Simulating Interactions Among Multiple Characters

Hubert P. H. Shum

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2010

Abstract

In this thesis, we attack a challenging problem in the field of character animation: synthesizing interactions among multiple virtual characters in real-time. Although there are heavy demands in the gaming and animation industries, no systemic solution has been proposed due to the difficulties to model the complex behaviors of the characters.

We represent the continuous interactions among characters as a discrete Markov Decision Process, and design a general objective function to evaluate the immediate rewards of launching an action. By applying game theory such as tree expansion and min-max search, the optimal actions that benefit the character the most in the future are selected. The simulated characters can interact competitively while achieving the requests from animators cooperatively.

Since the interactions between two characters depend on a lot of criteria, it is difficult to exhaustively precompute the optimal actions for all variations of these criteria. We design an off-policy approach that samples and precomputes only meaningful interactions. With the precomputed policy, the optimal movements under different situations can be evaluated in real-time.

To simulate the interactions for a large number of characters with minimal computational overhead, we propose a method to precompute short durations of interactions between two characters as connectable patches. The patches are concatenated spatially to generate interactions with multiple characters, and temporally to generate longer interactions. Based on the optional instructions given by the animators, our system automatically applies concatenations to create a huge scene of interacting crowd.

We demonstrate our system by creating scenes with high quality interactions. On one hand, our algorithm can automatically generate artistic scenes of interactions such as the fighting scenes in movies that involve hundreds of characters. On the other hand, it can create controllable, intelligent characters that interact with the opponents for real-time applications such as 3D computer games.

Acknowledgements

I would like to thank Dr. Taku Komura for his constant support on my research career. During the last seven years, we spent countless days and nights discussing on our researches. His creativity is one of the many things I have benefited a lot, and he is one of the few professors who really work with students until midnight whenever we have a tight deadline.

I would also like to thank my teammates. I wish to thank to Dr. Shuntaro Yamazaki for giving me numerous advices since the beginning of the interaction project. Many thanks to Edmond Ho for all the sharing in both researches and living during these difficult years, especially when I first arrived at Edinburgh. I must thank Masashi Shiraishi who works with me on the interaction project. It surely would be tough without him when working in those winter nights. I have to thank Ludovic Hoyet for working with me on the reinforcement learning project. Though his stay in Edinburgh is short, he brought our team a lot of joys and memories. Thank Pranjul Yadav for his hard work on the angular momentum project. It is a pity that he did not pursue his research career in Edinburgh. Finally, I wish to thank my new teammates Adam Barnett and He Wang, who shared the ups and downs during our studies.

I have to thank the generous supports of the IPAB. I must thank Dr. Sethu Vijayakumar for his advices during my PhD study. I also wish to thank Ms. Irene Madison and Ms. Jane Teplechuk for helping me with the tedious documentations that would take me ages to fill in.

Finally, I wish to thank Dr. Franck Multon and Dr. Subramanian Ramamoorthy as my examiners. Their suggestions and comments help a lot to improve this thesis.

This project was partly supported by a funding from RIKEN, a CERG grant from the Research Grant Council of Hong Kong, the Special Coordination Funds for Promoting Science and Technology from Ministry of Education, Culture, Sports, Science and Technology of Japan, and the Initiating Knowledge Transfer Fund from the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Hubert P. H. Shum*)

To my beloved wife,
who shares my joys and tears for each of my paper submissions,
and takes care of me throughout my doctoral study,
as well as my whole life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Character animation has become a popular research area due to the demands in the gaming and movie industries. For examples, three dimensional cartoons like “Monster, Inc.” and “The Incredibles” use a lot of human-like characters to carry out the story line. It is very common for these characters to interact with the environments or even with other characters in the movie. While the audience focus a lot on funny character models and realistic rendering effects, the movements of the characters, especially the interactions among them, give the souls to the character and implicitly catch the attention of the audience. Imagine watching “The Incredibles” or ”The Lord of the Rings” without any fighting scenes. The movies would become uninteresting and monotonic.

Although interactions among characters are popular, the industries still rely on systems that require heavy manual works, making the production of such scenes time consuming and costly. Recent researches propose a lot of algorithms to synthesize the movements of a single character, but only a few of them focus on the problem of the interactions among multiple characters because of the complexity of the problems. In this chapter, we briefly review some of the technologies used in producing character animations, and point out why it is difficult to generate interactions among multiple characters. We will then suggest the three major problems we are going to solve in this thesis, and give an overview on our researches.

1.1 Demands for Character Interactions

In movies like “Final Fantasy: The Spirits Within” and “The Polar Express”, we can see a lot of high quality human characters. A key aspect to generate high quality animations is the movements of the characters. For example, when a character is interacting with the environment, such as sitting on a chair in a moving train, the movements of the body must be natural and obey the laws of physics. Although it is possible to manually design the movements of the characters, which is known as keyframe animation, it requires skilled animators to create natural and realistic motions. Therefore, in the movies stated above, motion capture techniques are used. Instead of creating motions using mouse and keyboard, the animator capture the movements using three dimensional motion capturing devices. The motion data are then imported to computer graphics software and rendered with character models. The major advantage of applying motion capturing techniques to create character animations is that the motion are always natural as they are performed by real human. Furthermore, the intrinsic physical rules are always satisfied. For example, we can capture an actor lifting a heavy object and use them to control characters in a movie. We will find that the character keeps the body balanced when lifting since the motions are captured.

The interaction among characters is one of the most important movements in a scene. In games like “NBA Live” and “Winning Eleven”, players enjoys controlling characters to fight with or play sports games with computer controlled characters. In movies, high quality fighting scenes like those in “300” always catch the attention of the audience. Although motion capturing is effective to generate realistic human motions, it is still a relatively new technique and has a lot of limitations. One major problem is the difficulty to capture several actors interacting with each other at the same time. If we use the popular optical motion capture system, the actors will be occluded from the cameras, and hence parts of the body segments will not be captured. On the other hand, if we use magnetic or mechanical motion capture system, the actors need to wear bulky and heavy devices on their bodies, which will seriously affect their interactions. Furthermore, dense interactions such as fighting have a high chance to damage the capturing devices. Therefore, instead of capturing the motion data and displaying them directly, people try to reuse the movements captured by a single character. A typical approach is to create a motion database containing short motion clips captured by a single character. Then, the animators give high level instructions for creating the scene, and the system plan the motions of the characters such that they move as if they are interacting with each other.

There are some researches on generating intelligent characters to achieve high level goals automatically, such as running around based on the animators' design. However, few of them can generate characters that interact with other characters. The reason is that when creating a single character to explore or interact with an environment, everything is static except the character itself, and hence planning can be easily done. Once the planning is finished, the character simply carries out the plan since nothing unexpected would happen. However, when we wish two characters interacting with each other, both of them are dynamic to the opponent and would move around. Planning becomes more complex since we need to take into account the movement of the opponents. Due to such complexity, the interactions we can find in games are usually in low quality, while those in movies rely a lot on manual designs. A system that can automatically simulate high quality interactions is needed.

There are several important requirements for a good interaction simulation system. First, high quality interactions require intelligent characters that interact with their opponents as if they are real humans. For example, in the fighting scene of "The Matrix", characters are not simply standing still and being hit. Instead, they avoid opponents' punches and counter attack. To simulate high quality competitive interactions, each character has to acts as an individual and plan for its own benefits in order to be "smart". Second, real-time applications like computer games require the simulation to be completed with minimal computational cost. In games like "Tekken", the players control their characters to fight with computer controlled characters in real-time. While there are faster processors and better graphics cards, we need better algorithms to finish the processes in such a short time limit. Third, recent computer animations tend to use a huge number of characters in the scene. In movies like "The Lord of the Rings", there are thousands of background characters fighting with each other. In such scenes, we wish each character to interact with several of its neighbors at once with minimal manual processing. Current researches lack a framework to handle such kind of crowd interactions due to the large amount of information that needs to be considered. In this thesis, we will propose new algorithms to address these requirements.

In general, the technology we have to generate scenes of interactions among multiple characters is far behind from what we need in the industries. The reason of such lack of researches is due to the difficulty and complexity to control the characters when they are interacting with each other. This thesis proposes a complete framework to solve the problem, and hence generate high quality scenes up to the industrial standards.

1.2 Problems Definition and Methodology Overview

In this section, we define the three major problems we are going to solve in this thesis, and give an overview of our methodology to tackle the problems.

1.2.1 Simulating Interactions from Singly Captured Motions

While interactions are the key elements to create high quality character animations, it is difficult to produce such animations by keyframing because the movements of a character would affect another. Also, capturing multiple characters interacting with each other with motion capturing devices is almost impossible due to the limitation of the technologies. We need an algorithm to produce scene of dense interactions effectively by capturing the motion of a single actor only. The characters must behave intelligently as if they are real humans, and the animator should be able to control the scenes.

1.2.1.1 Our Method

We first capture the motions of a single actor for easy and effective capturing. The captured motions are rendered during run-time with virtual characters. We prefer to capture long sequence of motions to preserve the naturalness of the motions. Then, we segment the motions automatically into semantic actions, which are used as entities during the interaction synthesis processes. When displaying the actions, instead of simply displaying them in the order of capturing, we can reorder them to create different behaviors. For example, for a set of boxing actions, we can reorganize the actions such that the character alternatively punches and kicks. Furthermore, we can create two characters and tell them to perform a series of actions such that they appear to be interacting. Hence, the problem to simulate interactions can be considered as the problem to find out the right combination of actions for the two characters. More details can be found in Chapter 3.

A smart character interacting with its opponents must not simply perform the actions that cause the best immediate benefit. Rather, it should make careful plan and select the actions that benefit itself the most in the future. For example, during boxing, a simple attack may cause minimal benefit as it will be blocked easily by the opponent. Instead, it may be wiser to wait for the opponent to punch first, avoid such a punch and counter attack while the opponent's arms are unable to defence. To implement such intelligent, we apply artificial intelligence algorithms used to simulate computer players in chess games. We design an algorithm called temporal tree expansion, which uses the concept of game tree to evaluate the long term benefits of performing an action. With the game tree, we predict the future states of interactions after launching the possible choices of actions. We also predict what actions the opponent may perform to counteract in the future, assuming that the opponent tries to perform the best actions. By this way, we can select the actions that benefit the character the most in the future considering the possible reactions from the opponent. The game tree expansion approach can simulate realistic interactions such as fighting and chasing among a few characters. By changing the way to evaluate the benefits, we can adjust the behaviors of the characters, such as simulating a character that wish to run away from its opponent. We can simulate smarter characters that consider the further future, and less intelligent characters that consider only immediate benefits, by adjusting the size of the game tree. To control the movements of the characters, the animator can give high level commands that are evaluated during the tree expansion process. More details can be found in Chapter 4.

1.2.2 Precomputing Interactions for Real-Time Applications

Real-time applications like computer games, similar to movies, often involve character interactions. There are tons of fighting games and sport games that require dense interactions of multiple characters. The major focus of character interactions in games, apart from motion quality, is the computational cost. Since the characters in games must act in real-time, it is challenging to simulate high quality interactions in such a short time limit. We wish to dramatically decrease the computational cost for simulating interactions between two characters. Although simplifying the problem and creating interactions of lower quality may be a solution, we believe that it is possible to generate high quality interactions even in real-time by applying precomputation techniques.

1.2.2.1 Our Method

While the game tree approach can simulate high quality interactions, the most critical problem is that it is very computational costly. Simulating a few seconds of interactions would take several minutes. Thus, it cannot be applied in real-time applications like computer games. In most simulation systems, the optimal choice of actions depends on limited criteria, such as the distance between the two characters, their relative orientations, and the actions performing by them. In theory, it is possible to list out all the possible variations of these criteria, which are known as states, and precompute the optimal actions for each of them. However, there are too many states and precomputing them all will require an unreasonably long time and large memory. Fortunately, we find that although there are a lot of states, the characters mostly stay in a small subset of these states during interactions. For example, during fighting, the characters always stay in a preferred distance with respect to the opponent, and face the enemy during punches and kicks in order to hit the opponent. Hence, we can simply precompute the optimal actions for such small number of states that are relevant to the interactions. We propose a method to sample the useful states by evaluating their quality with respect to interactions. Then, we design a finite state machine called the Interaction Graph to organize the states, and hence precompute the optimal actions for each of them. With our method, high quality interactions can be simulated in real-time. Furthermore, the user can give high level control commands like those in computer games during run-time, and the system can select the optimal actions based on precomputed results immediately. More details can be found in Chapter 5.

1.2.3 Simulating the Interactions for a Crowd

We find that there is an increasing trend to use a huge number of characters in a scene like wars in the animation industry. In such scenes, there are two requirements. First, the characters must not only interact with a single opponent. Rather, we wish to see scenes in which multiple characters interacting at the same time. However, due to the complexity of interactions, it is very difficult to generate high quality interactions involving many characters. Second, computational cost to generate such interactions must be small, because we need to generate the interactions for tens or even hundreds of characters. We wish to have a system that can generate interactions within a crowd effectively.

1.2.3.1 Our Method

Although the Interaction Graph can simulate high quality interactions in real-time, due to the limitation in the complexity of the states, we can only simulate two characters interacting with each other. Even if we put many characters into the scene, they will only interact with one opponent at a time. This kind of one-to-one interaction will appear to be monotonic and unrealistic. Furthermore, the Interaction Graph generates a lot of nodes and edges, making it difficult to control the quality of the interactions and manually authoring the graph. We propose a method to combine one-to-one interactions to form many-to-many interactions for a large number of characters. First, we generate short duration of interactions between two characters using the temporal tree expansion approach and store the results in a structure called the Interaction Patch. We define a standard interfaces for the patches such that they can be concatenated to form longer interactions and interactions that involve more characters. During run-time, with a set of precomputed Interaction Patches, our system plan the best way to concatenate the patches with reference to optional users' preferences. The computational cost for concatenating the patches is very small, and hence we can simulate a scene with tens to hundreds of characters interacting with each other in real-time. We simulated scenes of crowd fighting, falling onto each other like dominos, playing American football, and helping each other to carry luggage. More details can be found in Chapter 6.

1.3 Thesis Structure

The structure of this thesis is as follows. First, we review the related researches in the field of character animations, and highlight those focusing on the interactions among characters in Chapter 2. We point out that there is a lack of researches on high-quality dense interactions. We then explain our methodology to capture motions from a single actor and simulate the interactions for multiple characters. We first talk about the processes to capture and process the motion data in Chapter 3. Then, we explain the framework to simulate dense interactions using artificial intelligence techniques called temporal tree expansion in Chapter 4. Although temporal tree expansion method can generate high quality interactions, it is computational costly and is not suitable for real-time processes. Thus, we propose a framework to precompute the interaction information into a data format called Interaction Graph in Chapter 5. Since the Interaction Graph can only generate one-to-one interactions, we propose the Interaction Patches that represent the interactions of two characters in a short duration in Chapter 6. The Interaction Patches can be precomptued and combined during run-time to create a huge scene of crowd interactions in real-time. Finally, we conclude the whole thesis in Chapter 7. Appendix A includes further information on fine tuning the movements of the characters and rendering them in the scene.

1.4 Summary

High quality interactions for multiple characters are important aspects in the movie and gaming industries. However, there is a lack of research due to the complexity of the problem. We indicated that the requirements of the research area can be divided into three problems. First, we have to simulate realistic interactions among characters. Second, the process has to be made real-time to suit the needs of computer games. Finally, we need a mechanism to create crowd interactions that involve tens to hundreds of characters. We gave an overview on the methodology we proposed. We proposed three ideas to deal with the problems, namely the temporal tree expansion, the Interaction Graph and the Interaction Patches. In the following chapters we will go into details for each of the ideas.

Chapter 2

Related Works

Motion synthesis has become one of the major research areas in computer graphics, robotics and biomechanics. Driven by the demand of computer games and movies, the field evolved rapidly in the past decade.

There are two main streams of techniques to simulate motions. On one hand, physical simulations, which are based on dynamics, model the motions with physical parameters such as force and angular momentum. Such simulations play a key role in controlling robots and analyzing human movements, mainly due to their accurate simulations of control forces. However, they are in general computational costly and are unable to guarantee realistic motions. More discussion can be found in Section 2.1. On the other hand, data-driven approaches, which are mainly based on kinematics, generate new motions with raw captured motions from motion capture system. Because human performed movements are used, it can simulate realistic motions with minimal computation. However, since the internal dynamics are not captured, the motions cannot be adapted to different environments nor react to external perturbations. Section 2.2 explains in details how data-driven approaches work.

While both streams of approaches can model and simulate the motion of a single character, it is unclear how these algorithms can be extended to model multiple interacting characters. In recent years, there are some new approaches to analyze the nature of interactions and apply them to control virtual characters. Most of these researches, however, can only produce limited interactions among characters, such as collision avoidance during walking. For those that can simulate denser interactions, they usually suffer from the complexity of interactions and produce only sub-optimal results. More information is available in Section

2.3.

2.1 Physical Simulation Approaches

There are two major advantages of physical simulation approaches. First, with a well defined physical model, they guarantee the physical correctness of the motion generated. This is important to generate control signals in robotics and evaluate human motions for medical or biological concerns. Second, because the motions are simulated based on dynamics, it is possible to adjust the motions by adding external perturbations to the characters. In computer games, characters are expected to react to dynamic environment and external forces. Physical simulations provide an excellent framework to simulate such behaviors.

We will discuss three popular areas of physical simulations in character animations. The proportional-derivative (Section 2.1.1) controller approximates the control signal as a weighted combination of errors. It is fast and effective to simulate simple human motions. Inverse dynamics (Section 2.1.2) is used to calculate the control forces required to perform a predefined motion. Once the control forces are optimized, the controller becomes independent to the reference motion. Spacetime constraints (Section 2.1.3) solve the body motions with multiple constraints by optimizing a set of objective functions, and guarantee the stability and optimality of the synthesized motion.

