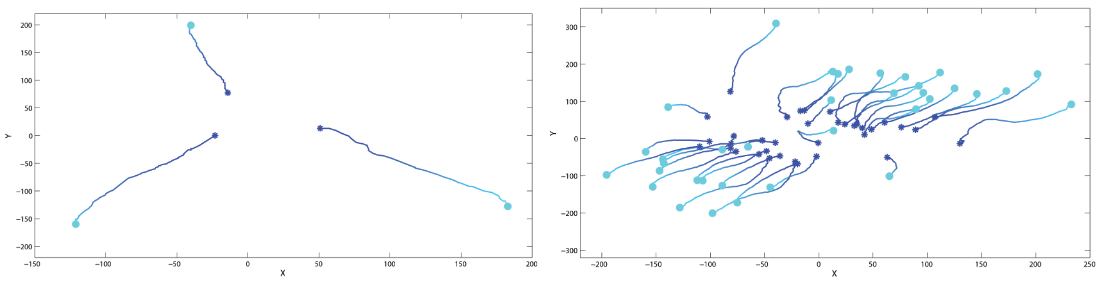
Figure 6.9: Confusion matrix for classification of basic user input gestures. Values in column i, row j indicate the proportion of all ith test gestures that were classified as the jth output gesture.

6.6.3 Producing Crowd Motions

To show the effectiveness of our method we produced a set of crowd motions from a number of example user inputs. Figures 6.10-6.12 show examples of users' gestures for controlling the basic "straight", "split", "converge", "expand", "contract", and "twist" movements of the crowd, and the trajectories for the crowd's motion generated by our system. It can be seen from these figures that our system generates crowd motions that accurately reflect the different user gesture types. In each case the trajectories produced by the new *crowd motion primitive* maintain the same style as in the original crowd motion examples (Figure 6.1). In particular, the primitive produced for the "twist" gesture creates a distribution of the trajectories that maintain the concentric circles style seen in the original motion (Figure 6.12(b)). Furthermore, the motions produced for the "split" and "converge" examples show the subgrouping of agent trajectories where the agents move directly towards or away from one another (Figure 6.10(a) & Figure 6.11(a)), whilst the "expand" and "contract" trajectories maintain the radial movement observed in the original data (Figure 6.10(b) & Figure 6.11(b)).

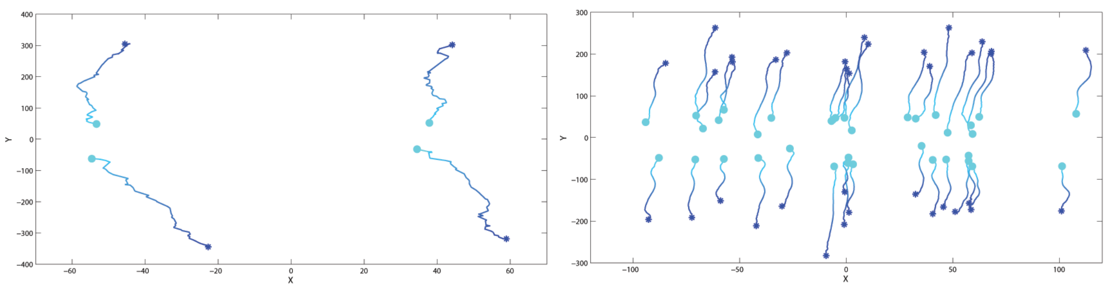


(a) A user's "converge" input (left) and the generated motion (right)

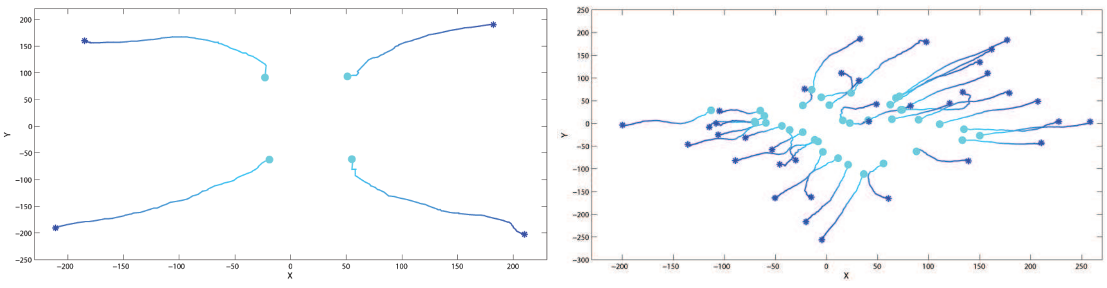


(b) A user's "contract" input (left) and the generated motion (right)

Figure 6.10: Gestures provided for basic "converge" and "contract" crowd motion types and the new motion generated by the current system. The light blue colour indicates the start of the input/motion and the dark blue colour indicates the end.



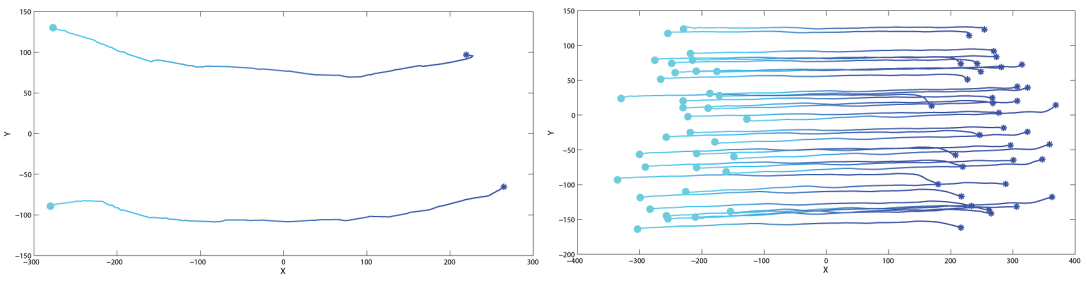
(a) A user's "split" input (left) and the generated motion (right)



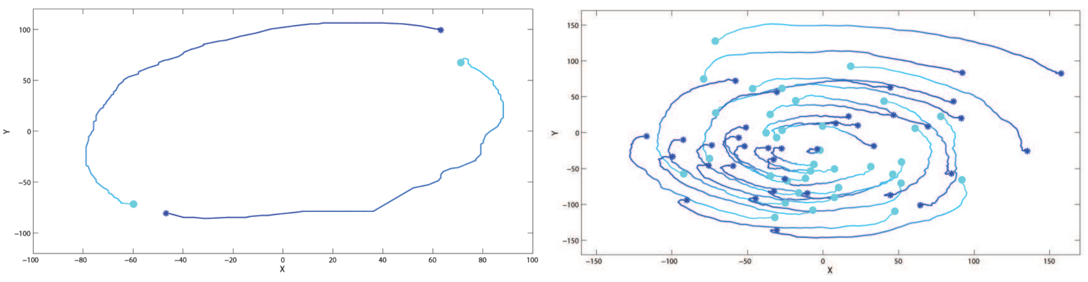
(b) A user's "expand" input (left) and the generated motion (right)

Figure 6.11: Gestures provided for basic "split" and "expand" crowd motion types and the new motion generated by the current system. The light blue colour indicates the start of the input/motion and the dark blue colour indicates the end.

The user gesture provided for the "expand" motion exhibits this radial property, where the user's touch inputs move away from the centre of the gesture (Figure 6.11(b)). This is different to the "split" gesture, that sees the user's inputs move directly away from one another (Figure 6.11(a)). Our system is able to distinguish between these different user input styles and produce the appropriate crowd motion.