2.1.1 Proportional-Derivative Controller

Forward dynamics has been used extensively for gait simulations [Raibert & Hodgins (1991), Van De Panne (Mar 1996), Liu & Popović (2002)]. One of the popular implementations of these systems is the proportional-derivative (PD) controller. In PD control, the joint torque required is determined by comparing the current joint angle and angular velocity to that of the desired posture:

$$\tau = k_p(\theta - \theta_d) + k_d(\theta' - \theta'_d) \tag{2.1}$$

where τ is the torque applied to the joint, θ and θ' are the joint rotation and angular velocity respectively, θ_d and θ'_d are the target joint rotation and angular velocity respectively. k_p is called the proportional gain, which governs the response rate, and k_d is the derivative gain, which is used to reduce overshoot. With PD controller, one can simulate the torque required for a character to perform different motions.

The PD control system can be extended further to simulate walk-to-run and run-to-walk transition [Hodgins (9-11 Apr 1991), Shiratori & Hodgins (2008)], as well as walking with different step length [Hodgins & Raibert (Jun 1991)]. Usually a finite state machine is used to connect multiple PD controllers and handle the transition between controllers. Apart from locomotion, athletic motions such as cycling and handspring vaulting [Hodgins et al. (1995)], and somersault motion [Playter & Raibert (7-10 Jul 1992)] are generated. Such systems require a certain amount of manual design to generate different types of motions. PD control is also suitable to generate responsive motions such as the falling back motions when one is being pushed [Zordan & Hodgins (2002)]. Since generating realistic full body motions require carefully designed dynamics systems, it is proposed to control characters by simply tracking captured human movements, such that the movements can react to external forces [Abe & Popović (2006)]. For speeding up the simulations, simplified polygons model can be used to represent the characters with high degrees of freedom [Mitake et al. (2009)].

One problem of PD control is that the system overpowers the natural dynamics of the object. This is because the forces applied by PD control are not optimized, and tend to be larger than required. Recently, the gentle forces are proposed to control fluids and deformable solids with optimal precomputed gain for each time step [Barbič & Popović (2008)]. However, it is not confirmed if such an approach works well when applied to human body with high degrees of freedom.

2.1.2 Inverse Dynamics

When controlling characters in a physical environment, we need to calculate the control forces or torques to perform different actions. One approach is capturing the actions and applying PD control to estimate the control forces to be applied. However, systems based only on such feedback forces are usually unstable due to the high gains required. Instead, the control torques can be computed by inverse dynamics and applied in a feed-forward system to minimize to uses of feedback controls [Yin et al. (2007), Oshita & Makinouchi (2001)].

Quadratic programming is proposed to solve the torque optimization problem for simplified walking characters [da Silva et al. (2008)]. Although such optimization is computational costly given the high degree of freedom for a human body, it can be solved in real-time by considering only short horizon, and hence generate an interactive system [Marco da Silva (2008)]. Alternatively, optimizations in high dimensional space can be solved in a properly reduced low-dimensional space for faster and more robust results [Barbič et al. (2009)]. With a combined force and torque controller, realistic motion to restore balance when standing can be simulated [Adriano Macchietto (2009)]. By training a higher level controller based on stepping pattern, locomotion that satisfies different stepping constraints can be generated [Coros et al. (2008)]. One may also combine controller trained with different source motions to create a controllable character that is capable of balancing from external forces and switching locomotion styles [Sok et al. (2007), Tsai et al. (2009)].

2.1.3 Spacetime Constraints

Although frame-based motion editing methods such as PD control can generate control signals, they may result in unstable motions with jittery movements. In contrast, spacetime constraints [Witkin & Kass (1988)] are introduced to optimize a motion segment in a given duration. When using the method, animators specify multiple constraints, which are usually represented as a set of keyframe postures, and apply a solver to compute the optimal control torques by minimizing a predefined objective function based on dynamics. With spacetime constraints, realistic motion like running and jumping can be generated [Popović & Witkin (1999), Liu & Popović (2002), Liu et al. (2005)]. By applying non-linear controllers to plan through contact state changes, agile locomotion can be generated [Muico et al. (2009)]. The method is also used to generate transitions between motion segments [Rose et al. (1996)], and adapt previously created motions to new situations and characters [Gleicher & Litwinowicz (1998)].

Although spacetime constraints can simulate stable and realistic motions, they are very computationally costly due to the nonlinearity of the objective functions used during the optimization stage. Work has been done to enhance the performance by introducing hierarchical structure to the solver [Liu et al. (1994)], and simplifying the system to a linear time process [Fang & Pollard (2003)]. By combining spacetime constraints and motion displacement map, simulation can be conducted in interactive time [Gleicher (1997)]. However, since the general concept of these researches is to generate a motion that minimizes a predefined function, it is difficult to generate interactions for real-time application where the constraints are not known in advance. As a result, it is suggested to dynamically update the constraints in every time-step during run-time such that animation can be generated with user interactions [Jain et al. (2009)]. It is also possible to apply spacetime optimization in every time step to interactively synthesize the interactions between characters and objects [Jain & Liu (2009)].

2.2 Data-Driven Motion Synthesis

Despite of the extensive uses in robotics, physical simulation is less popular in character animations. One major concern is that physically correct motions may not always appear natural. This is because naturalness in human motion is difficult to be well represented with simple physical formulas. Another problem is that these approaches are, in general, computationally costly due to the high degrees of freedom of the human body. For real-time applications such as computer games, computational resources have to be distributed to different modulus such as rendering and non-player characters (NPC) controls. A full system of physical simulations may not be affordable.

On the other hand, with the improvement in motion capture technology, it becomes easier to acquire 3D human motions using motion capture systems. Recently, researchers have focused more on data-driven approaches, while applying physically constraints to enhance the captured motions. The data-driven approaches can be classified into motion interpolation (Section 2.2.1) and motion rearrangement (Section 2.2.2). The former interpolates motion segments to create new ones, and the latter synthesizes motion by rearranging motion segments in a motion database.

2.2.1 Motion Interpolation

With motion interpolation techniques, new motions can be synthesized by blending multiple captured source motion segments [Bruderlin & Williams (1995)]. Due to the ability to create new motions, such approaches usually do not require a large motion database, which is favorable to systems with limited storage like game consoles.

With the simple Alpha blending algorithm, two motions are blended linearly frame by frame as follow:

$$M_{blend}(f) = \alpha M_1(f) + (1-\alpha)M_2(f) \forall f \in [f_{t_0}, f_{t_f}] \tag{2.2}$$

where f_{t_0} and f_{t_f} represent the frame range for blending the two motion, $\alpha = \frac{1-f}{(f_{t_f} - f_{t_0})}$ is a value between 1 and 0, M_{blend}, M_1 and M_2 are the postures at the blended motion, first motion and second motion, respectively.

Since the source motions may not be synchronized in speed and duration, blending them frame by frame usually leads to unnatural behaviors. Dynamic time warping is introduced to minimize the difference between two motions by synchronizing them during blending [Kovar & Gleicher (2003), Hsu et al. (2005)]. The general idea is to evaluate the posture differences between two source motions in every possible combination of frames pairs. Then, dynamic programming is applied to select the frame pairs with minimal difference for blending. Apart from synchronizing the source motions with minimal posture difference, some researchers suggest to consider the foot supporting state to determine synchronization frames [Ménardais et al. (2004)]. Thus, multiple synchronization frames are defined to indicate the change in supporting states, and blending is performed within the periods where supporting states remain unchanged. By this way, the foot skate of the characters, which is a common artifact for motion blending, can be minimized. Similarly, for rhythmic motions such as dancing, the beat patterns of the motions could be considered to synchronize the source motions [Ménardais et al. (2004)].

Motions of different logical context, in general, cannot be blended together. For instance, blending a punching motion with a kicking motion will lead to an unnatural result. Therefore, it is suggested that source motions should be classified into groups and blending should only be applied within each group [Park et al. (2004)]. Similarly, locomotion can be classified into different step patterns before blending such that foot sliding can be avoided [Ikemoto et al. (2007)]. Since classifying motions requires a lot of manual work, automatic approaches are introduced [Kovar & Gleicher (2004), Mukai & Kuriyama (2005)].

Dimensionality reduction techniques such as Principal Component Analysis (PCA) or Scaled Gaussian Process Latent Variable Model (SGPLVM) can be applied to create a reduced space of high dimensional motions. Although interpolation is not performed explicitly, each point in the reduced space represents an implicitly blended posture from multiple sources, and a trajectory in the reduced space represents a blended motion of the source motion style [Brand & Hertzmann (2000), Grochow et al. (2004)]. Furthermore, although it is difficult to control a character based on blending in the high dimensional joint space, it is possible to solve the optimization problem in the reduced space, and then project the posture back to the joint space [Safonova et al. (2004), Chai & Hodgins (2005), Bitzer et al. (2008)]. In other words, dimensionality reduction techniques can be considered as statistical means to group similar motion segments and generate new motions by implicit blending.

2.2.2 Motion Rearrangement

With motion rearrangement, new motions are synthesized by rearranging shorter motion segments in a sequence of captured motion. The Motion Graph approach is an effective algorithm to organize captured motion and interactively reproduce continuous motions based on a graph structure [Kovar et al. (2002), Lee et al. (2002)]. The graph is automatically generated based on a set of motions, with the nodes representing poses in the captured motion and edges representing motion segments. To create the graph, the similarity between arbitrary poses is evaluated based on kinematical data such as the position and velocity of joints [Wang & Bodenheimer (2003)]. The similar pairs are connected with edges. By traversing the graph, it is possible to generate long sequences of motion.

Since the Motion Graph produces a lot of edges and nodes without any context, it becomes difficult to control the character based on the user wishes. Recently, to reduce the complexity of the Motion Graph, some researchers proposed to generate a simpler, hub-to-hub Motion Graph called Fat Graph [Gleicher et al. (2003), Shin & Oh (2006)]. In such a graph, every node represents multiple similar poses and every edge represents multiple similar motion segments. A similar approach is to apply the concept of motif, which is the pattern representation used in deoxyribonucleic acid (DNA), to group similar actions into a single edge [Beaudoin et al. (2008), Jingjing Meng & Wu (2008)]. With a simplified graph, animators can easily modify the structure of the graph and even add in desired motions. Another solution for organizing complex Motion Graphs is to integrate behavior contexts into the captured motions and generate a hierarchical structure of motion data [Lau & Kuffner (2005), Kwon & Shin (2005), Chiu et al. (2007)].

Usually, motion segments of similar logical behavior are classified into the same group as a node, while the edges indicate possible transition between the groups. Motion planning with such graphs is more efficient since it can be carried out in the behavior level instead of the motion data level.

It is possible to control a character using the Motion Graph by defining objective functions to select appropriate motion segments. Previous researches created controllable characters performing different tasks such as running and exploring a territory [Choi et al. (2003)]. The quality of the resultant motion and the controllability of the characters, in general, depend a lot on the connectivity of the Motion Graph [Reitsma & Pollard (2004, 2007)]. Hence, some researchers propose to enhance the connectivity by blending multiple motions [Zhao & Safonova (2008)]. Furthermore, since these kinds of planning usually require a lot of computational power, methods to precompute the optimal actions to be performed and store them in a look-up table are proposed [Lau & Kuffner (2006)]. During run-time, the motions of a large number of characters can be simulated in real-time by referencing the precomputed look-up table. Roughly speaking, for such precomputation techniques, memory requirement increases exponentially with respect to the complexity of the system. As a result, the bottleneck of the system shifts from computational power to memory usage.

Recently, hybrid systems that combine the advantage of motion rearrangement and motion blending by integrating blending parameters into a Motion Graph are proposed [Safonova & Hodgins (2007), Heck & Gleicher (2007), Safonova & Hodgins (2008)]. In such graphs, each node contains a set of source motions for blending, and the edges denote the ability to transit from node to node. As a result, while the Motion Graph provides natural transitions to generate long sequence of motions, realistic variations of the source motions can be generated by motion interpolation. Furthermore, by embedding blending information into optimization algorithm such as reinforcement learning, characters can be controlled in a continuous action space [Wan-Yen Lo (2008)].

While it is originally proposed to organize and rearrange human motion, Motion Graphs can also be applied to organize the motion of flocks of birds [Lai et al. (2005)] by considering the whole group as a single object. Some researchers apply the graph to organize 2D human motion video [Flagg et al. (2009)] with the consideration on pixel similarity. Also, Motion Graph can be used to organize the movements of complex polygon meshes [James et al. (2007)] by dividing the meshes into smaller parts and create edges for each part.

2.3 Motion Planning For Interactions

Although it is popular to apply motion synthesis techniques to control a single character, controlling multiple characters to interact with each other remains an open problem. Current researches on this topic can be classified into five categories. First, the crowd simulation methods (Section 2.3.1) are shown to be efficient to generate the behaviors of a crowd of characters. A critical problem is that these methods cannot be easily extended to handle crowd with complex interactions such as people pushing around. Second, response systems (Section 2.3.2) are strong at generating reactive behaviors by combining motion capture data with physical dynamics. The drawback is that interactions simulated by these methods are bound to be passive, such as characters being pushed or avoiding. The third category is to generate motions based on statistical analysis (Section 2.3.3). By analyzing the behaviors of real humans, one can simulate characters with similar behaviors. However, such kinds of statistics are, in general, difficult to be acquired and limited in variations. Optimization-based methods (Section 2.3.4) can synthesize the required motion by optimizing pre-defined objective functions. Approaches like reinforcement learning are effective in synthesizing interactive behaviors. Nevertheless, more researches are required to generate realistic interactions among large number of characters. Finally, topology based methods (Section 2.3.5) are good at generate close interactions between two characters with multiple keyframes, but it is unclear how these approaches can be integrated with artificial intelligence controllers.

2.3.1 Crowd Simulation

The objective of crowd simulation is to model and simulate a large number of characters in an environment. Although simple flocking model to simulate the movements of flocks of birds and schools of fishes has been proposed decades ago [Reynolds (1987, 1999)], it is until recently that more delicate approaches appear to model the movement of humans. The use of social force is suggested to describe the internal motivation of the characters to perform actions or movements [Helbing & Molnar (1995)]. Based on the concepts of social forces, a dynamical model is proposed to simulate the movement of people in panic [Helbing et al. (2000)]. Later on, a probabilistic model is introduced for the selections of motions [Sung et al. (2004)]. Fluid dynamics is also used to determine the flow of people moving in the space [Treuille et al. (2006)]. In such systems, the density of characters in the environment is evaluated to generate a potential field. The positions of the characters, and their corresponding movements, are then updated based on the field.

Procedural based approached is proposed to simulate crowd with different movement behaviors based on a set of predefined rules. A city can be modeled as a 2D map and the pedestrians are assigned with the check-points to navigate the city [Loscos et al. (2003)]. These check-points can be further classified as interest points, for which the characters must pass through, and action points, for which the characters would perform certain actions [Musse et al. (1998)]. Such systems are extended to control the movements of a group of characters rather than a single one by classifying characters into leaders and members. Only leaders make decisions on check-points and members follow their corresponding leaders [Musse & Thalmann (1997)]. Collision avoidance for the characters can be implemented by checking the future trajectories of movements and steering away from potential collisions [Feurtey (2000), Loscos et al. (2003)]. The concept of proxy agents, which are virtual, invisible characters that influence nearby characters, is introduced to simulate social effects in a crowd such as giving places to the elderly [Yeh et al. (2008)].

To enhance the scalability of the crowd simulation system, as well as decrease the run-time overhead, patches based methods are introduced. The patches, which represent short segments of motions in small areas, are in general precomputed, and combined during run-time to create a huge scene. The motion patches is proposed to simulate scenes such as office and playground, in which each patch defines the movements of a character to interact with an object in a small rectangular area [Lee et al. (2006)]. When combining the patches during run-time, a scenes in which a lot of characters interacting with different objects in the world can be generated. Based on a similar concept, the crowd patches are created to precompute the behaviors when multiple characters avoid each other when walking [Yersin et al. (2009)]. In our work, we also apply patch based method to create a crowd scene. However, our patches define the dense interactions among multiple characters rather than simple locomotion.

The controllability of the crowd also becomes an important research topic in crowd simulation. While procedural based approaches can adjust the behavior of individual characters, there is a strong demand in games and movies to control the crowd of characters as a whole, such as maintaining the formation and moving in a specific manner. A spectral-based approach is applied to interpolate the motion of the characters in between multiply keyframes of crowd formation [Takahashi et al. (2009)]. On the other hand, by representing the spatio-temporal relationship of a group of moving characters as a mesh, mesh based deformation techniques can be applied to edit the movement of the whole crowd while minimizing the adjustment required [Kwon, Lee, Lee & Takahashi (2008)]. A similar idea is proposed to edit a crowd of character with multiple constraints [Kim et al. (2009)] by borrowing the Laplacian transform in mesh editing that produce as-rigid-as possible transformation [Igarashi et al. (2005)]. In our experiments, we also implements interfaces to control the characters in different levels.

In these works, the interactions between the characters are rather simple, such as avoiding other pedestrians in a path, or walking along with another character. Although crowd simulation is efficient to compute the movement of a large number of characters, due to the simplification of characters' behaviors, it is difficult to be applied for creating interactions when close contacts such as pushing, pulling or hitting motions are involved. Unlike previous researches, our proposed algorithm can generate a huge number of characters while capable of simulating the dense interactions among them.

2.3.2 Response System

How a person behaves when being pushed, pulled or hit is recently attracting researchers due to the demands in animation systems such as video games. Since solving the body poses during impact by frame based optimization sometimes leads to unstable body movements, optimization with space time constraints are proposed to guarantee the stability of the motion throughout time [Liu et al. (2006)]. The drawback is that the computational cost is very large even for an off-line process due to the high degrees of freedom for a human body. As a result, local optimization based on dynamics [Yamane & Nakamura (2000)] is proposed to solve the body postures during impact in real-time [Abe et al. (2007)]. Dimensional reduction techniques such as Principal Component Analysis (PCA) are proposed to simplify the system and speed up the optimization process [Ye & Liu (2008)]. Another solution is to apply machine learning techniques for training a system that could give plausible combinations of reactions [Arikan et al. (2005), Zordan et al. (2007)].

One important research area of response system is to generate realistic balancing motions in the presence of external perturbations. The zero moment point (ZMP) is an important criterion to balance the body of a character [Fujimoto et al. (1998), Li et al. (1992), Nishiwaki et al. (2001)]. The Three-Dimensional Linear Inverted Pendulum Mode (3DLIPM) is a popular method used in robotics to simplify the linear dynamics of the lower body such that appropriate control torques can be calculated efficiently [Kajita et al. (2002), Kajita, Matsumoto & Saigo (2001)]. Since angular momentum is not considered in 3DLIPM, angular momentum generated due to noise or external perturbation has to be minimized to zero using feedback controllers [Kajita, Yokoi, Saigo & Tanie (2001), Napoleon et al. (2002)]. As a result, an enhanced version of 3DLIPM called Angular Momentum inducing inverted Pendulum Model (AMPM), which can counteract angular momentum induced by external perturbations, is proposed [Kudoh & Komura (2003)]. Using AMPM, it is possible to calculate reactive motions for bipeds that preserve dynamic balance during locomotion [Komura et al. (2004)]. By further considering the difference of the moment of inertia between the current posture and the corresponding posture in a reference captured motion, it is possible to synthesize the movement for counteracting external perturbations and gradually moving back to the original gait motion [Komura, Leung, Kudoh & Kuffner (2005)].

Apart from dynamics based approaches, response motions during collision can be generated by combining motion capture data with inverse kinematics or forward dynamics [Komura et al. (2004), Komura, Ho & Lau (2005), Zordan et al. (2005)]. The idea is to apply dynamics to simulate the effect of impact, and then blend the posture to a captured reactive motion. Thus, realistic motions of falling down, regaining balance or even avoiding collision can be generated effectively.

The general focus of response systems is the responsive movement of the character. Thus, it is not trivial to extend such systems to control characters that actively interact with other characters such as punching an opponent in boxing. In our research, we apply response systems to simulate the movements during collisions, and propose new methods to simulate active movements.

2.3.3 Statistical Analysis

A straight forward approach to generate dense interactions between characters is to first capture the interactions of multiple humans with a motion capture system, then extract statistical information such as the trajectories of movement from the captured data, and finally use machine learning techniques to reproduce motions under different situations [Sang Il Park & Shin (2004), Lee et al. (2007), Lerner et al. (2007), Kwon, Cho, Park & Shin (2008)].

The drawback of a these methods is the difficultly to gather statistical data. Even with the state-of-the-art technology, it is not easy to acquire the motion data of multiple persons. Some researchers avoid the problem by limiting the system to handle two characters [Sang Il Park & Shin (2004), Kwon, Cho, Park & Shin (2008)]. This is because capturing multiple persons using motion capture system is very difficult due to collision of body parts and occlusion. Another method is to limit the motions to be locomotion, and use overhead cameras to track the movement of a crowd without capturing the details of the motions [Lee et al. (2007), Lerner et al. (2007)]. Then, by statistically evaluating the translations of the people, characters with similar behaviors can be synthesized. However, due to the limitations on motion capturing, these algorithms cannot be applied directly to generate motions of multiple interacting characters.

2.3.4 Optimization Based Approaches

Optimization based methods are effective in synthesizing character motions. Usually, machine learning techniques are used to train a character to optimize a predefined objective functions. Based on reinforcement learning [Sutton & Barto (1998)], the optimal control policy can be trained by letting a character to perform actions and observing the benefits. It is used to train a character to reach a target location while avoiding collision with obstacles based on a Motion Graph [Ikemoto et al. (2005)]. By embedding information for motion interpolation, a more flexible controller that can blend multiple motions automatically can be trained [Lo & Zwicker (2008)]. Reinforcement learning can also be used to enhance the controllability of a character by predicting the user control signals based on the training samples [McCann & Pollard (2007)]. Alternatively, using active learning, one can generate real-time highly-controllable characters by adaptively capturing new actions that can improve the quality and responsiveness of the controller [Cooper et al. (2007)].

Reinforcement learning can be extended to simulate high quality interactions among multiple characters based on singly captured motions. It is used to train boxing characters to approach and hit a target [Lee & Lee (2004, 2006)] and to train computer based players in a fighting game [Thore Graepel (2004)]. However, these researches focus mainly on relatively simple interactions due to the high dimensionality, and hence complexity, of the human motion. By assuming a continuous state space for human motion, and computing the optimal weights for the bases of the motion trajectories, it is possible to represent human motion in a lower dimensionality space [Treuille et al. (2007)]. However, this method cannot be used for handling discrete actions such as pushing, pulling, and avoiding.

A general problem of these techniques is that they only provide solutions for simulating the interactions of two or three characters. They do not provide a method to simulate a scene of many characters, except from allocating a number of pairs in the scene [Lee & Lee (2004, 2006)], or randomly allocating characters in the scene and letting them interact with each other once their distance is close [Treuille et al. (2007)]. In movies or games, we wish to see characters concurrently interacting with multiple neighbors in a crowd. Our proposed method can generate such kind of many-to-many interactions effectively.

2.3.5 Topology Based Approaches

Simulating close interactions between multiple characters is difficult because of the large number of contact points. Recently, the topology space has been proposed to simplify close interactions synthesis. In the topology space, a human body is defined by a set of string and the interactions of two characters are defined by the tangles between the two set of strings representing the two bodies [Ho & Komura (2009*b*)]. By combining singly captured motions with tangle information, one can simulate collision-free close interactions between two characters with a small number of keyframes [Ho & Komura (2007*b*)].

The topology space can also simplify the motion planning process during close interactions. While path planning techniques like Rapidly-Exploring Random Trees (RRT) [LaValle & Kuffner (2000), LaValle (1998)] in joint space can generate simple interactions between a character and an object [Shapiro et al. (2007), Esteves et al. (2005)], applying RRT in the topology space of multiple characters can further guarantees collision free paths [Ho & Komura (2007*a*)]. Moreover, being a global optimization technique, RRT requires huge computational cost. It is shown that in the topology space, reasonable interactions can be generated even if local, frame-based optimization is used [Ho & Komura (2009*a*)].

Although keyframe based close interactions can be simulated easily with the use of topology space, it is unclear how such operations can be integrated with artificial intelligence controllers. Furthermore, since the number of string pairs increases exponentially with the number of characters, it would be difficult to apply topology operations for more than two characters.

2.4 Summary

The two streams of motion synthesis techniques, physical simulation and data-driven approaches, are capable of synthesizing motion of a single character. Recently researchers are trying to apply such techniques to create scenes when multiple characters interacting with each other. However, the results are still far from satisfactory. We will propose a new method to simulate dense interactions between a few characters, and extend the scheme to simulate real-time controllable interacting characters and crowd with interactions.

Chapter 3

Data Preparation

In this chapter, we explain the processes to capture the motions of actors individually, segment them into shorter semantic actions, classify them into different categories, compose a data structure called action level Motion Graph, and embed supplementary information into the actions.

We apply an optical motion system to capture human motion in 60 Hz with 35 markers. The positions of the markers are then converted to a character model represented by a hierarchy of 25 joints using third party software. Each joint has 3 degrees of freedom for rotation and the pelvis joint has an additional 3 degrees of freedom for translation. In other words, a pose can be represented by a 78 dimensional feature vector.

3.1 Motion Capture and Motion Segmentation

We capture long sequences of motions of a single human. Capturing the motion of one person instead of those by two persons eases the collection of data. When capturing the motions of two actors at the same time, there are many problems especially for the popular optical motion capture system due to the occlusion of the body segments. Usually, such data requires a huge amount of post-processing to correct the positions of markers, and the quality of the motions is not as good as those when one subject is acting. Another advantage of capturing alone is that we can make various combinations of motions. If we simply replay the captured motions, the type of interactions will be limited, and as a result, the final animation will appear monotonic.

Here we define the term “motion” as the raw-captured data, and the term “action" as a semantic segment of the motion we captured. In the field of fighting, an action can be an attack (such as a left straight, jab or a right kick), a defence (such as parries, blocking or ducking), a movement (such as stepping to the left, stepping forward or back step), a reactive motion when being hit / pushed away, or the combinations of them. We capture long sequences of motions instead of shorter actions individually. The major advantage is that we can preserve of naturalness of the movements. However, we have to segment them into shorter semantic segments for better character controls.

We develop an automatic motion analyzer to segment raw motions into actions. It consists of three steps. First, for a captured motion, the acceleration of all joints is calculated (Figure 3.1 (a)). Second, the supporting feet patterns throughout the motion are detected (Figure 3.1 (b)). Details on determining supporting feet pattern automatically will be explained in Section 3.2. Third, we detect the periods when the sum of squares of the acceleration of the joints is larger than a predefined threshold (Figure 3.1 (c)). In these high acceleration periods, the body is expected to go through continuous movements such as attacks and defences. Finally, segmentation is performed at the center of the double support phases. However, we do not segment the motions during the high acceleration periods detected in the previous step in order to preserve the continuity of high acceleration movements (Figure 3.1 (d)). Easy as it may sound, due to the large variety of human actions, it is difficult to create a perfectly accurate segmentation system. Therefore, we allow users to fine tune the results.

We manually embed high level information into the segmented actions. First, for each action, there must be at least one movement class indicating the nature of the action, such as "attack", "avoid", "block", "movement", etc. Notice that there may be multiple movement classes within an action, such as blocking an attack with the arms while avoiding. Second, for actions that required the usage of specific joints, we record the names of the joints. For instance, in the action "right hook punch", the joint "right hand" is recorded to be the joint being used in the action. Finally, we include high level descriptions for special actions. In our system, such information includes the attacking direction for "attack" actions, and the rough defending body parts for "block" and "avoid" actions. They help in evaluating the suitability for performing the actions in different situations.

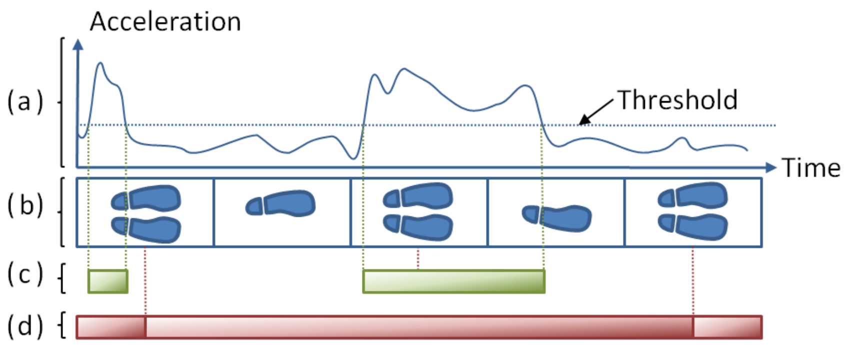

Figure 3.1

The motion segmentation system. (a) The sum of square of acceleration for all joints against time is plotted. (b) The supporting feet patterns of the motion are detected. (c) By applying a threshold in the acceleration plot in (a), the periods with high acceleration are extracted. (d) Actions are segmented at the centers of the double supporting phases, except those cutting through the high acceleration period extracted in (c).

3.2 Supporting Feet Patterns

We follow the algorithm proposed by Ikemoto et al. [Ikemoto et al. (2006)] to detect supporting feet patterns in our capture motions. While the authors suggest using a dedicated model with high dimensional feature space, we found that a simplified model works reasonably well. We simplify the model in two ways. First, instead of using the joint positions all the lower body joints as a feature vector, we only consider the speed and height of each individual foot. Second, we monitor a single frame instead of a window of frames to maintain a low dimensional feature space.

We apply a soft margin support vector machine (SVM) [Boser et al. (1992), Cortes & Vapnik (1995)] as our classifier. A two dimensional feature vector is used in the SVM to indicate the movement speed and height of a foot. Each feature vector is associated with a class, which is either 1 for "supported" or -1 for "unsupported". During the training stage, the user indicates the supporting feet for some actions, and the system generates training samples by associating the feature vectors with the classes of supporting feet. Then, the SVM calculates the hyperplane that maximizes the margin of the two classes of data:

$$\begin{gathered}\operatorname{argmin} \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_i \xi_i \\ \text{subject to } c_i(\mathbf{w}\cdot\mathbf{x_i} - b) \geq 1 - \xi_i\end{gathered} \tag{3.1}$$

where $\mathbf{w}$ is the normal to the hyperplane, ξ_i is the slack variable which measure the degree of misclassification of the feature $\mathbf{x_i}$, C is a parameter for penalty on error, c_i is either 1 or -1 indicating the class of $\mathbf{x_i}$, b determines the offset of the hyperplane from the origin along $\mathbf{w}$. Our SVM apply radial basis function as the system kernel:

$$K(\mathbf{x_i}, \mathbf{x_j}) = e^{-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2} \tag{3.2}$$

where $\mathbf{x_i}$, $\mathbf{x_j}$ are feature vectors, $\gamma > 0$ is a parameter for the kernel.

Notice that C in Equation 3.1 and γ in Equation 3.2 need to be tuned. Our system searches for the optimal values using brute force grid search. We train a SVM for each possible quantized values of C and γ. Then, we evaluate the performance of the SVM using leave-one-out cross-validation. The C and γ that perform the best will be used.

Once the SVM is trained, we can apply it to classify the supporting feet of any posture in the actions we have during run-time. Since the SVM is simple, the computational cost is neglectable. Apart from the small computational cost, the major advantage of our simplified system is that the amount on training data can be dramatically decreased. In the research of Ikemoto et al. [Ikemoto et al. (2006)], around 9000 frames of training data is needed, which is equivalent to 150 seconds of motion data. It should be noted that most motion database in our system contain only tens of seconds of data, and hence using such a large feature space is not feasible. With our simplified system, around 20 seconds of training data is enough to produce reasonably accurate classification.

3.3 Action Level Motion Graph

We build a Motion Graph [Arikan & Forsyth (2002), Lee et al. (2002), Kovar et al. (2002)] in the action level rather than the frame level, as in [Gleicher et al. (2003), Lau & Kuffner (2005), Kwon & Shin (2005)]. We call this data structure the action level Motion Graph. Planning based on such a graph is similar to the way human does, as people also use actions such as attack and defence as the fundamental entities during planning.

The processes to generate an action level Motion Graph are visualized in Figure 3.2. First, we extract the starting poses and the ending poses of all actions. Then, we apply K-mean to cluster similar poses together. The system determines the number of clusters, K, iteratively with a predefined threshold indicating the maximum pose difference within a cluster. Each cluster represents a group of similar poses and becomes a node in our Motion Graph. Some starting poses and ending poses may be grouped into the same cluster. This means that actions, which are represented by edges, with such ending poses can be connected to actions with such starting poses.

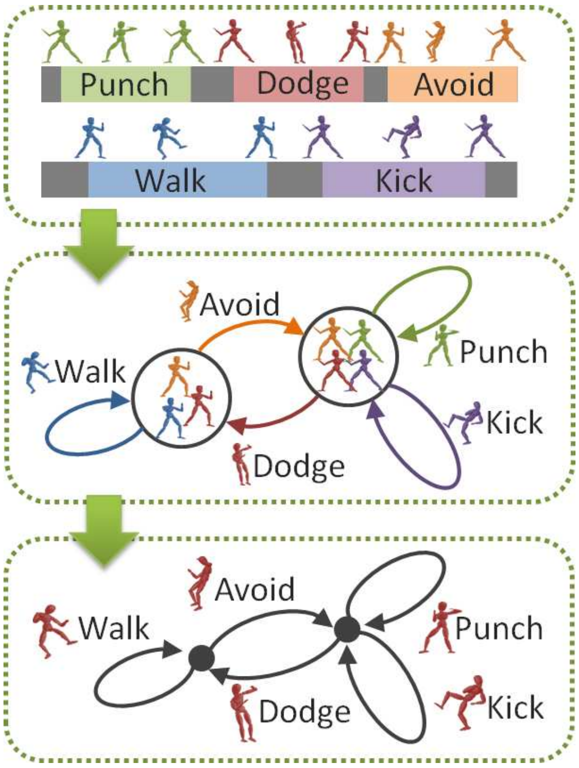

Figure 3.2
An action level motion graph that is generated from the boxing motion. (Upper) Starting and ending poses of actions are extracted. The bars represent motions and the colored regions represent actions. (Middle) Similar extracted poses are grouped into nodes, and actions are represented as edges. (Lower) The resultant action level Motion Graph.

We design our distance function as a weighted sum of six terms to calculate the difference between two poses during the K-mean classification. The weights are manually designed. For any two poses A and B, we first align them by translating along the floor plane and rotating around the vertical axis. Then, we apply the distance function to calculate their difference. The first two terms of the function evaluate the posture difference for the upper body:

$$D_{upper} = \sum_i \|J_i^A - J_i^B\| \tag{3.3}$$

$$D_{upper'} = \sum_i \|J_i^{A\prime} - J_i^{B\prime}\| \tag{3.4}$$

where J_i is the three dimensional Euler angle of the joint i, J_i' represents its derivative, and i represents the set of joints above the pelvis. The superscript represents the two poses A and B respectively. The next two terms evaluate the posture difference for the lower body:

$$D_{lower} = |L_l^A - L_l^B\| + |L_r^A - L_r^B\| \tag{3.5}$$

$$D_{lower'} = |L_l^{A\prime} - L_l^{B\prime}\| + |L_r^{A\prime} - L_r^{B\prime}\| \tag{3.6}$$

where L_l and L_r are the three dimensional leg vectors [Komura et al. (2004)] pointing from the pelvis to the left and the right foot respectively, L_l' and L_r' are the corresponding derivative. The final two terms evaluate the height difference for the body:

$$D_{height} = |H^A - H^B| \tag{3.7}$$

$$D_{height'} = |H^{A\prime} - H^{B\prime}| \tag{3.8}$$

where H is the height of the pelvis and $(H)'$ is its derivative. The final distance function between the two poses becomes:

$$\begin{aligned} D(A,B) \quad = \quad & w_{upper}D_{upper} + w_{upper'}D_{upper'} + w_{lower}D_{lower} \\ & + w_{lower'}D_{lower'} + w_{height}D_{height} + w_{height'}D_{height'} \end{aligned} \tag{3.9}$$

where the w terms are the corresponding weights. The design to separately weight the upper body, the lower body and the height of the body is important to cope with different motion data sets. For example, if we are considering fighting motions, all terms are equally important. However, for locomotion such as jogging and walking, the upper body and its derivative are far less important than the other terms, and the corresponding weights can be set small.