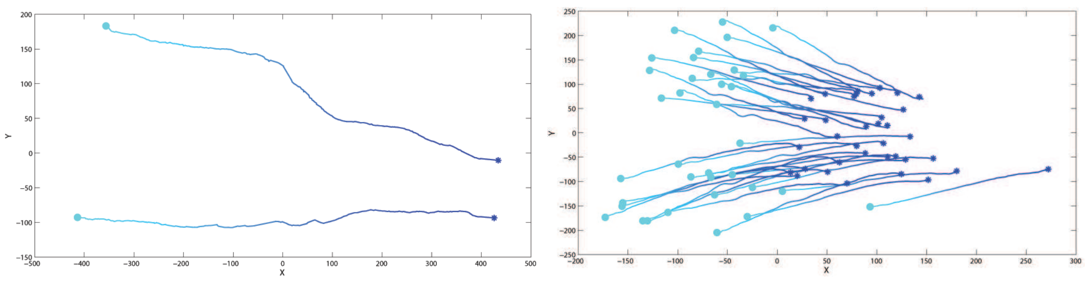
(a) A user's "straight" input (left) and the generated motion (right)



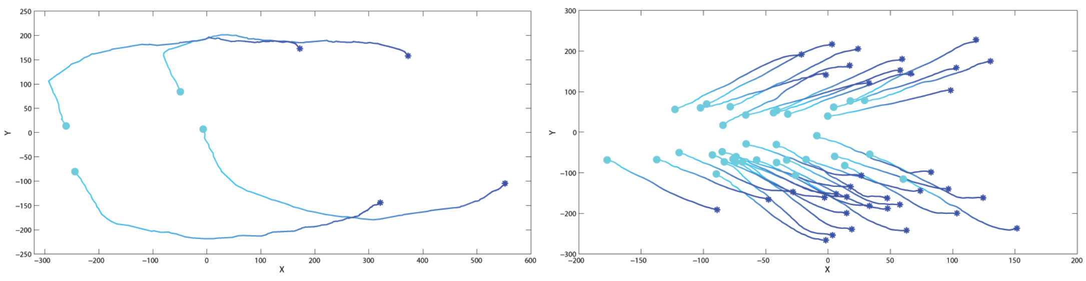
(b) A user's "twist" input (left) and the generated motion (right)

Figure 6.12: Gestures provided for basic "straight" and "twist" crowd motion types and the new motion generated by the current system. The light blue colour indicates the start of the input/motion and the dark blue colour indicates the end.

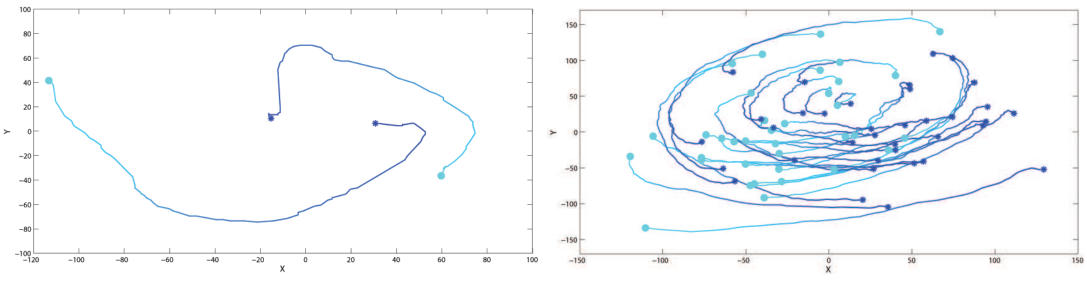
We also tested our system using a set of hybrid gestures, where a user provided input for controlling the "split & move", "converge & move", "twist & expand", and "twist & contract" examples seen in Figure 6.2. Figure 6.13 shows examples of users' gestures for these hybrid motions and the corresponding crowd trajectories produced by our system. It should be noted that these gestures were provided as controls for producing the crowd motions in Figure 6.2 but these motion examples do not form part of the *crowd motion space* described in Section 6.5; the motions that are produced here are types that are not present in our system's dataset. Here we can see that the generated motion displays the same properties present in each of the user gesture examples. The "converge & move straight" and "split & move straight" gestures both show movement of the user's touch inputs from left to right whilst simultaneously coming together or moving apart (Figure 6.13(a) & (b)).



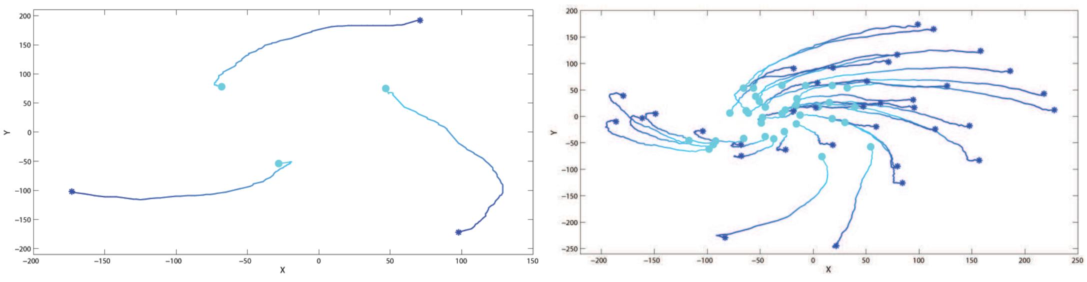
(a) A user's "converge & move" input (left) and the generated motion (right)



(b) A user's "split & move" input (left) and the generated motion (right)



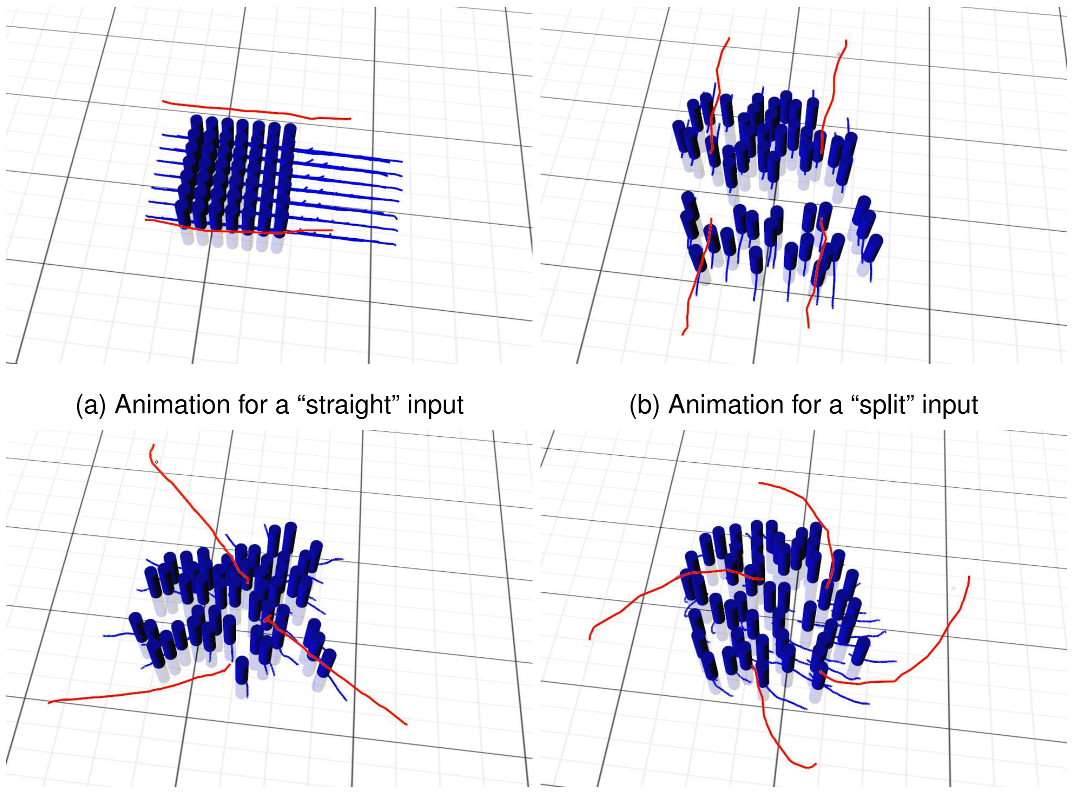
(c) A user's "twist & contract" input (left) and the generated motion (right)



(d) A user's "twist & expand" input (left) and the generated motion (right)

Figure 6.13: User gestures provided for hybrid crowd motion types and the new motion generated by the current system. The light blue colour indicates the start of the input/motion and the dark blue colour indicates the end.

The sets of trajectories generated as a result of these gestures show how our system can produce appropriate crowd motion in response to seeing these properties in a user's input. We can see a similar effect when a user performs a gesture with simultaneous twisting and contracting/expanding properties (Figure 6.13(c) & (d)). We have also produced animation of characters in response to a user's input to show the final result of our system (Figure 6.14). It can be seen that the generated crowd motion captures the features seen in the user's input. In all cases, the number of touch inputs provided for a gesture does not affect our system's ability to produce appropriate crowd motion in response to a user's input.


(a) Animation for a "straight" input

(b) Animation for a "split" input

(c) Animation for an "expand" input

(d) Animation for a "twist & expand" input

Figure 6.14: Examples of animated character motions produced from various user input gestures. The user input is shown by the red lines and the movement of the characters in the scene is shown by blue lines.

6.6.4 Computational Costs and 3D Rendering

The final animations produced in figure 6.14 are run on one core of a Core i7 2.67GHz CPU with 1GB of memory. For the multi-touch input we used a G4 multitouch overlay from PQ labs, attached to a 24" Acer S240HL LCD monitor. For the data set described in section 6.3 the computation of the features and the neighbourhood in *gesture space* for a new user gesture, the creation of a new *crowd motion primitive* from this information, and the sampling and application of the new trajectories to the crowd containing

50 characters in the scene are all done in real-time at a rate of $\sim$40 frames per second. We did however find that there was a slight slow down in framerate when computing a new crowd motion. The average time for our algorithm to convert a new user gesture was around 330 milliseconds. We can see from a breakdown of timings for different stages of the algorithm that the calculation of the nearest neighbours, which averaged around 300 milliseconds, is the main cause for this increased computation time. The creation of the *crowd motion primitive* accounts for approximately 12 milliseconds on average and the generation and assigning of the trajectories to the crowd takes about 4 milliseconds. In the current work we used a naive nearest neighbours search, computing the distance of the new gesture from all of the examples in our data set and then sorting them to find the nearest neighbours. The time taken for this stage could be improved by using better data structures for determining the nearest-neighbours at runtime.

6.7 Discussion

There are a number of key components to the system presented in this chapter. Firstly, the approach's ability to handle a variety of user control signals by accounting for individual variations when performing the same gesture type. Secondly, the effectiveness of our system's mapping from a novel user input to a corresponding crowd motion, including our system's ability to classify a user's gesture as part of this process. Finally, the generation of a new crowd motion based on the properties extracted from the user's input and its appropriateness for use in crowd animation. Here we discuss the effectiveness of our method for such tasks and propose some areas for future improvements.

6.7.1 Handling User Variations in Gestures

The approach presented in this chapter allows a user to provide control of a crowd's motion without the need to perform the input on a predefined control structure, such as the intermediate mesh used in chapters 3 & 5. This frees the user from having to control the crowd using a method that might require awkward placement of their fingers or hands, instead allowing the user to interact with the touch device in a way that is more comfortable for them. However, the removal of these restrictions makes the control of multiple crowds more difficult when using this method i.e. by not interacting with a specific control structure it is less obvious which crowd the control signal is intended for. This could be resolved by selecting the agents that are controlled using the positioning of the user's fingers when they are first placed onto the touch device.

The user-defined controls for crowd motion collected in the current work (Section 6.3) show consistency across all users in the style of gesture for each of the different crowd motion types. However, there are still variations in the number of fingers used to perform a gesture and the absolute placement of the fingers can be different for each user. For example, in the gestures for the "straight" crowd motion, some users placed their thumb and index finger that formed a line with one another when performing the gesture, while other users also placed their middle finger on the touch device, forming a triangle shape. Despite this variation, the overall movement of the user's fingers was similar in the two gestures. Since our system uses features that are invariant to these properties it can recognise the correct gesture type when presented with examples using both of these approaches.

Unfortunately, the invariance of our presented features to user differences in input does simplify the kinds of gestures that can be recognised by the current system. For example, to control a group of agents to pass between two other groups, a user may perform a gesture on the touch device that passes the index finger on one hand between the index and thumb of the other. To capture this "threading" of fingers on the touch device our set of features would need to be adapted to consider the spatial relationship between the touch inputs in a user's gesture. Furthermore, the features presented in Section 6.4.2 are not as effective when a user performs a gesture with two hands placed far apart from one another. If the user were to move the fingers on an individual hand towards themselves, the distance to centroid feature would not be able to highlight this because the centroid in this case is between the position of the two hands and the change in distance would be small. This situation could be resolved by first determining the existence of a two-handed gesture in a preprocessing step and then calculating the feature for each hand separately.

Our experimental results have shown that our system can produce appropriate, novel crowd motion based on real user input (Section 6.5.2). This suggests that our method can handle a variety of user inputs specified in a way that was intuitive to each of the users involved in our data collection phase (Section 6.3). Despite this, in the future it would be appropriate to conduct a user study to evaluate our system's ability to handle various user control styles.

6.7.2 Recognising User Gestures

Our experimental results show that our proposed set of gesture features can classify different types of user gesture well (Section 6.6.2). These features are a key part of our method and are particularly important for determining a gesture's neighbourhood in the *gesture space* along with the set of weights that we use to map from a new gesture to the final crowd motion. The crowd motions produced by our system show that our gesture features are effective for identifying the basic styles of user gesture by themselves and also when they are applied in combination (Section 6.6.3).

Lü & Li (2013) present a set of features based on translation, rotation, and scaling of a user's finger configurations that are similar to some of those presented in the current work. While their work uses these features to form a state machine for recognition of a predefined gesture, we use them to define the relationship between a new user gesture and the gesture examples in our dataset. Furthermore, we show that our minimum oriented bounding box feature is effective at distinguishing between a user's split/converge and expand/contract control signals. This is something that is not possible using only the set of features provided by Lü & Li.

Our current set of features is based on the change over time to properties of the position of a user's touch inputs. We would like to add to the kinds of gestures that can be used with our system by incorporating velocity information into our set of features. This would allow a user to control variations in the speed of the agents' movement as well as the overall crowd motion. We would also like to incorporate "intra-class" variations when processing a user's gesture. This sort of variation could include, for example, the dimensions of a user's gesture for controlling a wide or narrow "straight" motion or any of the other crowd motion types specified in section 6.3.

6.7.3 Generating Crowd Motion

In order to map from a user gesture to a crowd motion our method relies on the implicit relationship between the pairs of crowd motion examples and the gesture examples collected from users (Section 6.3). A more explicit relationship might have been established by using the eigenvectors of the user's input and those of the corresponding motion. However, performing PCA to get the eigenvectors for such data was not plausible due to the low number of trajectories in each data example (in most cases users tended to use 2 or 4 fingers to provide a gesture). Despite this, the implicit mapping provided by the user gesture and crowd motion pairs works well in our system. The many-to-one relationship between the set of user-defined gestures and the crowd motion data examples is well suited for our use of KNN to define the mapping between the *gesture space* and the *crowd motion space*.

In the current work we presented a *crowd motion primitive* as a model for crowd motion data. Our experiments have shown that new crowd motion can be generated from such a model whilst maintaining the underlying properties of the original crowd motion data (Section 6.6.3). Ju *et al.* (2010) presented a method for generating *Morphable Crowds* based on data examples of different styles of crowd motion. While their method is based on modelling the positions of characters surrounding an individual in a crowd motion, our method models the full trajectories of characters in the crowd.

An alternative, naïve approach to our method for generating crowd motion could be to replicate the user's input directly for each of the agents in the crowd. We argue instead that there are a number of advantages to using the approach presented here. Firstly, the signal on touch input devices can be very noisy, and it is possible that the input IDs for different fingers can temporarily swap, causing the trajectories to cross over or even be lost. Directly applying these trajectories to agents in the crowd would lead to undesirable motion and poor quality animation. By converting from a user gesture to a *crowd motion primitive*, our system acts as a filter for this signal noise, preventing it from affecting the final animation. Secondly, by using a model of the crowd motion we prevent the implicit constraints of a user's touch input from altering the final crowd motion and are able to maintain the characteristics of the original motion data. For example, when a crowd "twists" their motion forms concentric circles, with the agents toward the middle of the crowd moving in smaller circles than those on the outside. To specify such motion directly would require a large number of simultaneous touch inputs that would be awkward for a user to provide. Alternatively, if we were to simply replicate the user's finger movements this type of motion would not be reliably reproduced. By using our crowd motion models we can see that this property of the crowd movement is maintained. Finally, it would not be easy to combine different crowd motion styles if raw data was used, especially if the data examples contained different numbers of agents in the crowd. Using *crowd motion primitives* we can combine multiple models to generate hybrid motions in a straightforward manner and to produce trajectories for various size crowds (Section 6.5.2).