3.4 Action Combination Table

When we simulate interactions in later sections, we define objective functions to evaluate the suitability for the characters to perform any actions. The functions are useful in many aspects concerning interactions, but they fall short to represent the implicit factors that affect the action selection process. These implicit factors are difficult to be evaluated mathematically and require extra information from the animators. In this section, we explain the action combination table, which helps to determine the optimal actions to be performed based on the action by the opponent. The table is maintained manually. It (1) enables the system to take into account subtle factors of interactions which cannot be simply expressed by objective functions, and (2) provides interface to animators who want to pair up specific actions.

Each record in the table contains three fields: the character's action, the opponent's action, and a suitability value that describes how effective the character's action is with respect to the opponent's action. A positive value encourages the character to perform such an action when the opponent is performing the opponent's action, while a negative value means such an action would better be avoided.

In our system, the table is used to evaluate the quality of defensive actions during fighting, which is difficult to be evaluated numerically. It is known in boxing that sway back motion is effective for avoiding upper cuts and hooks, and head slip is good for avoiding straight punches. There are various factors such as the direction the punch is approaching from, and whether the defender can see the attacker all through the motion, that support these basic techniques. However, it is tedious to represent all factors accurately with numerical functions. By using the action combination table, we can indicate how appropriate a defence is with respect to an attack, and hence see more effective defences like those appearing in real boxing matches. The details on applying the table will be explained in Section 4.4.

The action combination table also provide animators an interface to embed manually designed plausible close-contact interactions into the scene. There may be special attacks and defences that look good while performed by two characters. Such action pairs, however, may not be well evaluated by objective functions due to their low effectiveness. With the table, the animator can make such interactions appear with minimal adjustments to the system. Although we only implement the attack-defence relationship in our system, the table can similarly be applied to pair up actions such as good-looking moves in dances, as well as tackles-avoids in soccer.

3.5 Summary

In this chapter, we explain the concept of action, which is a semantic segment of raw captured motions. We automatically segment actions based on the acceleration profile of the movements and the supporting feet patterns. We train a support vector machine to determine the supporting feet automatically. By applying K-mean clustering, we create the action level Motion Graph. We also design the action combination table to store high level relationship between actions.

Chapter 4

Temporal Tree Expansion

Synthesizing animations of multiple characters closely interacting with one another has a high demand in the computer animation and the 3D computer games industries. Due to the difficulties of directly capturing the motions of multiple subjects simultaneously, many methods to synthesize such animation from singly captured motions have been proposed. Most of them are based on optimal control: defining an objective function that represents the benefits when the characters perform the motions, and maximizing the long horizon rewards to control the characters.

However, unlike traditional optimal control in which the environment is static, simulating interacting characters requires the consideration of different possible reactions performed by the opponents. Game theory provides an excellent framework to model such planning process. In this chapter, we propose a new methodology based on game theory to synthesize animations of multiple characters intelligently competing with each other in a dense environment. We expand the game tree to evaluate all the possible results in the future. Then, min-max search is used to select actions that maximize the score of the controlled character and minimize that of the opponent. With our proposed method, the characters behave intelligently for interactions such as martial arts, tags and sport games.

A major problem of applying game theory to character control in such a competitive environment is the low controllability of the overall scene. If we design a reward function that guides all the characters to behave in the way we want, the opponents of the controlled character will try to penalize the character's reward. Therefore, it is difficult for the animators to control the scene. We propose a new method to embed a cooperative term into the traditional min-max framework. In our system, the characters compete intelligently based on the competitive functions, while cooperate with each other to follow the high level requests from the animator.

To show the effectiveness of our method, we simulate various competitive interactions of the characters. We show examples of boxing matches, in which the strength of each fighter can be adjusted by changing the depth of the game tree expanded. We also adjust the control parameters of the characters to simulate different styles of fighting, including outboxing and infighting. The animators can control the overall moving trajectories of the characters, as well as the frequency of launching a specific action. Our method is effective for animators to design scenes with a crowd of well controlled characters interacting like real humans.

4.1 Contributions in This Chapter

- We propose a new method to simulate dense interactions of intelligent characters by techniques in applying game theory such as game tree expansion and min-max search.

- We propose a multi-modal approach to enable the characters competing with each other while cooperatively achieving common goals.

4.2 Outline of the Method

The outline of our system is shown in Figure 4.1. It consists of five steps:

1. Capture the motion of a single actor.

2. Segment the motion into semantic actions, and organize the actions in an action level motion graph.

3. Simulate the interactions of two characters by expanding a game tree, which predicts the future states with respect to different choice of actions.

4. Evaluate the game tree by min-max search and select the optimal action.

5. Let the character perform the optimal action, and hence generate a scene of dense interactions.

Steps 1 and 2 are precomputed, while steps 3 to 5 are performed in run-time repeatedly. Whenever a character finishes its action, steps 3 to 5 are performed so that it selects the optimal action.

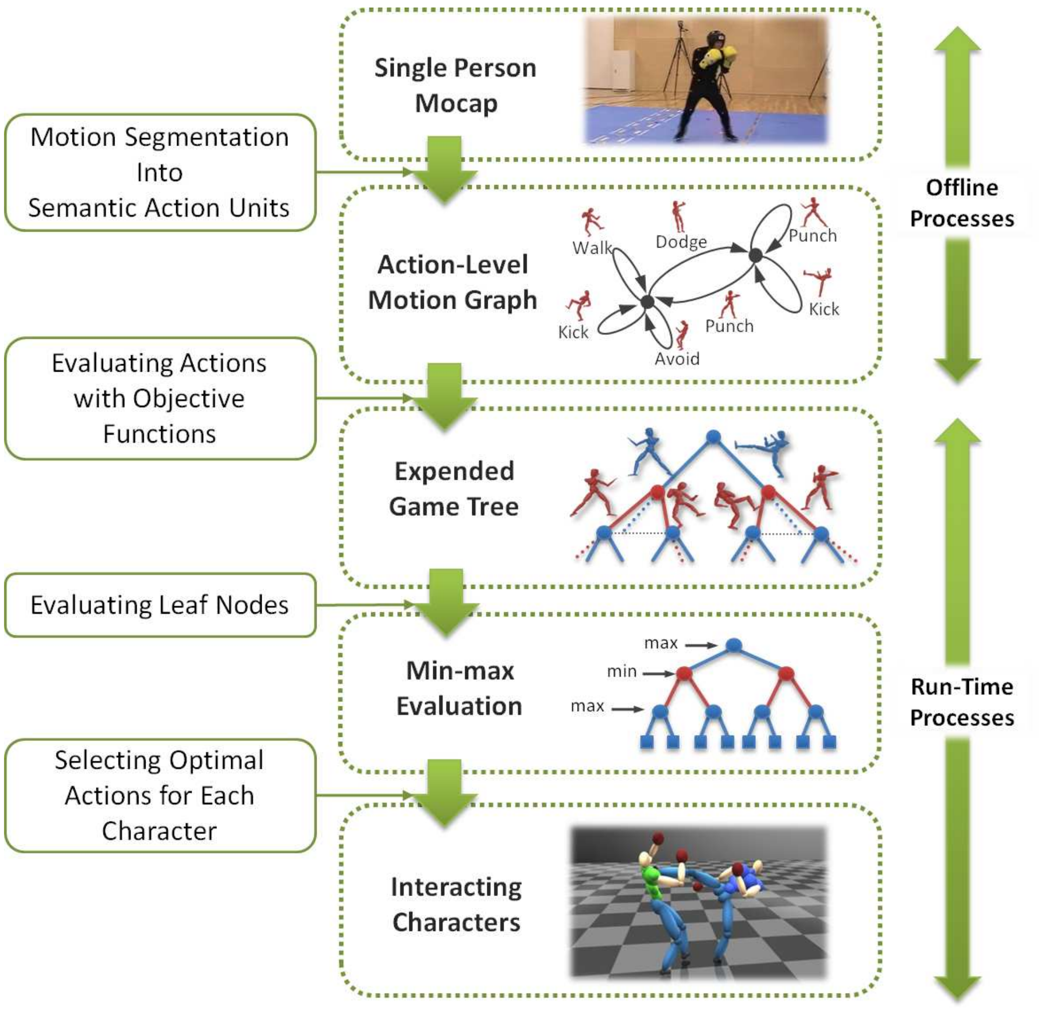

Figure 4.1: The outline of the temporal tree expansion method.

4.3 Multi-modal Character Control

In this section, we explain our multi-modal control method that enables the characters to compete with others while cooperate in the aspect to satisfy the requirement of the animator. When a character is going to select a new action, a game tree is expanded and the possible outcomes in the future are evaluated. Using min-max search, a character can be controlled intelligently to compete with its opponents. However, min-max search is surprisingly inefficient to control the overall scene, such as encouraging the interacting character to follow a predefined path. If we apply an objective function to guide the characters to follow the path, they try to prevent their opponents to follow such a path due to the min-max framework. As a result, the deeper the tree is expanded, the smarter they are to block the opponents, and the slower they follow the path. We propose a multi-modal evaluation system that embeds cooperative evaluation functions into the min/max framework, such that the characters have the intelligence to compete with each other while helping each other to achieve common goals.

4.3.1 Game Tree Expansion

For controlling the characters, we adopt methods used for AI players in strategy games such as chess. To control them intelligently, only considering the immediate benefit is not enough. For example, in chess, a movement that shows the greatest effect in one ply, such as taking a valuable piece like a castle or a bishop, is not necessarily the best choice for winning at the end. By expanding the game tree and evaluating the static position after a few plies, one can make a choice that benefits the player in long term. Here we apply a similar approach to evaluate the long term benefit of performing an action.

The major difference between character interactions and chess is that the choices made by the characters are performed in a continuous time domain. Every node in our game tree represents the state of interaction between two characters when either of them is about to select a new action. The edges from the node represent the possible choices of actions in such a state. Notice that the two characters perform their selected action concurrently, and whenever any of them finish their actions, a node is added and another expansion is performed. Figure 4.2 shows an example of an expanded game tree, with the vertical axis representing time. The blue character starts the game tree expansion process with a choice of two actions, a_1 and a_2, at the time t_1. Based on the choice of the blue character, the red character has a choice of actions to counter act at t_2. Notice that the action selected by the blue character is still continuing when the red character makes the selection, as indicated by the blue dotted lines. The action a_1 ends at t_3, and another level of nodes will be added for the blue character with edges a_3 to a_6, based on the red character's choice of actions.

Since the actions in our game tree have different durations, the order of ex-

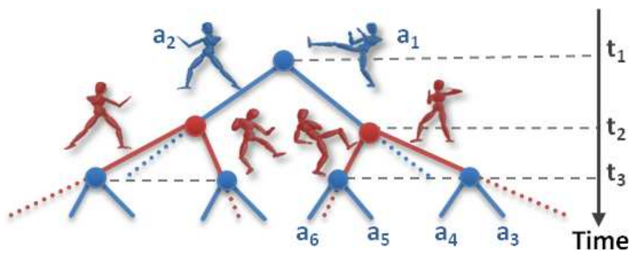

Figure 4.2: An expanded game tree of fighting between two characters. The distance along the vertical axis represents time. The nodes represent the states of fight of the two characters when any of them select new actions, and the edges represent the choices of action. The dotted lines indicate the continuation of the selected actions while the opponent selects its actions.

pansion is not always alternate between the two characters. If a character selects an action with long duration, its opponent may perform several actions before its next turn to select. In Figure 4.3, the blue character selects a long action a_1 at t_1. When the red character expands a node at t_2, since it selects a short motion b_1, it can further expand another node when b_1 ends at t_3. Finally, when the a_1 ends at t_4, the blue character expands the tree again.

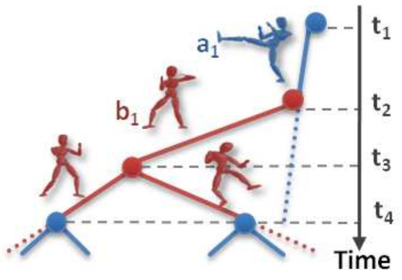

Figure 4.3: The expansion of the game tree is not always alternate. Since the action selected by the blue character is long, the red character expands two levels of tree before the blue character further expands.

In some situations, a character may be forced to perform an action when some criteria are satisfied. In terms of fighting, when a character is being hit, it will either be knocked down onto the ground immediately, or just lose balance and walk a few steps to recover the balance and resume the fight. This response motion will be decided based on the current state of the body and the impulse added to the body, and the character being hit does not have a choice for the action. In our system, according to the posture of the body and the direction and strength of impulse, we simulate the initial reaction based on rigid body dynamics and then blend the motion with the reactive motion selected from the motion database [Arikan et al. (2005), Zordan et al. (2005)]. Figure 4.4 shows an example of launching reactive motion during game tree expansion. The blue character selects a_1 and the red character selects b_1. It turns out that the red character will be hit by the blue one at t_1. The red character is forced to discard the latter part of b_1 as shown in the dotted part of b_1, and perform a falling back action b_2. Since the red character has to perform the reactive motion, it loses its chance to expand the game tree.

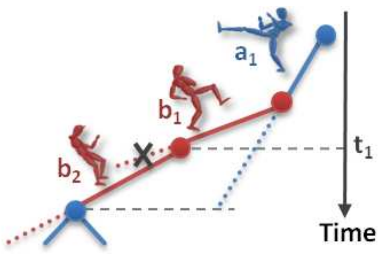

Figure 4.4: The character is forced to stop the current action to perform a reactive action when being hit. Since the reactive motion is determined by the system rather than selected by the character, there is only one outgoing edge.

4.3.2 Evaluating Competitiveness and Cooperativeness

We adopt the min-max framework to evaluate the long term benefit of launching an action. The framework is updated in two aspects to fit in our problem. First, the state evaluation for each leaf node is performed by evaluating the whole path from the root node to the leaf node, rather than considering the leaf node only. Second, we embed cooperative function into the evaluation process such that the character can compete intelligently while cooperatively achieve a predefined goal.

For every edge e, we define two functions to evaluate the competitiveness ($F^{comp}(e)$) and the cooperativeness ($F^{coop}(e)$). The details of the objective functions will be explained in Section 4.4. Suppose character A is competing with character B and is expanding the game tree to select an action. The node expanded by A is called a max node and that by B is called a min node. The competitive and cooperative scores of a leaf node l in the tree are defined as:

$$\begin{aligned} S^{comp}(l) &= \sum_{e_i \in \mathbf{e}_{\mathbf{max}}} F^{comp}(e_i) - \sum_{e_j \in \mathbf{e}_{\mathbf{min}}} F^{comp}(e_j) \\ S^{coop}(l) &= \sum_{e_i \in \mathbf{e}_{\mathbf{max}}} F^{coop}(e_i) + \sum_{e_j \in \mathbf{e}_{\mathbf{min}}} F^{coop}(e_j) \end{aligned} \tag{4.1}$$

where $\mathbf{e_{max}} \cup \mathbf{e_{max}}$ represents the set of edges from the root node to the leaf node l, with $\mathbf{e_{max}}$ represents the set of edges expanded from the max nodes, and $\mathbf{e_{max}}$ represents those from the min nodes. Figure 4.5 (Left) gives an example of leaf node evaluation. In the figure, for the leaf node l, $S^{comp}(l)=F^{comp}(e_0)+F^{comp}(e_2)-F^{comp}(e_1)$ and $S^{coop}(l)=F^{coop}(e_0)+F^{coop}(e_2)+F^{coop}(e_1)$.

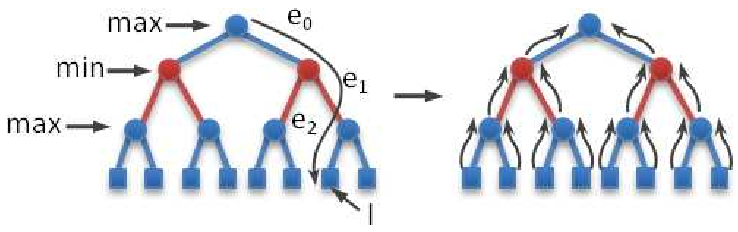

Figure 4.5: (Left) The scores of the leaf nodes (squares in the figure) are evaluated from the root node to the leaf nodes with Equation 4.1. (Right) Min-max is conducted by recursively applying Equation 4.2 and 4.3 from leaf nodes to root node.

As in traditional min-max evaluation systems, equation 4.1 is applied to calculate the scores of the leaf nodes only. For the internal nodes, the scores are evaluated by propagating that of the leaf nodes recursively from the leaf nodes towards the root node, as shown in Figure 4.5 (Right). The cooperative score are embedded into the min-max structure. It is evaluated differently for the min and max nodes such that both characters regard the cooperative terms as benefits. For an internal node n with its set of children nodes n_i, the optimal choice c of the children nodes is evaluated as:

$$c = \begin{cases} \text{argmax}_i(S^{comp}(n_i)+S^{coop}(n_i)) \text{ if } n_i \text{ is a max node} \\ \text{argmin}_i(S^{comp}(n_i)-S^{coop}(n_i)) \text{ if } n_i \text{ is a min node} \end{cases} \tag{4.2}$$

and the scores of the internal nodes n are evaluated as:

$$\begin{aligned} S^{comp}(n) &= S^{comp}(c) \\ S^{coop}(n) &= S^{coop}(c) \end{aligned} \tag{4.3}$$

Equation 4.2 and 4.3 are applied to evaluate all internal nodes recursively from the leaf nodes towards the root node. Finally, Equation 4.2 is applied to the root node to select the optimal action to perform.

4.3.3 Pruning Non-Plausible Choices

In order to reduce the computational cost and avoid non-plausible interactions, we prune the mal choices of actions when expanding each node in the game tree. This pruning can be applied alongside with traditional alpha-beta pruning as we follow the min-max framework.