The use of a Gaussian mixture model (GMM) to represent the distribution of the individual trajectory data in our *crowd motion primitives* is effective for capturing the clustering of trajectories in the original data. We chose to use two Gaussian components in our GMM as this enabled us to model the various kinds of crowd motion we used in our experiments. As a user is only able to use two hands to provide a gesture to our system this number of components seems appropriate for the current application. Since our method is sampling from a model's GMM in order to produce new crowd trajectories it is possible in some circumstances that our model cannot reproduce the original motion data effectively. This can happen if a crowd motion example contains a large imbalance in the size of groups in the data e.g. separately clustered motion of a very large group and a very small group of characters. In this case, because sampling is probabilistic, the trajectories shown by the small group of characters are less likely to be drawn from the distribution represented by the model's GMM.

Although our approach is able to generate crowd motion styles that are not present in the original data set (Section 6.6.3) this ability is limited by the examples that the system is provided with. Currently, due to ease of data collection, these crowd motion examples are created by the system presented in chapters 3 & 5. A *crowd motion primitive* is general enough to be used with different types of crowd motion data that could be generated from another crowd animation system. Future work would also consider the use of motion data examples taken from real crowds.

6.8 Summary

In this chapter we have proposed an alternative data-driven approach to our method presented in Chapter 3 for control of crowd motion using a multitouch device. In contrast to our previous method, the approach presented in this chapter does not place as many restrictions on how a user is able to interact with the touch device, instead allowing a user to interact naturally to provide their input gesture. Our approach uses the properties of the user input instead of an intermediate control mesh to generate a corresponding crowd motion, enabling a user to more directly specify the movement of the crowd.

In this work we use input control and crowd motion pairs collected from real users to define the concept of a *gesture space* and its associated *crowd motion space*. The implicit correspondence between the pairs of gesture and crowd motion data is used to define a mapping between these two spaces. Given a new user input gesture, we can use its similarity to examples in our dataset and this implicit mapping to create a new crowd motion that is appropriate to the user's control. To produce novel crowd motion we define a model of crowd movement that we term a *crowd motion primitive*.

A *crowd motion primitive* is able to generate crowd motion for an arbitrary number of characters that maintains the properties of the original motion data. We show how several primitives can be combined to create new styles of crowd motion.

Our experimental results show that our proposed gesture features are expressive enough to capture different properties of a user's multitouch input, and mean that our system can distinguish between various user control gestures. We tested our system with inputs of the same type as those in our user gesture examples as well as for styles not seen in the original data set. The resulting crowd trajectories show how our system can produce crowd motion that corresponds well to a user's input gesture.

Chapter 7

Conclusion

The main goal of this thesis was to provide a method for realtime, intuitive control of the motion of a crowd in interactive applications. Such control is important for efficient creation of crowd simulations and for providing enjoyable interactive experiences for users. Previous research approaches require a lot of time and input to specify crowd motion. These methods do not provide a user with the ability to directly, and continuously express the desired crowd motion, instead expecting them to apply constraints to existing crowd motion and/or to define intermediate crowd configurations to dictate their movement. Furthermore, these approaches do not consider the direct interaction between characters in the crowd and their surrounding environment. This thesis proposes the use of a multitouch device for user input in order to solve these problems. We develop algorithms for controlling the high-level movement of a crowd in complex environments using a touch device and methods for handling the low-level interactions between the crowd and the environment. We provide experimental results to evaluate the effectiveness of our approaches.

7.1 Findings and Contributions

This section provides an overview of the methods presented in this thesis and highlights the contributions arising from this work.

To resolve the issue of a lack of single-step, interactive approaches for control of crowds in previous research, we have presented a method for realtime specification of a crowd's motion in a constrained environment using a multitouch device. In our approach a user can alter the movement of the crowd through a set of simple gestures that affect the crowd's formation at different levels of detail. The subtle movements of the user's fingers impact on the overall shape of the crowd and how each character interacts with the environment. Our approach combines the user-defined movement of the crowd with the restrictions placed on their formation by the environment to specify the final crowd motion. This allows the user to focus on the design of high level movements, while leaving the fine details to the system (Chapter 3).

To enable control of a crowd in complex environments we propose the use of a mass transport solver to allow characters to reassign their goal position in a user-defined crowd formation. This reassignment is achieved by minimising the total cost for all the characters to reach their goal locations. This enables a crowd to track the user's control well by accounting for perturbations in the individual characters' movement caused by the environment (Chapter 3).

We carried out an evaluation of the usability and intuitiveness of our proposed multitouch crowd control framework via a user study. We measured the performance of users ability to control a crowd to carry out a task using our multitouch method and a traditional mouse-based control scheme and provide a comparison of the results (Chapter 4). This comparison indicated that a multitouch device is effective for control over crowd simulation. Our multitouch crowd control framework showed reduce times for task completion in most scenarios and showed high scores when rated by users for ease of use, ability to complete the given task, and the feeling that they are able to control the crowd's behaviour. The results of our study highlighted a limitation of our approach for defining the crowd's movement, particularly in terms of the restrictions implied by our use of a mesh for user control.

We describe an environment-aware cost metric for the movement of a character in environments containing traversable obstacles. The proposed metric uses information from motion capture data to account for the effect of interacting with traversable obstacles on the cost of a character's path. This enables effective path planning to be performed in a wider variety of virtual environments and generates realistic biasing of an agent's movement through areas of more easily traversable terrain. By incorporating this metric into the mass transport solver as part of the multitouch control scheme in chapter 3 a user is able to control large crowds to move in complex environments and simulate crowd motion involving interesting character-environment interactions (Chapter 5). We introduce a feedback loop to adjust the user's control signal based on the current state of the characters in the crowd. This creates a better coupling between the user's commands and the actual movement of the crowd in the simulation, resulting in better formation tracking and a more appropriate final animation (Chapter 5).

To overcome the limitations of our approach in chapters 3 & 5 we presented an alternative data-driven method for inferring appropriate crowd motion based on a user's input signal. We use input controls and crowd motion pairs collected from real users to define the concept of a *gesture space* and its associated *crowd motion space*. Our algorithm uses a mapping defined between these spaces to generate new crowd motion that corresponds to a user's input gesture. We present a set of features that are invariant to variability in users' preferred touch input style. This allows a user to interact naturally with the touch device to provide their input gesture. We show how these features can be used for recognising different properties of a user's multitouch input, allowing our system to distinguish between a variety of control signals. We define a model of crowd movement called a *crowd motion primitive* that allows the final crowd motion to be generated for any number of characters and to combine crowd motion types to create new styles of crowd motion not seen in the original data set (Chapter 6).

7.2 Limitations and Future Research Directions

7.2.1 Combining Flexible Control with Environment Interactions

One future direction for this research is to incorporate obstacle avoidance into the motion generated by our method in chapter 6. Currently the trajectories that are produced do not consider the existence of obstacles around the crowd. One approach would be to combine the generated trajectories with the planning approach seen in chapters 3 & 5, where characters will loosely follow the path closest to them but only do so if they are not blocked by an obstacle. A better approach would be to incorporate information on the environment into the initial generation of the trajectories. New examples of crowds performing manoeuvres in constrained local environments could be added to our crowd motion database. We could store the *crowd motion primitives* generated by these examples along with a description of the crowd's surrounding environment. This would result in a set of models that can be used in different environmental situations that the crowd is in. At runtime the current state of the crowd, including the positions of the agents and the configuration of any surrounding obstacles, could be matched with crowd-environment models in the database. Given a model that matches well to the crowd's state, the method could sample new trajectories for the crowd that is appropriate to their current situation.

7.2.2 Controlling Group Interactions

The work presented in this thesis considers the high-level control of a single crowd that can interact at a low-level with static and dynamic environments. A future extension of this approach would be to simulate multiple crowds, where individual characters can interact with one another. This kind of simulation is important in sports games for example, where often there is a high-level strategy for the team but also there are low-level conflicts between the characters. In this situation the high-level control scheme developed in chapter 3 could be used to specify the formation of the team. Our solution for assigning character positions in the formation could be extended to take into account the global formation objective as well as a local objective of interacting with a nearby character. In this way a character can still respect the command of the user but be able to place themselves well to counteract the opponent crowd.

7.2.3 User Control Over Low-Level Character Interactions

In chapter 5 we presented a method for characters in a crowd to interact with the environment whilst obeying the controls of a user. While this method works to simulate these interactions, they occur passively and the user does not have control over the kind of interaction characters should perform. In some cases a character can interact with an object in different ways. For example, when confronted with a low hanging horizontal pole a character could choose to jump over it or to crawl under it. In other cases a character might encounter a small obstacle where it would be better to sidestep the object rather than to interact with it. This is also the case when groups are interacting with one another. A user may wish for the characters they are controlling to dodge the characters in the other crowd or to directly engage them. Currently the user does not have the ability to make this decision because the interactions are fixed in data patches that are embedded in the environment. It might be possible to use information from the user's gesture to bias the kind of action that might be performed in these situations. For example, if a user's control is quite fast then characters might be more aggressive in their interactions with other characters.

7.2.4 Controlling Non-Planar Crowd Movement & Particle Systems

The approaches presented in this thesis consider control of crowd movement on a two-dimensional plane. There is also a need to control groups of characters in three-dimensional space, for instance controlling a flock of birds or a school of fish. In these cases the movement of a crowd could be considered as a three-dimensional particle system; a concept that is ubiquitous in computer graphics where it is used for simulation of fire and fluids. A future research direction would be to explore how the control schemes presented in this thesis might apply to such systems. A multitouch device offers effective control over two-dimensional motion but is not sufficient for expressing such 3D control. Other devices, such as Microsoft's Kinect, allow a user to interact in this space and future work would explore the use of such devices for this kind of control. Although it would be necessary to adapt our method to three-dimensional interaction, the solutions for simultaneous formation and motion control and low-level character-environment interactions presented in this thesis could still apply. For instance, the use of the mass transport solver and our environment-aware cost metric in chapters 3 & 5 are still important for handling perturbations in the crowd's movement such that they can follow the user's control effectively. Furthermore, our data-driven approach in chapter 6 provides a general framework for a system to allow a user to control crowd simulation in an intuitive way.

7.3 Publications and Acknowledgements:

The concepts related to multitouch formation control in Chapter 3 are included in:

- Henry, Joseph, Shum, Hubert P. H., & Komura, Taku. 2012. Environment-aware realtime crowd control. *Pages 193-200 of: Proceedings of the 11th ACM SIGGRAPH Eurographics conference on Computer Animation*. EUROSCA'12. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

The work related to the evaluation of our multitouch control scheme in Chapter 4 and the control of crowds involving character-environment interactions in Chapter 5 are included in:

- Henry, Joseph, Shum, Hubert P. H. & Komura, Taku. 2014. Interactive Formation Control in Complex Environments. *Visualization and Computer Graphics, IEEE Transactions on*, **20**(2), 211-222.

The final rendering of the full body character motion for the work in chapters 3 and 5 and figures 3.1 and 3.7 were produced by Hubert P.H. Shum. The motion capture data for the character-environment interactions in chapter 5 was provided by Myung Geol Choi.

Bibliography

Allain, Pierre, Courty, Nicolas, & Corpetti, Thomas. 2014. Optimal crowd editing. *Graphical Models*, **76**(1), 1–16.

Alonso-Mora, Javier, Breitenmoser, Andreas, Rufli, Martin, Siegwart, Roland, & Beardsley, Paul. 2011. Multi-robot system for artistic pattern formation. *Pages 4512–4517 of: 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Anderson, Matt, McDaniel, Eric, & Chenney, Stephen. 2003. Constrained animation of flocks. *Pages 286–297 of: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Anthony, Lisa, & Wobbrock, Jacob O. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. *Pages 245–252 of: Proceedings of Graphics Interface 2010*. GI '10. Toronto, Ont., Canada, Canada: Canadian Information Processing Society.

Arkin, Ronald C. 1992. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, **9(3)**(3), 351–364.

Balch, T., & Hybinette, M. 2000. Social potentials for scalable multi-robot formations. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 73–80.

Balch, Tucker, & Arkin, Ronald C. 1998. Behavior-based formation control for multi-robot teams. *Robotics and Automation, IEEE Transactions on*, **14**(6), 926–939.

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

Berndt, Donald J., & Clifford, James. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. *Pages 359–370 of:* Fayyad, Usama M., & Uthurusamy, Ramasamy (eds), *KDD Workshop*. AAAI Press.

Bishop, Christopher M. 1996. *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press.

Bonneel, Nicolas, van de Panne, Michiel, Paris, Sylvain, & Heidrich, Wolfgang. 2011. Displacement interpolation using Lagrangian mass transport. *Proceedings of the 2011 SIGGRAPH Asia Conference on - SA '11*.

Bouvier, Eric, & Guilloteau, Pascal. 1996. Crowd Simulation in Immersive Space Management. *Pages 104–110 of: Proceedings of the Eurographics Workshop on Virtual Environments and Scientific Visualization '96*. London, UK, UK: Springer-Verlag.

Braun, Adriana, Musse, Soraia Raupp, de Oliveira, Luiz Paulo Luna, & Bodmann, Bardo EJ. 2003. Modeling individual behaviors in crowd simulation. *Pages 143–148 of: Computer Animation and Social Agents, 2003. 16th International Conference on*. IEEE.

Brogan, David C, & Hodgins, Jessica K. 1997. Group behaviors for systems with significant dynamics. *Autonomous Robots*, **4**(1), 137–153.

Chang, Jen-Yao, & Li, Tsai-Yen. 2007. Simulating crowd motion with shape preference and fuzzy rules. *In: Proceedings of International Symposium on Artificial Life and Robotics (AROB2007)*. Citeseer.

Cheney, Ward, & Kincaid, David R. 2008. *Linear Algebra: Theory and Applications*. 1st edn. USA: Jones and Bartlett Publishers, Inc.

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

Choi, Myung Geol, Kim, Manmyung, Hyun, Kyung Lyul, & Lee, Jehee. 2011. Deformable Motion: Squeezing into Cluttered Environments. *Computer Graphics Forum*, **30**(2), 445–453.

Clements, Richard R, & Hughes, Roger L. 2004. Mathematical modelling of a mediaeval battle: the Battle of Agincourt, 1415. *Mathematics and Computers in Simulation*, **64**(2), 259–269.

Courty, Nicolas, & Corpetti, Thomas. 2007. Crowd motion capture. *Computer Animation and Virtual Worlds*, **18**(4-5), 361–370.

Desai, J. P., Ostrowski, J. P., & Kumar, V. 2001. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, **17(6)**, 905–908.

Fiorini, P., & Shiller, Z. 1998. Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research*, **17**(7), 760–772.

Floater, Michael S. 2003. Mean value coordinates. *Computer Aided Geometric Design*, **20**(1), 19–27.

Funge, John, Tu, Xiaoyuan, & Terzopoulos, Demetri. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. *Pages 29–38 of: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co.

Geraerts, R., & Overmars, M. H. 2007. The Corridor Map Method: A general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds*, **18(2)**, 107–119.

Golas, Abhinav, Narain, Rahul, & Lin, Ming. 2013. Hybrid long-range collision avoidance for crowd simulation. *Pages 29–36 of: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '13*. ACM Press.

Goldenstein, Siome, Karavelas, Menelaos, Metaxas, Dimitris, Guibas, Leonidas, Aaron, Eric, & Goswami, Ambarish. 2001. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, **25**(6), 983–998.