Although there are a huge number of choices for the actions to launch, many of them never happen as they cause obvious disadvantages or illogical behaviors (Figure 4.6). The criteria to prune the actions based on the situation of the characters are listed below:

- **Actions that cause penetration to the opponent**: Collisions of the characters are examined. The actions that cause one character brutally overlapping with the others are considered invalid. If none of the actions can avoid penetration, we try to keep those that are penetration-free at the last frame, such that the next interaction does not start with penetration.

- **Actions that end up with wrong facing angles**: We require the character to face the opponent at the last frame of the actions. In our system, we apply this criterion to ensure that the fighting and chasing characters do not turn their back to their opponents when they finish any actions.

- **Actions out of distance**: It is meaningless to launch some actions if the opponent is further than the reaching distance. This criterion is applied in our fighting experiments such that the character does not attack nor defence if the opponent is further than 1 and 3 meters respectively.

- **Actions required to be launched**: In some situations, only a specific set of actions logically makes sense. This criterion is applied in our fighting system to enforce the characters selecting only defensive actions when the opponent attacks.

The criteria listed above are in descending order of importance. If none of the actions can satisfy all criteria, the subset that satisfies those of higher importance will be selected. By pruning the actions as suggested, apart from ensuring that animations appear natural, we reduce the computational cost of strategy making. Empirically, we can prune at least half of the available choices using these pruning policies. This would reduce the computational cost approximately by $O(\frac{1}{2}A^D)$, where A is the number of available actions and D is the depth of the game tree.

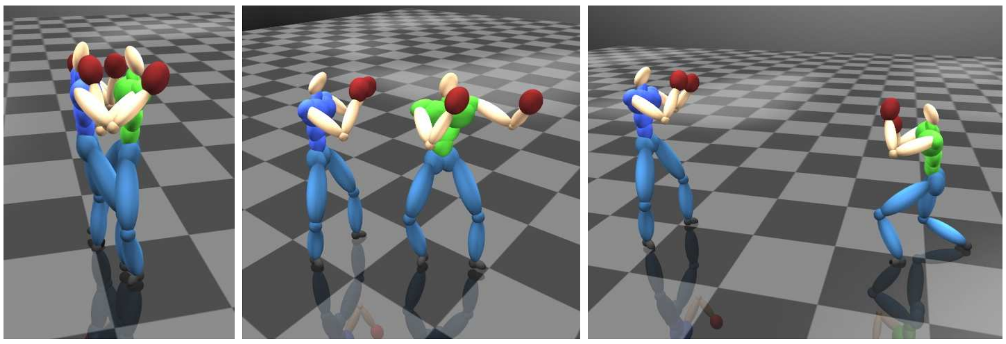

Figure 4.6: Examples of actions launched by the green character that has to be pruned: (Left) Penetrating the opponent (Middle) Turning the back to the opponent while fighting (Right) Defending while opponent is far away

4.4 Objective Functions

In this section, we explain the objective functions used in Equation 4.1 to evaluate the competitiveness and cooperativeness of the interactions. The competitive function evaluates how good an action is for a character to compete with its opponent in a game. The cooperative function evaluates how good the character can cooperate with its opponent by launching an action. By combining the two functions, controllable characters with realistic behaviors can be generated. The exact values of the parameters used in these functions can be found in Table 4.1 and Table 4.2.

4.4.1 Competitive Function

The competitive function evaluates how good each character is competing with the other characters during close interactions including fighting, chasing and sports. The function consists of three terms: the **movement term**, the **scoring term**, and the **action combination term**. In this section, we explain the objective of each term, and detail the function design of each term in our system.

The **movement term** evaluates the distance and facing angle. Since the character is supposed to compete with its opponent, the movement term is defined with respect to the opponent:

$$f^{mov1} = w_\theta(\theta - \theta_d)^2 + w_r(r - r_d)^2 \tag{4.4}$$

where θ, r are the relative orientation and distance from the opponent respectively, θ_d, r_d is the preferred relative orientation and distance, and w_θ, w_r are the weight constants for each term. In our system, we always set $\theta_d = 0$ such that the character tries to face the opponent. r_d depends on the type of interaction and the movement style. For example, in boxing, an infighter prefers to keep short distance with the opponent. In that case, r_d is set to be small such that higher scores are given to actions that bring the character closer to its enemy. On the contrary, for an outboxer or a passive fighter who prefers to escape from the opponent, high scores are given to actions that increase the distance between them.

The **scoring term** evaluates how effective the action is to compete with the opponent based on the rules of the game. In general, it is defined as the weighted sum of the damage the character giving to and receiving from the opponent:

$$f^{score} = w_D^+ D^+ - w_D^- D^- \tag{4.5}$$

where D^+ is the damage that the character gives to the opponent, D^- is the damage to be received, and w_D^+, w_D^- are positive weight constants for each term. In our system of boxing, the damage is set proportional to the velocity of the attacking segment at the moment it lands to the opponent. The weight constants depend on the competing style of the character. For boxing, in case the fighter is an outboxer that is less aggressive, w_D^+ is set small and w_D^- is set large. In case a fighter is running out of time and is losing the fight, it has to fight more aggressively regardless of the risk of being hit; in that case, w_D^+ is increased and w_D^- is decreased.

The **action combination term** evaluates the suitability of performing an action based on the action combination table explained in Section 3.4. Considering the action to be performed by the character and its opponent, we search the table to see if such a pair of actions is defined:

$$f^{comb} = \begin{cases} w_S S & \text{if the action pair exists in the table} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where w_S is a weight, S is the suitability value as indicated in the table. In our system, this term is only used to evaluate the quality of defensive action when the opponent is attacking.

The competitive function is the sum of the three terms:

$$F^{comp} = f^{mov1} + f^{score} + f^{comb} \tag{4.7}$$

The competitive function is general enough to be used for various competitive interactions such as fighting, chasing, and sports. For example, in a game of chasing, we can increase the preferred distance for the character running away, and shorten it for the chaser. For sports like basketball, we can design a scoring function that considers the probability to throw the ball into the basket, such that the character will try to shoot when there is no opponent in front of it.

4.4.2 Cooperative Function

The cooperative function evaluates how much the characters are cooperating to achieve a common goal. In general, such a common goal is the requirement of the animator. For example, the animator might want to specify the overall trajectory of the two characters when moving. Alternatively, he/she might want the characters to launch actions in some specific style. Such factors are evaluated by the cooperative function. The cooperative function is composed of two terms: the **movement term** and the **special requirement term**.

The **movement term** is defined similarly to the one used in competitive function. The difference is that we now consider the global position and orientation of the character after performing the action:

$$f^{mov2} = w_\gamma(\gamma - \gamma_d)^2 + w_p(p - p_d)^2 \tag{4.8}$$

where γ, p are the global orientation and position of the character in the world coordinate system, γ_d, p are the respective desired value, w_γ and w_p are the respective weights. Empirically, we found that if we wish a character to follow a predefined trajectory, instead of utilizing the γ term, it would be more effective to define the trajectory as a series of check points, and update the value of p_d whenever a checkpoint is reached.

The **action requirement term** gives high score to the character if a specific action is performed:

$$f^{req} = \begin{cases} w_r & \text{when } A^+ \text{ performed} \\ -w_r & \text{when } A^- \text{ performed} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where w_r is the weight, $A+$ is the set of actions to be performed and $A-$ is the set of actions not to be performed. The animator can make use of the action requirement term to favor the use of good looking actions when they are successfully performed. Also, by requesting a character to perform reactive motions, we implicitly require such a character to be hit by the opponent, since reactive motions are not designed to be performed without being hit. By dynamically updating A, the animator can control the flow of the animation.

Finally, the two terms are summed to compose the cooperative function:

$$F^{coop} = f^{mov2} + f^{req} \tag{4.10}$$

4.5 Experimental Result

An optical motion capture system was used to capture the motions of one actor at a time. The frame rate was set to 60 postures per second. We have captured the shadow boxing motion of an energetic kick boxer for 7 minutes, that of a tired boxer for 7 minutes, and a running-around motion for 1.5 minutes. They were automatically segmented into 279, 240 and 215 actions, respectively. These motion sets were used to control the virtual characters. Various experiments based on the temporal expansion approach were conducted. The parameters of the competitive and cooperative functions for each experiment are shown in Table 4.1 and Table 4.2 respectively.

The computation time depends on the size of the action set and the connectivity of the Motion Graph. In general, using a computer of Pentium 4 Dual Core 3.0 GHz CPU and 2 GB of RAM, it takes 5 minutes to create a video of 30 seconds when expanding the game tree for three levels to determine every action of the characters. As discussed before, since the computational cost increase exponentially with respect to the depth of the tree, expanding a game tree with more than five levels is not advised.

When rendering the scene, we designed a particle system to handle unexpected collisions among the characters and adjust the actions performed for better visual effects. Further information can be found in Appendix A.

4.5.1 Kick Boxing

Firstly, we simulated a fight between two characters using the actions of the energetic boxer. High quality interactions such as realistic attacks and defences were shown (Figure 4.7). Although both characters use the same action set, we can simulate different levels of intelligence by altering the depth of the game tree expansion. We simulated a less intelligent fighter by setting the intelligence level to two, and a smart fighter by setting the level to four. The intelligent fighter always wins the match as its decision is based on further expansion of the game tree. When designing the weight of the objective function, since the purpose of the movement term was just to guide the characters to their sweet spots for attacking, its corresponding weight was set smaller than that of the scoring term.

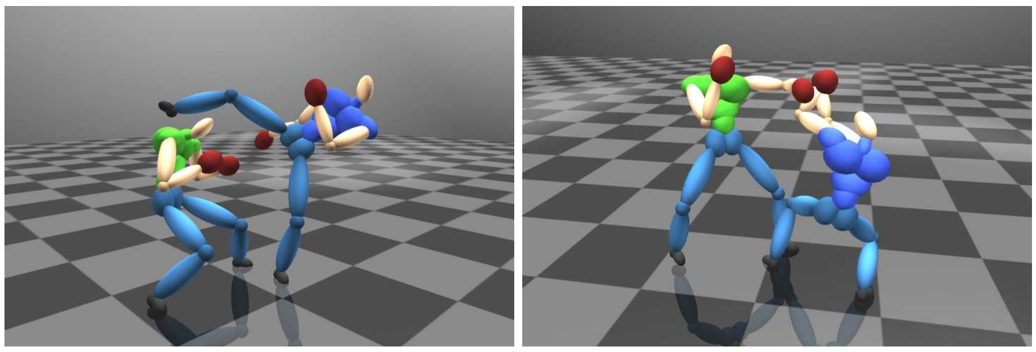

Figure 4.7: High quality interactions simulated by the temporal tree expansion method. The figures show realistic attacks and defences performed by the two characters.

Secondly, we simulated a match between an energetic fighter and a tired fighter. Since the motions of the tired fighter are slow, the tired character keeps being hit by the energetic fighter when the intelligence levels are the same (Figure 4.8). However, the tired character becomes stronger than the energetic fighter when it expands the game tree much deeper than that of the energetic fighter. In our experiment, we expand five levels for the tired boxer and two levels for the energetic boxer. As a result, although the movements of the tired boxer are slow, the boxer arranges an effective sequence of actions to hit the opponent (Figure 4.9).

Thirdly, different styles of fighting were simulated by adjusting the objective function. It is known that infighters prefer to fight in close distance, and hence use short range attacks such as upper cuts and hooks more frequently. As a result, they become more aggressive as the duration of such attacks are short, and stopping the attacks will endanger the fighter as he/she will be in the reaching distance of the opponent. On the other hand, outboxers prefer to keep distance from the opponent and use long range attacks such as straight punches and kicks more often. They also move around more as they need to keep distance with the opponent. In order to simulate such effects, we first classified the attacks into short and long range ones based on the attacking positions. Then, an aggressive infighting style is modeled by setting the preferred distance to short, and giving higher score to successful short range attacks (Figure 4.10 (a)). The outboxing style is modeled

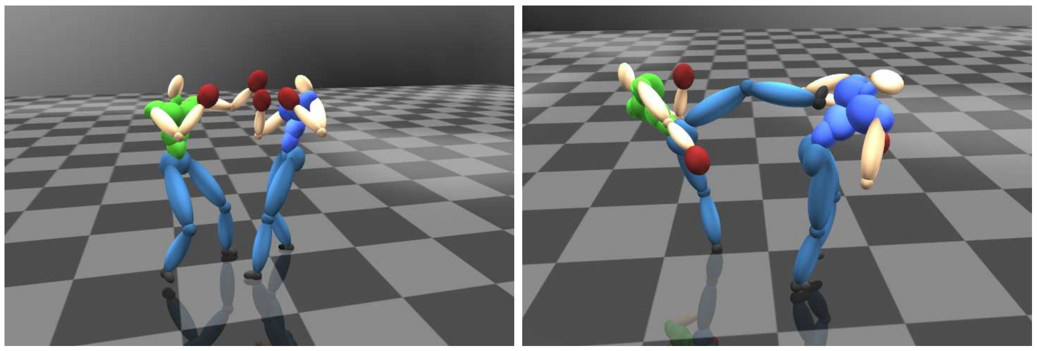

Figure 4.8: Energetic boxer (the green character) fighting with a tired boxer (the blue character). With the same intelligence levels, the tired boxer keeps being hit due to (Left) the inefficient attacks and avoids, and (Right) slow movements such as turning around.

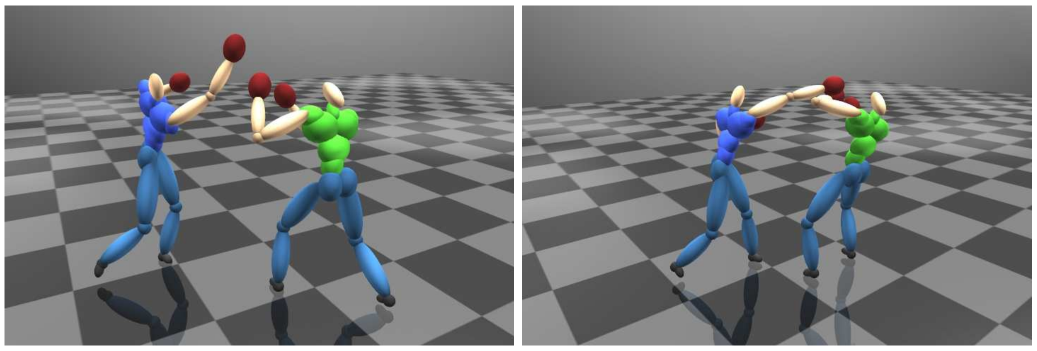

Figure 4.9: Energetic boxer (the green character) fighting with a tired boxer (the blue character). With a superior intelligence level, the tired boxer can hit the opponent with an effective arrangement of the slow punches.

by setting the preferred distance long and giving higher scores to successful long range attacks (Figure 4.11 (b)).

Then, a scene of a crowd of fighters moving along predefined pathes while fighting was simulated (Figure 4.12). Each of the pathes is modeled as a series of check points to be reached by a pair of characters. Each check point is defined by a 2D position on the floor, an optional timing value, and the optional requirements on action usage. Whenever a character reach the position of the current check point, and wait until the indicated time has reached, we update the check point to the next one. Since the movements of the characters are constraints statio-temporally, we can design a scene with a lot of characters without any risk of unexpected collision. We made use of the movement term in the cooperative function in Equation 4.10 such that higher score is given to an action that guides the char-

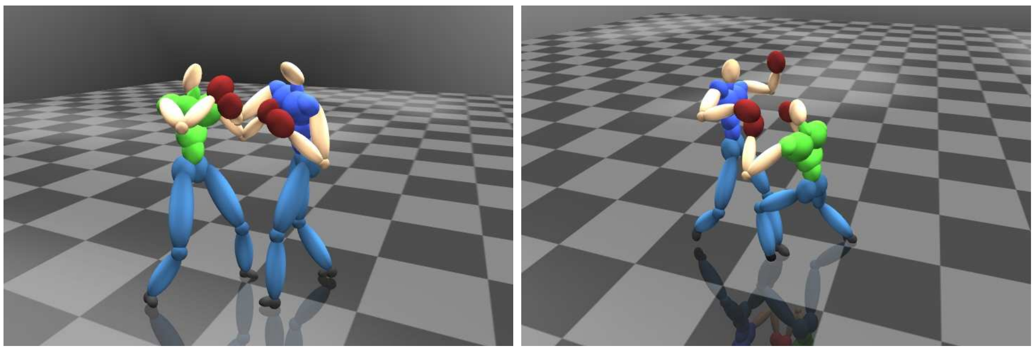
Figure 4.10: Infighters simulated by our system that prefer (Left) short distance to opponent and (Right) short range attacks such as hook punches, upper cuts, elbow punches and knee kicks.

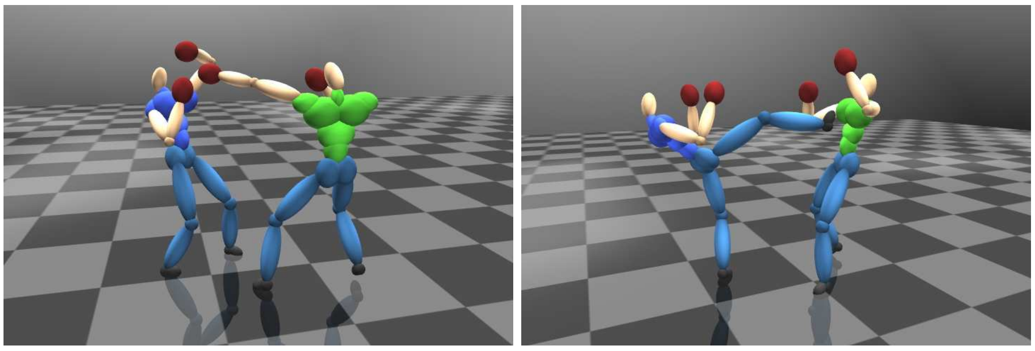
Figure 4.11: Outboxers simulated by our system that prefer (Left) long distance to opponent and (Right) long range attacks such as kicks and straight punches.

acters to the next check point. We also make use of the action requirement terms to tell the blue characters knock down their opponents at the last check points. On the other hand, the fighting behaviors of the two characters are simulated by the competitive functions in Equation 4.7.

4.5.2 Chasing and Running Away

A scene where a character chases another was simulated (Figure 4.13). The movements of both characters are based on the running-around motion. The preferred distance of the chaser is set short and that of the character that is running away is set long. Moreover, based on the scoring function, when the chaser catches the opponent, high score is given to the chaser and high penalty is given to the opponent. As a result, the chaser tries to approach its opponent while the opponent tries to get away. When we increase the intelligent level of the chaser to four and lower that

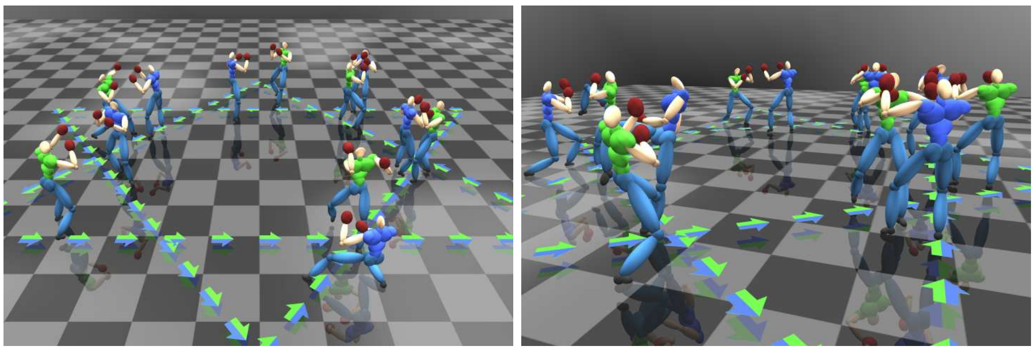

Figure 4.12: TA crowd of characters fighting with the opponents competitively while following predefined curve cooperatively using our multi-modal framework.

of the running away character to two, the chaser can catch the opponent quickly (Figure 4.14). By integrating the cooperative function, we can simulate catching and running away characters while following a predefined trajectory (Figure 4.15).

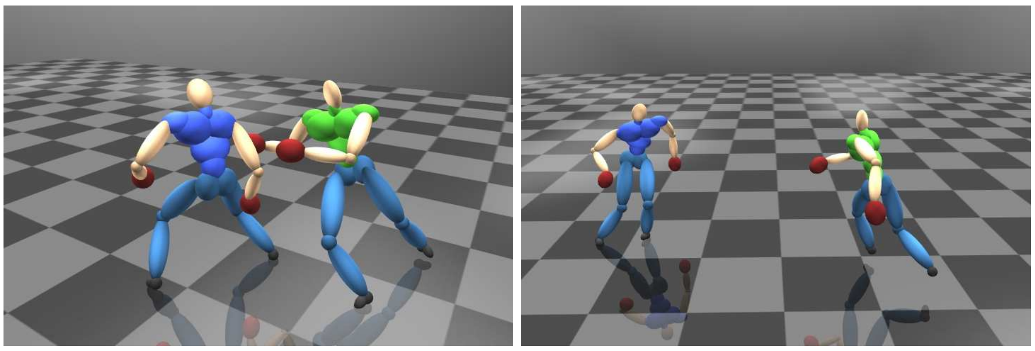

Figure 4.13: The green character chasing and catching the blue one. In this case, they have similar intelligence levels, and hence the green character can rarely catch the opponent.

We also simulated a scene where two characters chase one character. In this case, the game tree is composed of nodes and edges which represent the actions of three characters. The score of the each action is computed based on the status of two characters. The score of the chaser is computed by the chaser's action and the current status of the character running away. The score of the character running away is computed by its action and the status of the chasing character that is closer to it. When evaluating the leaf nodes of the game tree, the scores of edges by the chasers are summed. As a result, the chasers cooperate with each other to catch the character that is running away (Figure 4.16).

	$\mathbf{w}_\theta$	$\mathbf{w_r}$	$\theta_\mathbf{d}$	$\mathbf{r_d}$	$\mathbf{w_D^+}$	$\mathbf{w_D^-}$	$\mathbf{w_S}$
General Boxer	10^1	10^1	0°	$0.8m$	10^5	10^5	10^2
Infighter	10^1	10^1	0°	$0.5m$	$10^5/10^1$ †	10^5	10^2
Outboxer	10^1	10^1	0°	$2.0m$	$10^1/10^5$ †	10^5	10^2
Boxer (Path)	10^1	10^1	0°	$0.8m$	10^5	10^5	10^2
Chaser	10^1	10^1	0°	$0.1m$	10^5	0	0
Runaway	10^1	10^1	0°	$3.0m$	0	10^5	0
Chaser (Path)	10^1	10^1	0°	$0.1m$	10^5	0	0
Runaway (Path)	10^1	10^1	0°	$3.0m$	0	10^5	0

† *The parameters used for short range and long range attack respectively*

Table 4.1: The parameter used in the competitive function to simulate various effects

	$\mathbf{w}_\gamma$	$\mathbf{w_p}$	$\mathbf{w_a}$
Boxer (Path)	0	10^1	10^6
Chaser (Path)	0	10^1	0
Runaway (Path)	0	10^1	0

Table 4.2: The parameter used in the cooperative function (Unlisted simulations do not require the cooperative function)

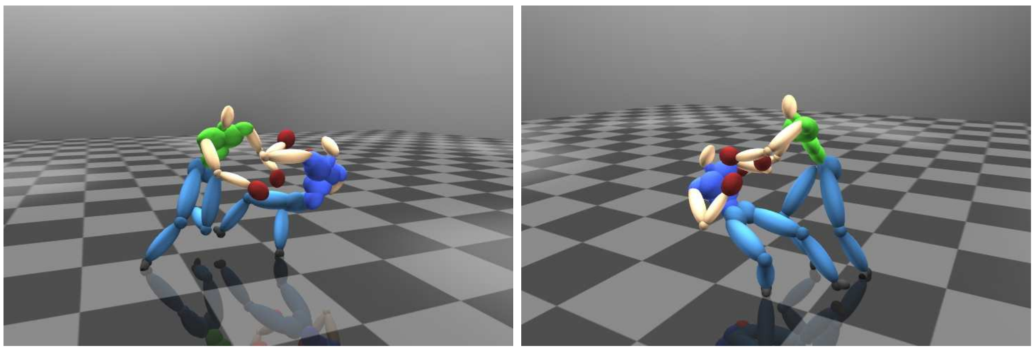

Figure 4.14: The green character chasing and catching the blue one. The green character is set of have a higher intelligent level, and hence is able to catch the opponent quickly.

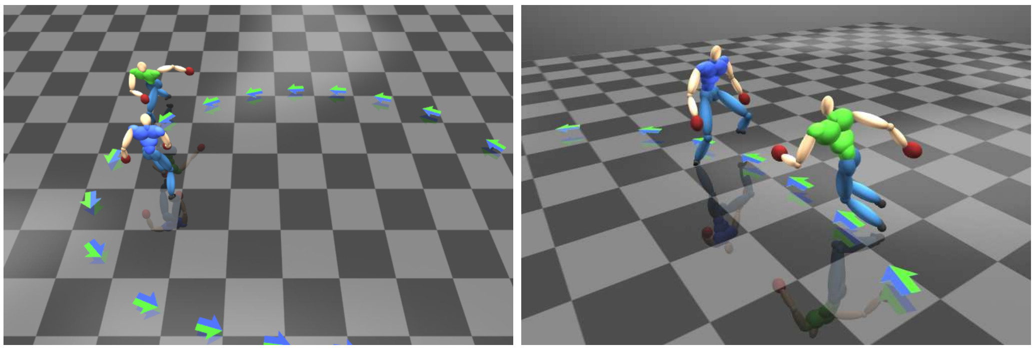

Figure 4.15: The green character chasing and catching the blue one. The catching and running away behaviors correspond to the competitive function, while the behaviors to follow the path correspond to the cooperative function.

4.6 Discussions

4.6.1 Action Evaluation

The action combination term (f^{comb}) in Section 4.4 is currently defined as an element of the competitive function (F^{comp}). This is because the term is used to model the attack-defence relationship, and we wish the smarter character to perform better defensive motion. However, in case the animator wants the two characters to cooperatively perform nice-looking interactions, the term has to be moved to the cooperative function (F^{coop}), otherwise the two characters will prevent their opponents to act as indicated in the action combination table.

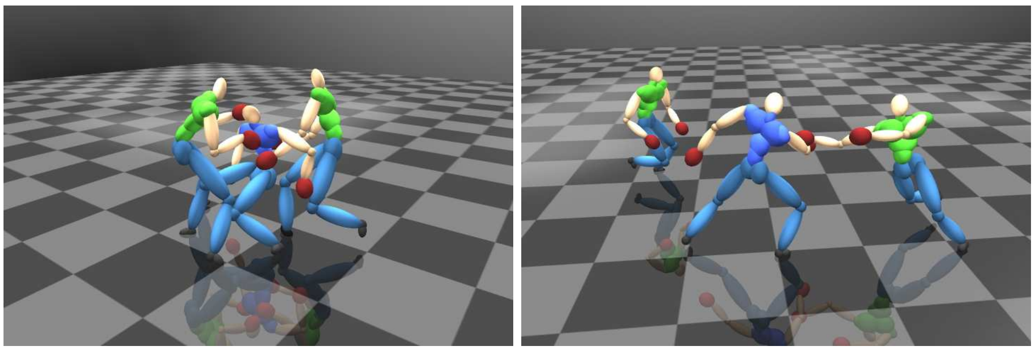
Figure 4.16: The two green characters chasing and catching the blue one. Since the score of the two green characters are summed, they form a team and catch the blue character cooperatively.

4.6.2 Game Theory Related

In our model, we assume each character has perfect knowledge on its opponent in terms of the opponent's strategy and action evaluation functions. Due to such knowledge, the min-max search always gives optimal results. However, if the knowledge on the opponent is incomplete and inaccurate, opponent-model search [Carmel & Markovitch (1996)] and probabilistic opponent-model search [Donkers et al. (2001)] may perform better. Opponent-model search can improve the performance on decision making by building a profile on the opponent based on a history of previous moves [Donkers (2003)]. On the other hand, probabilistic opponent-model search assumes the actual profile of the opponent to be a mixture of several predefined profiles [Riley & Veloso (2006), Donkers et al. (2004)]. Both methods take advantages on the observation on the opponent and select the actual optimal actions. For example, in a chess game, the min-max search considers experienced and novice opponents in the same way [Shannon (1988)], while opponent-model search and probabilistic opponent-model search take into account the mistakes that are made by the opponents [Donkers et al. (2001)].

To implement opponent-model in our system, one simple approach is to observe the actual moves made by the opponent. Then, during the tree expansion process, we determine the probabilities of the opponent to launch its actions based on the observed history, and evaluate the opponent nodes with the expectation values. However, we prefer min-max rather than opponent-model in the our system. This is because the temporal tree expansion method is slow and hence its application is limited to computer animation rather than computer games. Opponent-model is useful in computer games, where the opponent is a human character, and the computer controlled characters need to adapt to the behaviors of the human character. However, in computer animation, we wish the characters to be well controlled and acts consistently. The behaviors of the characters are normally predefined by the animators, and hence the opponent-model cannot benefit the system.

Nash equilibrium is considered to be a solution for most game theory based system. In such equilibrium, all players in the game cannot obtain better rewards by changing their control strategies. It is proved that there exists at least one Nash equilibrium point in any game with a finite set of actions [Neumann & Morgenstern (1944), Nash (1951)]. While a lot of researches focus on computing the equilibrium points in a game [Avis et al. (2010), Solan & Vieille (2010)], we try to avoid approaching equilibrium in our system. This is because when two characters are close to an equilibrium point, they are very likely to stay at the point for a long duration, which leads to a monotonic animation. One example of the equilibrium in a fighting game is that both characters defense themselves forever, which is obviously not what we wish to see [Thore Graepel (2004)]. Fortunately, since each action in our system has specific attacking and defending points with different durations, the state space in our system is highly irregular. This reduces the chance that the characters stay at an equilibrium point. Furthermore, we give penalty to actions that are recently used. With such a penalty function, even if the characters reach an equilibrium point, the point will shift to other locations in the next time step.

4.6.3 Usage Complexity

The proposed algorithm is easy to use. Still, animators are expected to have basic understanding on the parameters that affect the tree expansion process, such as the depth of the game tree and the parameters in the objective functions, in order to generate the required style of animation. One advantage of the system is that it is based on short horizon optimization. In other words, once the parameters are set, the results can be generated almost immediately. Therefore, it is possible to tune the parameters by trial and error.

Currently, we implemented functions to monitor the collisions of characters. Such information is used in the objective function to determine valid attacks and defenses in a fight, as well as valid catches during chasing. We also have functions to monitor the position and orientation of the characters, which are used to determine the characters' movements. Although these monitoring functions cover a large area of interactions, some animators may wish to define other means to evaluate the actions. In such a case, they need to implement additional monitoring functions, and integrate them into the objective functions. This will require in-depth understanding on the system.

Designing the trajectories of movement is straight forward. We provide animators with functions to construct simple trajectories such as straight lines, arcs and circles. The animator can then adjust the trajectories and insert action requirement constraints if needed. Our system can highlight overlapping area among trajectories by considering both spatial and temporal information of the checkpoints, and give the animators a better understanding on what will happen.

The proposed algorithm is fully automatic except adding semantic tags to classify the actions. Such tags are necessary for coupling the attacks and defenses, and usually require the knowledge from specialist in the field. If we have a number of tagged actions already, this process can be automated as the newly captured actions can be tagged based on the similarity to existing actions.

4.6.4 Limitations

There are also some drawbacks in our system. Firstly, the invalid combinations for pruning the sub-tree during the temporal expansion must be determined by the expert who knows the nature of the interactions well. Secondly, we cannot currently handle continuous contacts such as those appearing in wrestling. However, such continuous contact does not happen often in martial arts such as Karate, kickboxing, Taekwondo, or other sports such as basketball, soccer or rugby. In other words, our method can be applied for most competitions.

4.6.5 Computational Cost

The process of expanding the game tree takes up a major part of the computational power. Fortunately, this process can easily be broken down into multiple parallel processes. More specifically, we can implement a multi-thread system with each thread expanding a sub-tree of the whole game tree. As multi-core processer becomes cheap and popular, the performance of our method can be greatly enhanced.

When applying our method to generate an animation of mass crowd fighting, expanding the game tree for all the characters in a single tree is computational costly and quite a waste as characters far away cannot actually interact. We can handle such cases by first finding out the small of group of people having interactions and expand different game trees for each group. We can monitor and switch in and out the members of the group in case the distance from each other becomes smaller or larger.

4.7 Summary

In this chapter, we presented a method to simulate competitive scenes in which multiple characters are densely interacting with each other using singly captured motions. We proposed a method called temporal expansion approach to determine the strategy of the character. We showed that various styles of fighting and chasing can be created by changing the parameters of the game tree such as its depth and the evaluation function. We embedded the cooperative functions into the min-max framework such that the characters can follow high level instructions cooperatively when competing with each other. As a result, we can create intelligent characters that can compete well and be controlled easily.

Chapter 5

Interaction Graph

Intelligent computer-controlled characters are essential in computer games and animation. In many computer games, users can usually control a character to interact with other computer controlled characters. The intelligence of the computer controlled character is important as it can affect the quality of the game. On the other hand, background characters in computer animation are also usually controlled by the computer. If their movements are unrealistic due to their poor intelligence, the animator needs to manually edit them, which will result in a huge amount of extra cost. In theory, the temporal tree expansion method in Chapter 4 is a perfect solution to generate characters with realistic interactions. However, in practice, the algorithm is too computational costly to be used for real-time applications such as computer games. Moreover, if there are a huge number of characters in the scene, the time required to plan the movements for all characters will be very long even as an offline process.

Traditional techniques such as decision trees and flocking have been used to control such characters. However, those techniques can only generate reactive movements, and cannot realize strategic movements that benefit the characters in the future.

Reinforcement learning enables real-time optimal control of characters. It has been used to control pedestrians to avoid other obstacles or characters walking in the streets [Ikemoto et al. (2005), Treuille et al. (2007)], control a boxer to approach and hit the target [Lee & Lee (2004)], make the transition of actions by the user-controlled character smooth [McCann & Pollard (2007)], and train a computer controlled fighter in computer games [Thore Graepel (2004)]. However, there are two problems that we face when we try to use reinforcement learning to control human characters intelligently when they are interacting with another character.

First of all, the state space is too large. The state space increases exponentially with respect to the number of parameters. Parameters such as the action the character is undertaking, its body position and orientation, and the timing to launch the action are going to form the state space. The number is going to be doubled if there are two characters. As a result, it is difficult for existing adaptive learning techniques such as Q-learning [Watkins (1989)] to explore the whole state space to search for optimal policies.

Another problem is that the way the people behave change according to various factors such as their mood, habits, and preferences of actions; however, previous animation techniques used “on-policy” [Sutton & Barto (1998)] reinforcement learning methods, which require the system to be retrained in case the reward function is changed. For example, in boxing, there are boxers called infighters who prefer to fight aggressively in short distance, and use punches such as upper cuts and hooks more. On the contrary, there are outboxers, who prefer to stay away from the opponent and as a result, prefer to use straight punches which are effective in long distance. If we train a virtual boxer by an on-policy reinforcement learning approach, it will not be able to compete well with other fighters who have different styles of fighting. The system needs to be pre-trained for various types of fighters, and the policy needs to be switched according to the type of the opponent, which will be very computationally costly.

In this research, we make use of the fact that the subspace of meaningful interactions is much smaller than the whole state space of two characters. We efficiently collect samples by exploring the subspace where dense interactions of the characters exist and favoring samples which have high connectivity with other samples. Using the samples collected, a finite state machine (FSM) called Interaction Graph is composed. In order to control the character in an optimal way, a min-max search / dynamic programming is conducted on the Interaction Graph.