Gu, Qin, & Deng, Zhigang. 2011a. Context-aware motion diversification for crowd simulation. *Computer Graphics and Applications, IEEE*, **31**(5), 54–65.

Gu, Qin, & Deng, Zhigang. 2011b. Formation sketching: an approach to stylize groups in crowd simulation. *Pages 1–8 of: Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society.

Gu, Qin, & Deng, Zhigang. 2013. Generating Freestyle Group Formations in Agent-Based Crowd Simulations. *IEEE Computer Graphics and Applications*, **33**(1), 20–31.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., & Dubey, P. 2009. ClearPath: highly parallel collision avoidance for multi-agent simulation. *Pages 177–187 of: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '09.*

Guy, Stephen J, Chhugani, Jatin, Curtis, Sean, Dubey, Pradeep, Lin, Ming, & Manocha, Dinesh. 2010. Pledestrians: a least-effort approach to crowd simulation. *Pages 119–128 of: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association.

Heigeas, Laure, Luciani, Annie, Thollot, Joëlle, & Castagné, Nicolas. 2003. A Physically-Based Particle Model of Emergent Crowd Behaviors. *In: Graphicon.*

Helbing, D. 1992. A fluid-dynamic model for the movement of pedestrians. *Complex Systems*, **6**, 391–415.

Helbing, Dirk, & Molnar, Peter. 1995. Social force model for pedestrian dynamics. *Physical review E*, **51**(5), 4282.

Helbing, Dirk, Farkas, Illes, & Vicsek, Tamas. 2000. Simulating dynamical features of escape panic. *Nature*, **407**(6803), 487–490.

Helbing, Dirk, Molnár, Péter, Farkas, Illés J, & Bolay, Kai. 2001. Self-organizing pedestrian movement. *Environ. Plann. B*, **28**(3), 361–383.

Helbing, Dirk, Buzna, Lubos, Johansson, Anders, & Werner, Torsten. 2005. Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions. *Transportation Science*, **39**(1), 1–24.

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

Henry, Joseph, Shum, Hubert P. H., & Komura, Taku. 2012. Environment-aware real-time crowd control. *Pages 193–200 of: Proceedings of the 11th ACM SIGGRAPH / Eurographics conference on Computer Animation*. EUROSCA'12. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Henry, Joseph, Shum, Hubert P. H., & Komura, Taku. 2014. Interactive Formation Control in Complex Environments. *Visualization and Computer Graphics, IEEE Transactions on*, **20**(2), 211–222.

Ho, Choon Sing, Nguyen, Quang Huy, Ong, Yew-Soon, & Chen, Xianshun. 2010. Autonomous multi-agents in flexible flock formation. *Pages 375–385 of: Motion in Games*. Springer.

Ho, Choon Sing, Ong, Yew-Soon, Chen, Xianshun, & Tan, Ah-Hwee. 2012. FAME, soft flock formation control for collective behavior studies and rapid games development. *Pages 258–269 of: Simulated Evolution and Learning*. Springer.

Hongwan, L., Wai, F.K., & Chor, C.H. 2003. *A study of pedestrian flow using fluid dynamics*. Tech. rept. National University of Singapore.

Hoogendoorn, S., & Daamen, W. 2005. Self-Organization in Pedestrian Flow. *Traffic and Granular Flow '03*, 373–382.

Hoogendoorn, Serge, & Bovy, Piet. 2000. Gas-Kinetic Modeling and Simulation of Pedestrian Flows. *Transportation Research Record*, **1710**(1), 28–36.

Hoogendoorn, S.P., & Bovy, P.H.L. 2004. Pedestrian route-choice and activity scheduling theory and models. *Transportation Research Part B: Methodological*, **38**(2), 169–190.

Hughes, Roger L. 2002. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, **36**(6), 507–535.

Hughes, Roger L. 2003. The flow of human crowds. *Annual review of fluid mechanics*, **35**(1), 169–182.

Hyun, K, Kim, M, Hwang, Y, & Lee, J. 2013. Tiling Motion Patches. *IEEE transactions on visualization and computer graphics*.

Igarashi, Takeo, Moscovich, Tomer, & Hughes, John F. 2005. As-rigid-as-possible shape manipulation. *Pages 1134–1141 of: ACM Transactions on Graphics (TOG)*, vol. 24. ACM.

Jiang, H., Xu, W., Mao, T., Li, C., Xia, S., & Wang, Z. 2010. Continuum crowd simulation in complex environments. *Computers & Graphics*, **34(5)**, 537–544.

Jiang, Yingying, Tian, Feng, Zhang, Xiaolong, Liu, Wei, Dai, Guozhong, & Wang, Hongan. 2012. Unistroke Gestures on Multi-touch Interaction: Supporting Flexible Touches with Key Stroke Extraction. *Pages 85–88 of: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*. IUI '12. New York, NY, USA: ACM.

Jin, Xiaogang, Xu, Jiayi, Wang, Charlie CL, Huang, Shengsheng, & Zhang, Jun. 2008. Interactive control of large-crowd navigation in virtual environments using vector fields. *Computer Graphics and Applications, IEEE*, **28**(6), 37–46.

Jordao, Kevin, Pettré, Julien, Christie, Marc, & Cani, Marie-Paule. 2014. Crowd Sculpting: A Space-time Sculpting Method for Populating Virtual Environments. *Computer Graphics Forum*.

Ju, Eunjung, Choi, Myung Geol, Park, Minji, Lee, Jehee, Lee, Kang Hoon, & Takahashi, Shigeo. 2010. Morphable crowds. *ACM Transactions on Graphics (TOG)*, **29**(6), 140.

Kamphuis, A., & Overmars, M. H. 2004. Finding paths for coherent groups using clearance. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '04*.

Kanyuk, Paul. 2009. Brain Springs: Fast Physics for Large Crowds in WALL E. *Computer Graphics and Applications, IEEE*, **29**(4), 19–25.

Karamouzas, Ioannis, & Overmars, Mark. 2010. Simulating the local behaviour of small pedestrian groups. *Pages 183–190 of: Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*. ACM.

Karamouzas, Ioannis, Geraerts, Roland, & van der Stappen, A. Frank. 2013. Space-Time Group Motion Planning. *Algorithmic Foundations of Robotics X*, **86**, 227–243.

Kato, Jun, Sakamoto, Daisuke, Inami, Masahiko, & Igarashi, Takeo. 2009. Multi-touch interface for controlling multiple mobile robots. *Pages 3443–3448 of: CHI'09 Extended Abstracts on Human Factors in Computing Systems*. ACM.

Kim, Jongmin, Seol, Yeongho, Kwon, Taesoo, & Lee, Jehee. 2014. Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics*, **33**(4), 1–10.

Kim, Manmyung, Hyun, Kyunglyul, Kim, Jongmin, & Lee, Jehee. 2009. Synchronized multi-character motion editing. *Page 79 of: ACM Transactions on Graphics (TOG)*, vol. 28. ACM.

Kim, Sujeong, Guy, Stephen J., & Manocha, Dinesh. 2013. Velocity-based modeling of physical interactions in multi-agent simulations. *Pages 125–133 of: Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '13*. ACM Press.

Kin, Kenrick, Hartmann, Björn, DeRose, Tony, & Agrawala, Maneesh. 2012. Proton: Multitouch Gestures As Regular Expressions. *Pages 2885–2894 of: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. New York, NY, USA: ACM.

Klotsman, Marina, & Tal, Ayellet. 2012. Animation of flocks flying in line formations. *Artificial Life*, **18**(1), 91–105.

Koenig, S., & Likhachev, M. 2002. Improved fast replanning for robot navigation in unknown terrain. *Proceedings 2002 IEEE International Conference on Robotics and Automation - ICRA '02*, **1**, 968–975.

Kwon, Taesoo, Lee, Kang Hoon, Lee, Jehee, & Takahashi, Shigeo. 2008. Group motion editing. *Page 80 of: ACM Transactions on Graphics (TOG)*, vol. 27. ACM.