Our character controlling policy is close to the optimal policy, although we plan actions only on a subset of the whole state space. As the way the space is explored is independent of the reward function for strategy making, we can also recompute the policy of the characters in run-time based on the user’s preference. We can simulate various activities by two characters such as fighting, chasing, playing sports, or carrying luggage together. Our method can plan strategic movements for Non-Player Characters (NPCs) in 3D computer games. For example, we can control virtual fighters in boxing games, or the background crowd moving or fighting with each other in computer animations, or characters collaboratively working, such as carrying a box.

5.1 Contributions in This Chapter

- We propose a new off-policy learning approach that can sample a huge state space by using criteria that favor states with good connectivity and more interactions.
- We propose a finite state machine called the Interaction Graph to precompute the optimal actions for a character to collaborate or compete with another character intelligently.

5.2 Outline of the Method

The outline of Interaction Graph is shown in Figure 5.1. It consists of five steps:

1. Capture the motions of a single person conducting the target motion.
2. Generate the action level Motion Graph structure out of the motion data.
3. Explore the combined state space of two characters by simulating the interactions of the two characters and expanding the game tree.
4. Generate the Interaction Graph of the two characters and find the most appropriate movements of the characters at each node by dynamic programming or min-max search.
5. At each state, the corresponding character selects the precomputed optimal action. If the animator/user wants to change the policy/strategy of the control, the information in the lookup-table is recomputed by re-running dynamic programming or min-max search. This can be done in a few seconds, and can be run in background while simulating the interactions.

Steps 1 to 4 are precomputed during the preprocessing stage, while step 5 is done during run-time. Since the optimal policy to control the characters is precomputed, the run-time overhead is very small. Hence, our system can simulate interacting characters in real-time.

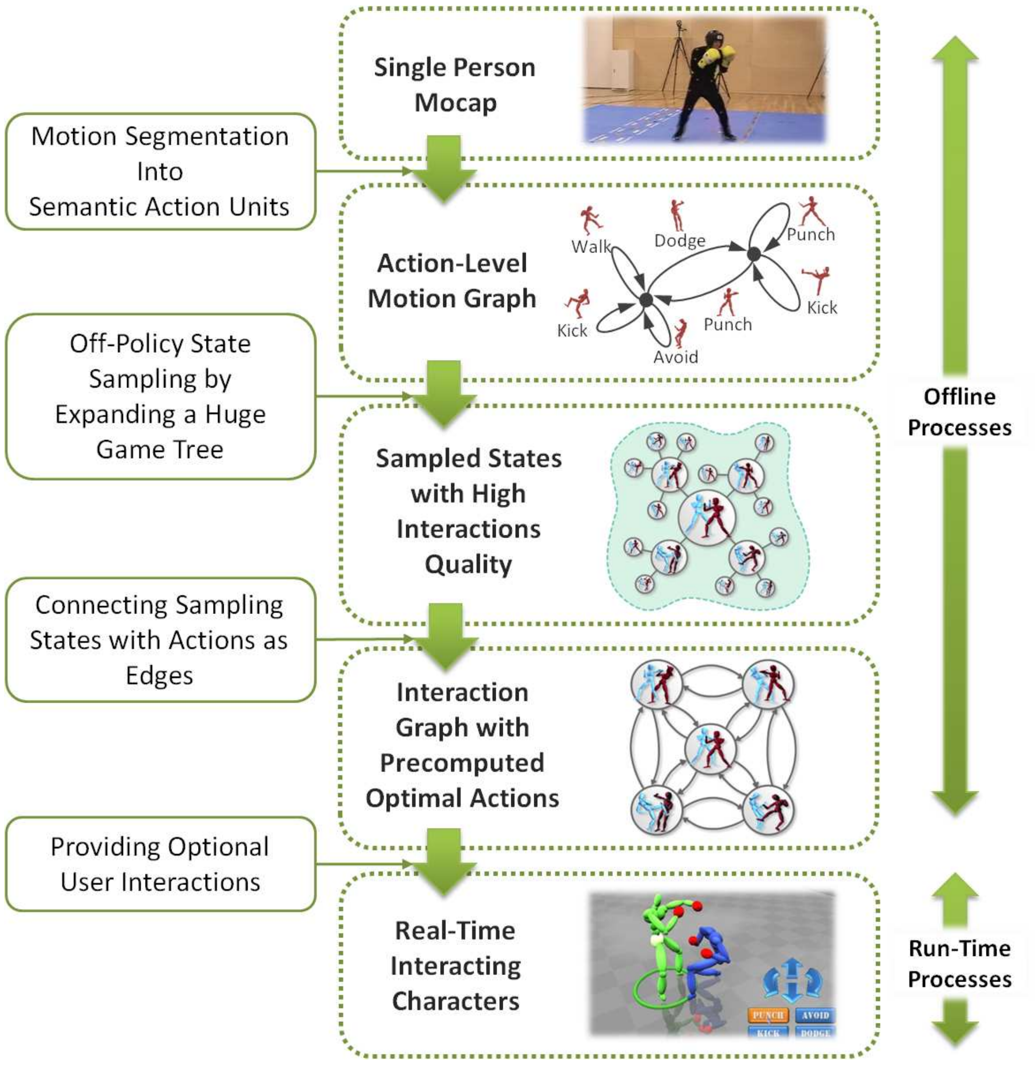
Single Person Mocap
Motion Segmentation Into Semantic Action Units
Action-Level Motion Graph
Walk
Dodge
Punch
Kick
Punch
Kick
Avoid
Off-Policy State Sampling by Expanding a Huge Game Tree
Sampled States with High Interactions Quality
Offline Processes
Connecting Sampling States with Actions as Edges
Interaction Graph with Precomputed Optimal Actions
Providing Optional User Interactions
Real-Time Interacting Characters
PUNCH
AVOID
KICK
DODGE
Run-Time Processes

Figure 5.1: The outline of the Interaction Graph.

5.3 Sampling the State Space

In this section, we explain how we explore and collect sample data in the state space of two interacting characters.

5.3.1 State Representation

Here we explain how the status of two interacting characters is represented; the approach is general enough so that any kind of interactions can be applied. The state space we consider here is composed of statuses when either character has finished an action and is about to start a new action. It can also be statuses when an action of the character is interrupted by the other character and is forced to start a new action.

Suppose we define the two characters by A and B. We express the state when character A is about to select the next action by the following vector: $I^A = (r, \theta_a, \theta_b, Next(M_a), M_b, F_b)$ where r is the distance between the two characters, θ_a and θ_b are the angles made between the facing direction of character A and B, and the line connecting the two, respectively, M_a is the action just finished by character A, M_b is the action character B is currently undertaking, $Next(M_a)$ is the set of actions that can be launched after M_a, and F_b is the frame number character B is at in action M_b (Figure 5.2). The facing direction of each character is determined by its orientation of the head. We can define a state where character B is about to select an action in the same way: $I^B = (r, \theta_b, \theta_a, Next(M_b), M_a, F_a)$, where the variables are defined in the same way as those of character A.

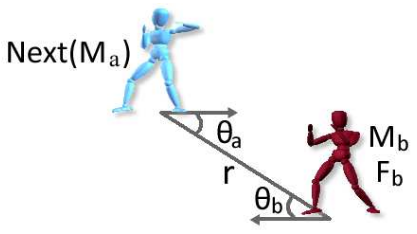

Figure 5.2: Elements of the state that represent the relationship of the two characters interacting

5.3.2 Data Sampling

The data samples are collected by simulating the interactions of the two characters as explained in Section 4.3, and saving the state samples of I^A and I^B. During the simulation, when a character ends its action, it selects an action among the next available actions in $Next(M_{\{a|b\}})$. Instead of continuously simulating the interactions along the time line as done in other adaptive reinforcement learning approaches, we collect the samples by selectively expanding the game tree (Figure 5.3). The nodes of the tree represent the states such as I^A and I^B, and the edges represent the actions that can be chosen by the character.

The most important issue in this research is how to select a sample in the game

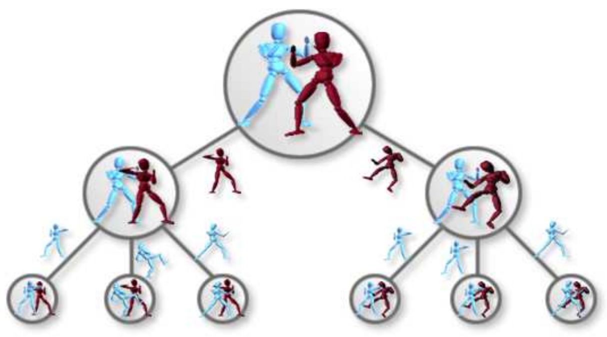

Figure 5.3: An expanded game tree. The nodes represent the states of interaction when either fighter launches new actions. Each edge represents the action that has been selected by the fighter.

tree, and how to further expand the tree. We need to explore the space well enough to find the policy that can give the optimal solution to the character. On-policy methods such as SARSA [Rummery & Niranjan (1994)] have been used to train the computer controlled characters [Thore Graepel (2004)]. In on-policy methods, the samples collected are dependent on the reward function. Such methods will selectively explore the subspace which gives better rewards. For the simulation of two characters interacting, the user might prefer to change the behavior of the characters by adjusting the reward function. In such a case, if we use on-policy approaches, we need to retrain the system all over again. Instead, we use an off-policy approach here, in which the way we explore and collect samples is independent of the way the actions are rewarded. Using such an approach, the samples obtained can be used for different reward functions. As a result, we can change the behavior of the human character controlled by the system during run-time.

However, we cannot use well-known off-policy approaches such as Q-learning [Watkins (1989)], in which the selection of the action is strongly affected by a random factor. This is because the state space is too large. In this research, we explore the state space in a specialized way for the interactions of two characters. We use an off-policy approach in which the criteria of selecting the space to explore are independent of the reward function. We need to explore the subspace that includes a lot of meaningful interactions between the characters.

Starting from an appropriate state, we expand the game tree. We favor states which (1) have high connectivity with other samples, and (2) results in dense interactions of the characters.

Criterion (1) can be evaluated by counting the number of edges going out from the state that redirect the characters to existing states in the state archive. Let us define this number by J_1. We adopt criterion (1) because we require a graph with high connectivity to increase the controllability of the character. A state of low connectivity is less useful as the chance to visit it is low and it is difficult to get back to other important states. If the number of states with low connectivity increases, the character needs to pass through a lot of extra states to finally start an effective action. This drops the performance of the character and eventually the animation will also appear unnatural.

Regarding criterion (2), the way to evaluate the quality of interactions between the characters must be defined by the user: this can be simple if we are handling activities in which objective is clear. For example, it is easy to define the quality of collaborative interactions such as carrying luggage together or competitive interactions such as fighting. The first way to evaluate an interaction is by setting constraints. For example, when carrying luggage together, the two characters must not be too far away from each other as the luggage will fall down onto the ground. Actions leading them to get too close also cause problems. Therefore, thresholds to keep the distance between the two characters are set. Actions that violate such constraints can be considered as unsuitable and can be given low scores. For competitive actions such as boxing, the objective is to hit each other, or block / avoid those attacks which will otherwise cause the character to fall down. For boxing, we evaluate the amount of dense interactions by the sum of attacks in the reaching distance and effective defence actions by both characters. Let us define the amount of interaction by J_2.

At every state visited, we compute $J_1 + J_2$ of all the actions that can be launched. The children nodes are sorted based on the score, and the top 30% nodes will be further expanded. All the nodes explored in the game tree are saved in the sample archive.

The exploration continues until either the number of samples in the archive reaches a maximum limit, or until the number of newly created samples that do not duplicate with those already in the archive becomes smaller than a predefined threshold.

5.4 Interaction Graph

Based on the samples collected in the previous section, we compose a FSM whose states represent the interactions of two characters. We call this FSM the Interaction Graph. Once the Interaction Graph is composed, by defining the objective function that evaluates the action chosen by each character, we can search for the optimal action by using dynamic programming or min-max search. It is also possible to change the way each character behaves by editing the reward function and recomputing the policy in run-time.

In the following subsections, we explain about composing the states of the Interaction Graph based on the collected samples, connecting the states by edges, and finally searching for the optimal action at every state on the graph.

5.4.1 Creating States of Coupled Actions

After producing sufficient samples of interactions, the similar ones are grouped together using K-mean, and the states of the Interaction Graph are produced. A distance function D is defined as follows to calculate the difference between two samples $I_i = (r^i, \theta^i_a, \theta^i_b, Next(M^i_a), M^i_b, F^i_b)$ and $I_j = (r^j, \theta^j_a, \theta^j_b, Next(M^j_a), M^j_b, F^j_b)$:

$$D(I_i, I_j) = \begin{cases} \frac{|r^i - r^j|}{\sigma_r} + \frac{|\theta^i_a - \theta^j_a|}{\sigma_\theta} + \frac{|\theta^i_b - \theta^j_b|}{\sigma_\theta} \\ \text{if } Next(M^i_a) = Next(M^j_a)\ ,\ M^i_b = M^j_b \text{ and } |F^i_b - F^j_b| < F'_e \\ \infty \text{ Otherwise} \end{cases} \tag{5.1}$$

where σ_r and σ_θ are constants to normalize the effects of the distance and the rotation angles respectively, and F'_e is a threshold value.

Empirically, we found that $\sigma_r = 0.5m$, $\sigma_\theta = \pi/6$ give good results. A state of the Interaction Graph is produced for every clustered group, which is represented by the average sample (Figure 5.4).

One important issue is how to determine the tolerance for grouping samples. If this value is too small, there are going to be too many states, and if this value is too large, there are going to be too many artifacts, such as foot sliding, sudden rotation of the body, and fast transition from one posture to another. Since foot sliding is the most noticeable artifact, we determine the threshold in a trial and error manner based on the amount of foot sliding that can occur.

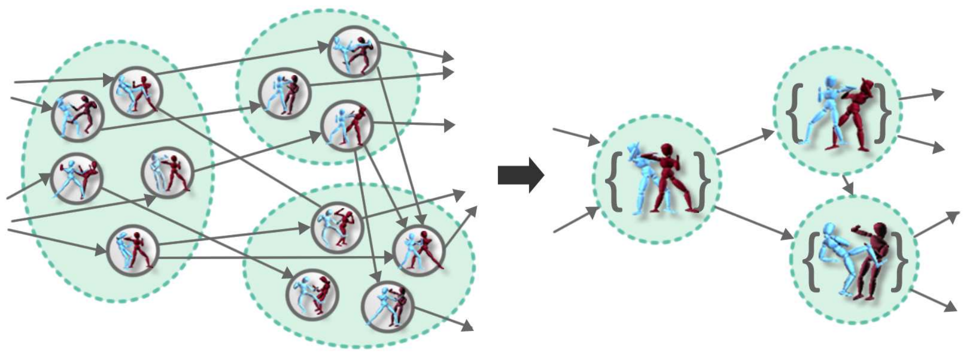

Figure 5.4: Grouping samples to form nodes in the Interaction Graph. Using the samples created by tree expansion, the nodes are composed by grouping similar samples.

5.4.2 Creating the Edges of the Interaction Graph

An edge in the Interaction Graph represents an action performed by the character, and points from the original state before the action to the resultant state after the action. Recall one of the elements in the state, $Next()$ is the set of next available actions by the corresponding character. For every action in $Next()$, as a result of launching it, we might arrive to another state in the Interaction Graph. If that happens, the two states are connected by an edge. However, since the graph does not cover the whole state space, there is a chance that no state is available in the graph after an action is launched. In that case, the edge is not generated, and the character will not launch such an action at the state. After scanning through all the $Next()$ actions in all the states and linking the states by edges, the composition of the Interaction Graph is completed (Figure 5.5).

5.4.3 Search on the Interaction Graph

Once the Interaction Graph is created, we can do strategy-making by using dynamic programming or min-max algorithm. If the two characters are collaboratively working, they will select actions that maximize a common return function. Such a problem can be solved by dynamic programming. In case of competitive activities such as boxing, each character will try to maximize its own return value and minimize the opponent's return value. Such a problem can be solved by min-max search. In either case, we first need to define the reward function that evaluates the individual interaction. The rewards for each action at each state are

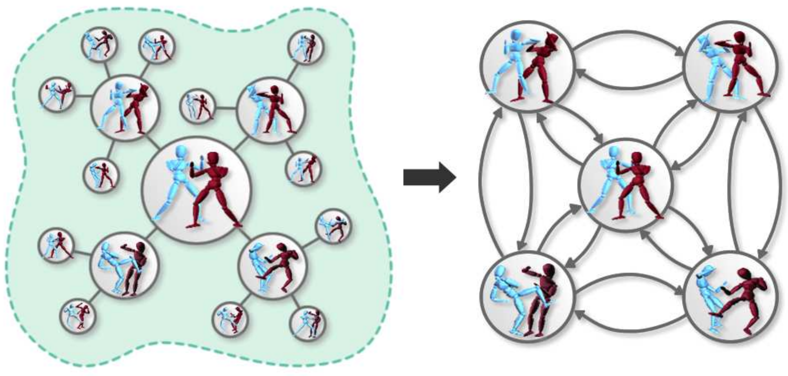

Figure 5.5: Creating the edges of the Interaction Graph: (Left) the state sampled by tree expansion in the state space (Right) the Interaction Graph generated by connecting the sampled states with actions as edges

precomputed, and therefore, its computational cost does not affect the run-time performance.

To simulate two characters intelligently interacting with each other, we need to find the optimal policy to control the characters. More specifically, at each time step i, suppose the character selects an action and gets a **reward** defined by r_i. The **optimal policy** π offers an action at every state that maximizes the following **return value**:

$$R = \sum_i \gamma^i r_i \tag{5.2}$$

where γ is called the **discount factor**, with the range defined as $0 \leq \gamma \leq 1$.