Lai, Yu-Chi, Chenney, Stephen, & Fan, ShaoHua. 2005. Group motion graphs. *Pages 281–290 of: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM.

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

Lau, Manfred, & Kuffner, James J. 2005. Behavior planning for character animation. *Pages 271–280 of: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM.

Lau, Manfred, & Kuffner, James J. 2006. Precomputed search trees: planning for interactive goal-driven animation. *Pages 299–308 of: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '06*.

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*. Eurographics Association.

Lee, Yongjoon, Wampler, Kevin, Bernstein, Gilbert, Popović, Jovan, & Popović, Zoran. 2014. Motion fields for interactive character locomotion. *Communications of the ACM*, **57**(6), 101–108.

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. *Computer Graphics Forum*, **31**(2pt2), 489–498.

Lerner, Alon, Chrysanthou, Yiorgos, & Lischinski, Dani. 2007. Crowds by example. *Pages 655–664 of: Computer Graphics Forum*, vol. 26. Wiley Online Library.

Lewis, M Anthony, & Tan, Kar-Han. 1997. High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, **4**(4), 387–403.

Li, Norman H.M., & Liu, Hugh H.T. 2008. Formation UAV flight control using virtual structure and motion synchronization. *Pages 1782–1787 of: American Control Conference, 2008*. IEEE.

Li, Tsai-Yen, & Chou, Hsu-Chi. 2003. Motion planning for a crowd of robots. *Pages 4215–4221 of: IEEE International Conference on Robotics and Automation, 2003 - ICRA '03*. IEEE.

Li, Y., Christie, M., Siret, O., Kulpa, R., & Pettré, J. 2012. Cloning crowd motions. *Pages 201–210 of: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '12*.

Lipman, Yaron, Levin, David, & Cohen-Or, Daniel. 2008. Green Coordinates. *ACM Trans. Graph.*, **27**(3), 78:1–78:10.

Loscos, C., Marchal, D., & Meyer, A. 2003. Intuitive Crowd Behaviour in Dense Urban Environments using Local Laws. *Page 122 of: Proceedings of the Theory and Practice of Computer Graphics 2003 - TPCG '03*.

Løvås, Gunnar G. 1994. Modeling and simulation of pedestrian traffic flow. *Transportation Research Part B: Methodological*, **28**(6), 429–443.

Low, Chang Boon, & San Ng, Quee. 2011. A flexible virtual structure formation keeping control for fixed-wing UAVs. *Pages 621–626 of: 9th IEEE International Conference on Control and Automation (ICCA), 2011*. IEEE.

Lü, Hao, & Li, Yang. 2013. Gesture studio: authoring multi-touch interactions through demonstration and declaration. *Pages 257–266 of: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.

Mataric, M. 1993. Designing emergent behaviors: From local interactions to collective intelligence. *Pages 432–441 of: Proceedings of the International Conference on Simulation of Adaptive Behaviour: From Animals to Animats*, vol. 2.

Mataric, M.J. 1992. Minimizing complexity in controlling a mobile robot population. *Pages 830–835 of: Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE Comput. Soc. Press.

McNamara, Antoine, Treuille, Adrien, Popović, Zoran, & Stam, Jos. 2004. Fluid control using the adjoint method. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2004*, **23(3)**, 449–456.

Mehrjerdi, Hasan, Ghommam, Jawhar, & Saad, Maarouf. 2011. Nonlinear coordination control for a group of mobile robots using a virtual structure. *Mechatronics*, **21**(7), 1147–1155.

Micire, Mark, Desai, Munjal, Courtemanche, Amanda, Tsui, Katherine M, & Yanco, Holly A. 2009. Analysis of natural gestures for controlling robot teams on multi-touch tabletop surfaces. *Pages 41–48 of: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ACM.

Milazzo, Joseph, Rouphail, Nagui, Hummer, Joseph, & Allen, D. 1998. Effect of Pedestrians on Capacity of Signalized Intersections. *Transportation Research Record*, **1646**(1), 37–46.

Min, Jianyuan, & Chai, Jinxiang. 2012. Motion graphs++. *ACM Transactions on Graphics*, **31**(6), 1.

Min, Jianyuan, Chen, Yen-Lin, & Chai, Jinxiang. 2009. Interactive generation of human animation with deformable motion models. *ACM Transactions on Graphics*, **29**(1), 1–12.

Moussaïd, M., Helbing, D., Garnier, S., Johansson, A., Combe, M., & Theraulaz, G. 2009. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society B: Biological Sciences*, **276**(1668), 2755–2762.

Moussaïd, M., Perozo, N., Garnier, D., Helbing, D., & Theraulaz, G. 2010. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. *PLoS ONE*, **5(4)**, e10047.

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

Musse, S.R., & Thalmann, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, **7**(2), 152–164.

Musse, S.R., Jung, C.R., Jacques Jr, J., & Braun, A. 2007. Using computer vision to simulate the motion of virtual agents. *Computer Animation and Virtual Worlds*, **18(2)**, 83–93.

Narain, Rahul, Golas, Abhinav, Curtis, Sean, & Lin, Ming C. 2009. Aggregate dynamics for dense crowd simulation. *Page 122 of: ACM Transactions 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. *ACM Transactions on Graphics (TOG)*, **29**(4), 123.

Paris, Sébastien, Pettré, Julien, & Donikian, Stéphane. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Pages 665–674 of: Computer Graphics Forum*, vol. 26. Wiley Online Library.

Park, Min Je. 2010. Guiding flows for controlling crowds. *The Visual Computer*, **26**(11), 1383–1391.

Patil, Sachin, Van Den Berg, Jur, Curtis, Sean, Lin, Ming C, & Manocha, Dinesh. 2011. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, **17**(2), 244–254.

Pelechano, Nuria, Allbeck, Jan M, & Badler, Norman I. 2007. Controlling individual agents in high-density crowd simulation. *Pages 99–108 of: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Pelechano, Nuria, Allbeck, Jan M., & Badler, Norman I. 2008. Virtual Crowds: Methods, Simulation, and Control. *Synthesis Lectures on Computer Graphics and Animation*, **3**(1), 176.

Peng, Jufeng, & Akella, Srinivas. 2005. Coordinating Multiple Robots with Kinodynamic Constraints Along Specified Paths. *The International Journal of Robotics Research*, **24(4)**, 295–310.

Peters, Christopher, & Ennis, Cathy. 2009. Modeling Groups of Plausible Virtual Pedestrians. *IEEE Computer Graphics and Applications*, **29(4)**, 54–63.

Pettré, Julien, Laumond, Jean-Paul, & Thalmann, Daniel. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *Pages 81–90 of: First International Workshop on Crowd Simulation*.

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.

Poonawala, Hasan A, Satici, Aykut C, & Spong, Mark W. 2013. Leader-follower formation control of nonholonomic wheeled mobile robots using only position measurements. *Pages 1–6 of: 2013 9th Asian Control Conference (ASCC)*. IEEE.

Qingge, J. I., & Can, G. A. O. 2007. Simulating crowd evacuation with a leader-follower model. *IJCSES International Journal of Computer Sciences and Engineering Systems*, **1(4)**, 249–252.

Qiu, Fasheng, & Hu, Xiaolin. 2010. Modeling group structures in pedestrian crowd simulation. *Simulation Modelling Practice and Theory*, **18**(2), 190–205.

Rekik, Yosra, Grisoni, Laurent, & Roussel, Nicolas. 2013. Towards Many Gestures to One Command: A User Study for Tabletops. *Pages 246–263 of:* Kotz, Paula, Marsden, Gary, Lindgaard, Gitte, Wesson, Janet, & Winckler, Marco (eds), *INTERACT (2)*. Lecture Notes in Computer Science, vol. 8118. Springer.

Rekik, Yosra, Vatavu, Radu-Daniel, & Grisoni, Laurent. 2014. Match-up & conquer. *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces - AVI '14*, 201–208.

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

Reynolds, Craig W. 1987. Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH 1987*.

Rubine, Dean. 1991. Specifying Gestures by Example. *SIGGRAPH Comput. Graph.*, **25**(4), 329–337.

Rubner, Yossi, Tomasi, Carlo, & Guibas, Leonidas J. 1998. A metric for distributions with applications to image databases. *Pages 59–66 of: Sixth International Conference on Computer Vision, 1998*. IEEE.

Sakuma, Takeshi, Mukai, Tomohiko, & Kuriyama, Shigeru. 2005. Psychological model for animating crowded pedestrians. *Computer Animation and Virtual Worlds*, **16**(3-4), 343–351.

Sandler, Seth. 2008 (August). *Community Core Vision Software*. `http://ccv.nuigroup.com/`.

Sandler, Seth. 2010 (November). *Multitouch Mini*. `http://sethsandler.com/multitouch/mtmini/`.

Schmidt, Ryan. 2009 (February). *As-Rigid-As-Possible 2D Shape Manipulation Demo*. `http://www.dgp.toronto.edu/~rms/software/Deform2D/`.

Schwartz, J. T., & Sharir, M. 1983. On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers. *The International Journal of Robotics Research*, **2**(3), 46–75.

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

Shum, Hubert PH, Komura, Taku, Shiraishi, Masashi, & Yamazaki, Shuntaro. 2008. Interaction patches for multi-character animation. *Page 114 of: ACM Transactions on Graphics (TOG)*, vol. 27. ACM.

Simeon, T., Leroy, S., & Lauumond, J. P. 2002. Path coordination for multiple mobile robots: a resolution-complete algorithm. *IEEE Transactions on Robotics and Automation*, **18(1)**, 42–49.

Singh, Shawn, Kapadia, Mubbasir, Hewlett, Billy, Reinman, Glenn, & Faloutsos, Petros. 2011. A modular framework for adaptive agent-based steering. *Pages 141–150 of: Symposium on Interactive 3D Graphics and Games - I3D '11*. ACM Press.

Snape, Jamie, van den Berg, Jur, Guy, Stephen J., & Manocha, Dinesh. 2009. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. *Pages 5917–5922 of: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.

Sorkine, Olga, Cohen-Or, Daniel, Lipman, Yaron, Alexa, Marc, Rössl, Christian, & Seidel, H-P. 2004. Laplacian surface editing. *Pages 175–184 of: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM.

Spry, Stephen, & Hedrick, J Karl. 2004. Formation control using generalized coordinates. *Pages 2441–2446 of: CDC. 43rd IEEE Conference on Decision and Control, 2004.*, vol. 3. IEEE.

Sud, A., Andersen, E., Curtis, S., Lin, M.C., & Manocha, D. 2008. Real-Time Path Planning in Dynamic Virtual Environments Using Multiagent Navigation Graphs. *IEEE Transactions on Visualization and Computer Graphics*, **14**(3), 526–538.

Sung, Mankyu, Gleicher, Michael, & Chenney, Stephen. 2004. Scalable behaviors for crowd simulation. *Pages 519–528 of: Computer Graphics Forum*, vol. 23. Wiley Online Library.

Sung, Mankyu, Kovar, Lucas, & Gleicher, Michael. 2005. Fast and accurate goal-directed motion synthesis for crowds. *Pages 291–300 of: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM.

Takahashi, Shigeo, Yoshida, Kenichi, Kwon, Taesoo, Lee, Kang Hoon, Lee, Jehee, & Shin, Sung Yong. 2009. Spectral-Based Group Formation Control. *Pages 639–648 of: Computer Graphics Forum*, vol. 28. Wiley Online Library.

Teknomo, Kardi. 2006. Application of microscopic pedestrian simulation model. *Transportation Research Part F: Traffic Psychology and Behaviour*, **9**(1), 15–27.

Thalmann, Daniel, & Musse, Soraia Raupp. 2013. Crowd Simulation.

Toussaint, Godfried. 1983. *Solving geometric problems with the rotating calipers*.

Treuille, Adrien, Cooper, Seth, & Popović, Zoran. 2006. Continuum crowds. *Pages 1160–1168 of: ACM Transactions on Graphics (TOG)*, vol. 25. ACM.

Tsitsiklis, John N. 1995. Efficient algorithms for globally optimal trajectories. *Automatic Control, IEEE Transactions on*, **40**(9), 1528–1538.

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

van den Akker, M., Geraerts, R., Hoogeveen, H., & Prins, C. 2010. Path planning for groups using column generation. *Pages 94–105 of: Proceedings of the Third international conference on Motion in games - MIG '10*.

van den Berg, J.P., & Overmars, M.H. 2005. Prioritized motion planning for multiple robots. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 430–435.

van den Berg, Jur, Patil, Sachin, Sewall, Jason, Manocha, Dinesh, & Lin, Ming. 2008a. 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 den Berg, Jur, Lin, Ming C., & Manocha, Dinesh. 2008b. Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation. *Pages 1928–1935 of: IEEE International Conference on Robotics and Automation*. IEEE.

van den Berg, Jur, Guy, S. J., Lin, M. C., & Manocha, D. 2009. Reciprocal n-body collision avoidance. *Pages 3–19 of: International Symposium of Robotics Research*.

Vatavu, Radu-Daniel, Anthony, Lisa, & Wobbrock, Jacob O. 2012. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. *Pages 273–280 of: Proceedings of the 14th ACM International Conference on Multimodal Interaction*. ICMI '12. New York, NY, USA: ACM.

Wang, P.K.C. 1989. Navigation Strategies For Multiple Autonomous Mobile Robots Moving In Formation. *Journal of Robotic Systems*, **8(2)**, 177–195.

Wang, Yanzhen, Xu, Kai, Xiong, Yueshan, & Cheng, Zhi-Quan. 2008. 2D shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds*, **19**(3-4), 411–420.

Weng, Yanlin, Xu, Weiwei, Wu, Yanchen, Zhou, Kun, & Guo, Baining. 2006. 2D shape deformation using nonlinear least squares optimization. *The Visual Computer*, **22**(9-11), 653–660.

Wilkie, D., van den Berg, J., & Manocha, D. 2009. Generalized Velocity Obstacles. *Pages 5573–5578 of: IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.

Wobbrock, Jacob O., Wilson, Andrew D., & Li, Yang. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. *Pages 159–168 of: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. UIST '07. New York, NY, USA: ACM.

Wojtan, Chris, Mucha, Peter J, & Turk, Greg. 2006. Keyframe control of complex particle systems using the adjoint method. *Pages 15–23 of: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Wolinski, D., Guy, S. J., Olivier, A. H., Lin, M., Manocha, D., & Pettré, J. 2014. Parameter estimation and comparative evaluation of crowd simulations. *Computer Graphics Forum*, **33**(2), 303–312.

Xu, Jiayi, Jin, Xiaogang, Yu, Yizhou, Shen, Tian, & Zhou, Mingdong. 2008. Shape-constrained flock animation. *Computer Animation and Virtual Worlds*, **19**(3-4), 319–330.

Xu, M., Wu, Y., Ye, Y., Farkas, I., Jiang, H., & Deng, Z. 2014. Collective Crowd Formation Transform with Mutual Information based Runtime Feedback. *Computer Graphics Forum*, **(accepted)**.

Xu, Mingliang, Wu, Yunpeng, & Ye, Yangdong. 2012. Smooth and efficient crowd transformation. *Proceedings of the 20th ACM international conference on Multimedia - MM '12*, 1189–1192.

Yang, Wenwu, Feng, Jieqing, & Wang, Xun. 2012. Structure Preserving Manipulation and Interpolation for Multi-element 2D Shapes. *Computer Graphics Forum*, **31**(7), 2249–2258.

Yeh, H., Curtis, S., Patil, S., van den Berg, J., Manocha, D., & Lin, M. 2008. Composite Agents. *Pages 39–47 of: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

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*. ACM.

Yu, Qinxin, & Terzopoulos, Demetri. 2007. A decision network framework for the behavioral animation of virtual humans. *Pages 119–128 of: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Zheng, Liping, Zhao, Jianming, Cheng, Yajun, Chen, Haibo, Liu, Xiaoping, & Wang, Wenping. 2014. Geometry-constrained crowd formation animation. *Computers & Graphics*, **38**(Feb), 268–276.