Let us define the reward function for carrying luggage as an example of collaborative activities. If we want the characters to proceed to a specific direction with respect to their average facing direction, we can compute the reward r_ϕ as follows:

$$r_\phi = w_\theta \theta^2 + w_d(d - d_p)^2 + w_v(v - v_\phi)^2 \tag{5.3}$$

where θ is the relative orientation of the carriers with respect to their partners, d is the distance with the opponent, d_p is the desired distance between the two, v is the average velocity of the two characters, v_ϕ is the preferred average velocity of the two characters that define the desired direction of movement, and w_θ, w_d, w_v are the weight constants for each term. We compute r_ϕ for eight different directions, as we would prefer to interactively control the two characters to different directions during runtime. Using the reward function, we can compute the return value at each state by Equation 5.2 using dynamic programming.

For competitive interactions such as boxing and chasing, we use the simplified version of the objective functions defined in Section 4.4. Let us briefly review the reward function that evaluates the action of fighters. It is composed of three criteria: (1) the relative orientation (θ) and distance (r) of the fighter from the opponent, (2) the damage the fighter has given to the opponent (D^+), and (3) the damage the fighter received from the opponent (D^-). The reward we use has the following form:

$$r_{box} = w_\theta \theta^2 + w_d (d - d_p)^2 + w_D^+ D^+ - w_D^- D^- \tag{5.4}$$

where d_p is the preferred distance by the fighter, D^+ is the damage that the fighter has given to the opponent, D^- is the damage received, and $w_\theta, w_d, w_D^+, w_D^-$ are the weight constants for each term.

In case of competitive activities, we do min-max search instead of dynamic programming, on the Interaction Graph. Let us explain how to do such a min-max search on the Interaction Graph. Assume $W_{i,j}$ to be the reward of the j-th transition going out from state i and is computed by the reward functions (Equation 5.4 for fighting). The value tells us how much the character earns or loses by launching the $j-$th action at state S_i. For all state-action pairs, $W_{i,j}$ are precomputed so that there is no need to evaluate the interactions of characters during run-time. Suppose the best score the character can obtain from state S_i as a result of searching d levels ahead is $V_{i,d}$, and we know $V_{i,d}$ for all the nodes already. The value of $V_{i,d+1}$ can be computed by checking all the out-going edges from state S_i, sum the reward of the edge $(W_{i,j})$ with the return value of child state for depth d, and find out the edge that returns the largest (if it is a max node) or smallest (if it is a min node) value (Figure 5.6):

$$V_{i,d+1} = \{\max/\min\}(W_{i,j} + V_{s(i,j),d}) \tag{5.5}$$

The ID of the best next state is saved in $E_{i,d+1}$. The pseudo code of this procedure is shown in Algorithm 1.

The computational cost for finding the optimal path for min-max search is $O(N \times D)$, where N is the total number of states in the FSM and D is the maximum depth of the min-max search.

All $E_{i,d}$, which are the best states to transit to next at each state, are saved in a look-up table. This min-max computation has to be done only once for the

Algorithm 1 Min-max on the Interaction Graph

```
/* Initialize the score of 0-depth to 0 */
for $i = 0$ to $N-1$ do
    $V_{i,0} = 0$
end for
/* Evaluate the optimal action */
/* For each depth of search */
for $d = 0$ to $D-1$ do
    /* For each state $S_i$ of the FSM */
    for $i = 0$ to $N-1$ do
        /* Scan the children of $S_i$ */
        /* $s_{ij}$ is its index of the $j$-th child of $S_i$ */
        if $S_i$ is a max node then
            $V_{i,d+1} \Leftarrow max_j\{V_{s_{ij},d} + W_{i,j}\}$
            $E_{i,d+1} \Leftarrow s_{ij}$ s.t. $max_j\{V_{s_{ij},d} + W_{i,j}\}$
        else { $S_i$ is a min node }
            $V_{i,d+1} \Leftarrow min_j\{V_{s_{ij},d} + W_{i,j}\}$
            $E_{i,d+1} \Leftarrow s_{ij}$ s.t. $min_j\{V_{s_{ij},d} + W_{i,j}\}$
        end if
    end for
end for
```

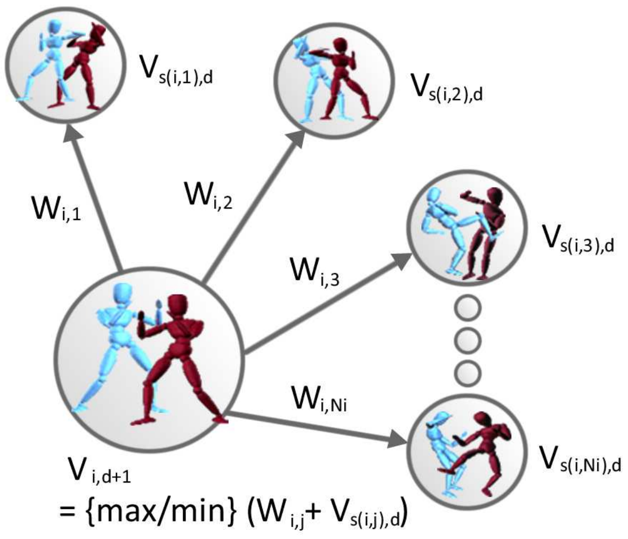

Figure 5.6: Every node has a table which keeps $V_{i,d}$, which is the best score the character can obtain from state S_i as a result of searching d levels ahead. The number of out-going edges from S_i is N_i, the ID of the state on the other end of the j-th edge from state i is s_{ij}.

whole graph unless the evaluation function is changed. This happens only when the user wants to change the parameters or sub-functions in Equation 5.4. Even if that is the case, it can be done in a very short time as all the parameters required to recompute them are embedded in the data structure of the transition.

5.5 Experimental Results

We have simulated scenes of fighting as examples of competitive interactions and scenes of carrying luggage as examples of collaborative interactions. We have captured the shadow boxing of a boxer for 2.5 minutes, and motions to carry objects for 2.5 minutes. Each motion was segmented into 137 and 167 actions, respectively, and was classified into different groups. A simulation based on the game tree expansion was first done to compute and collect the samples. Using the obtained samples, Interaction Graphs of different categories were created.

As in the previous chapter, a particle system was used to handle unexpected collisions among the characters and adjust the actions performed for better visual effects. Further information can be found in Appendix A.

5.5.1 Competitive Interactions: Kick Boxing

We created a motion database of kick boxing. Then, we generated an Interaction Graph of 79,855 states and 3,392,297 edges. The construction of the Interaction Graph took around 180 minutes using a Pentium 4 Dual Core 3.0 GHz CPU and 2 GB of RAM.

5.5.1.1 Game-Style One-to-One Fighting

In order to show the real-time performance of our system, we have implemented a game-style interface which the user can control a character to fight with the computer controlled character (Figure 5.7). The user can give high level commands such as “move forward / backward ”, “turn left / right ” “punch”, “kick”, “dodge”, and “avoid” to the character; the best action that belongs to such categories are selected based on the Interaction Graph. The action of the opponent character is selected based on min-max search in the graph. All these searches can be undertaken just by looking up the table, and therefore, the computer controlled character can react in real-time.

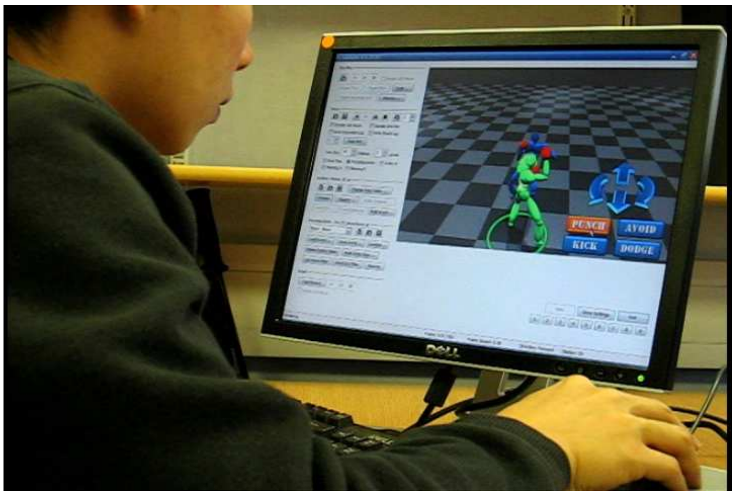

Figure 5.7: Using the game style interface, the user can control a character to fight with the computer controlled character by the Interaction Graph.

A screen shot of such a scene is shown in Figure 5.8. The strength of the computer controlled character can be adjusted by the depth of the search on the Interaction Graph; if we want to keep it weak, we can set it to 1, which means the computer controlled character will only select the action with the best immediate effect. It can be made stronger by increasing the depth of the search.

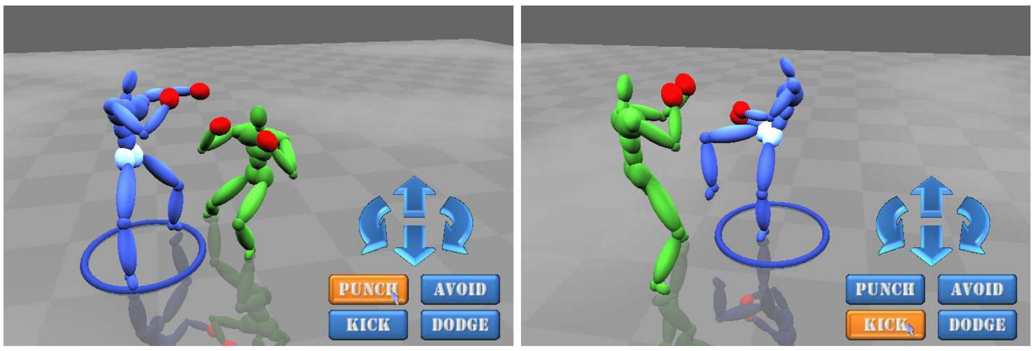

Figure 5.8: Using the high level commands, the user interacts with the computer controlled character based on Interaction Graph.

5.5.1.2 Multiple Characters Fighting

We can easily increase the number of boxers and create a scene where many characters are fighting with their opponents (Figure 5.9). The characters are split into two teams, and each character fights with the closest character in the opposite team.

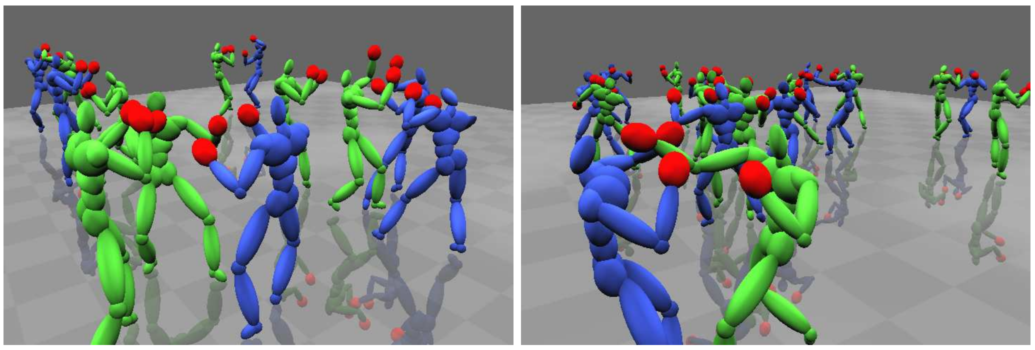

Figure 5.9: Using the Interaction Graph, we can simulate a lot of characters fighting with each other in real-time.

During the simulation, sometimes other boxers get closer than the opponent the boxer is currently fighting with. In such a case, the boxer switches the opponent. Therefore, in some cases there are scenes where one character appears to be fighting with two or more characters. Also, there are cases that multiple characters fight with the same enemy due to the lack of enemies. Each of them will therefore consider the enemy is fighting solely to itself. However, since the state space only defines two characters, the enemy will only decide its action based on one of its many opponents, which is the closest one.

5.5.1.3 Changing Fighting Styles During Run-Time

In order to show the effect of changing the reward function during run-time, we have enabled the computer controlled character to change the style of fighting while fighting with the user-controlled character. When the style of fighting is switched to outboxing, if the user character approaches to the computer character, it will step backward or to the side to keep the distance between the two. On the other hand, when the computer character's style is switched to infight, it becomes more aggressive and always tries to hit the user fighter. Snapshots are shown in Figure 5.10 and Figure 5.11. Regarding the parameters of the reward function, we used the same values as those in Chapter 4.

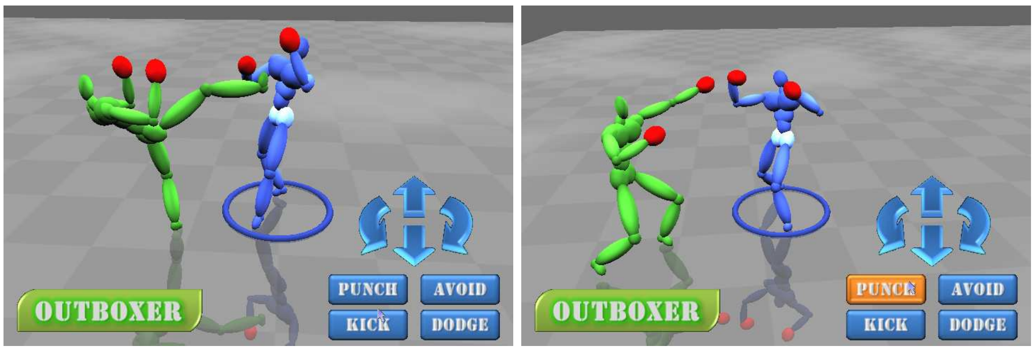

Figure 5.10: Computer controlled character fighting with outboxing style

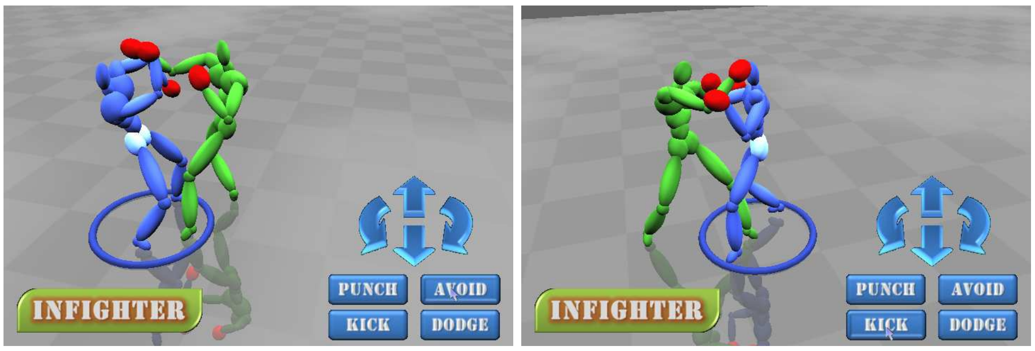

Figure 5.11: Computer controlled character fighting with infighting style. Fighting Styles can be switched during run-time.

The computation of new rewards for different fighting styles takes merely a few seconds. We can compute the new rewards using a background thread whenever the user requests a change on style. Alternatively, we can precompute two Interaction Graphs of different reward functions, and switch between the graphs during run-time.

-	Human	Static	Game Tree 2	Game Tree 3
Interaction Graph 1	-15	34	-37	-53
Interaction Graph 3	23	54	14	-19
Interaction Graph 5	28	55	28	-4

Table 5.1: The score table of the matches between the Interaction Graph controlled character and the human player (Human), a character trained based on static objects (Static), and game tree search with expansion of two levels (Game Tree 2) and three levels (Game Tree 3). Each row shows the scores when the depth of the search on the Interaction Graph is 1, 3 and 5.

5.5.1.4 Comparison of Different Controllers

We held matches between a character controlled by the Interaction Graph with three sorts of boxers; a human-controlled character, a computer controlled character trained to hit a static object [Lee & Lee (2004)], and a computer controlled character by game tree expansion with the full character state space. The character trained to hit the static object is considered to be the weakest, as it has no idea of defence. The character based on the game tree expansion is the strongest, as it evaluates all possible combinations by expanding the game tree; however, it cannot select an action in real-time when the depth level is ≥ 3. The results are shown in Table 5.1. The scores are calculated by subtracting the number of successful attacks by the Interaction Graph controlled character from the successful attacks by the opponent.

It can be observed that an Interaction Graph controlled character is already too strong for the human controlled character to compete with when the depth level is over three. The character trained based on the static object also cannot perform well as it does not have any concept of defence. The Interaction Graph controlled character is weaker than that controlled by game tree expansion. The strength of the fifth level Interaction Graph character is about the same as the three level game tree boxer. This is because the game tree boxer's action is based on the precise simulation of the fight. On the other hand, the choice of actions by the Interaction Graph includes quantization error. When the depth of the level increases, the quantization error starts to accumulate. Five levels of expansion is equivalent to a movement of 2.5 seconds in average. In activities such as fighting, each action is very quick and short; 2.5 seconds is already very long ahead in the future compared to the duration of each individual action. Therefore, we can say our results are satisfactory, taking into account that the characters can perform near-optimally by simply using a look up table.

In order to check how many of the important states the Interaction Graph is covering, we have examined how many of the states visited by the game tree expansion approach are covered by the Interaction Graph. If this ratio is high, that means the Interaction Graph has a control policy comparable to that of game tree expansion. The result was 93%, which is very high, considering the actual size of the state space.

5.5.2 Collaborative Interactions: Carrying Luggage

We created a motion database for carrying luggage. In fact, the database only contains the motions of a single character walking around, while the arm movement of carrying luggage is generated by inverse kinematics. An Interaction Graph of 128,804 states and 4,685,246 edges was generated. The construction of the Interaction Graph took around 200 minutes using a Pentium 4 Dual Core 3.0 GHz CPU and 2 GB of RAM.

5.5.2.1 Avoiding Balls While Carrying Luggage

Rewards were set to move the characters to eight locations around the characters, and eight different policies were computed to move to each location. Once the policies are obtained, the user can interactively specify the direction the characters should move, and the optimal action is chosen from the corresponding policy. The user can control the characters to move to arbitrary directions to avoid being hit by balls rolling in random directions. Screen shots of the characters controlled to avoid the obstacles are shown in Figure 5.12.

5.6 Discussions

5.6.1 State Sampling

Due to the exponentially growing size of the state space with respect to the dimensionality, reinforcement learning could not be efficiently applied for dense interactions such as fighting, in which the characters need to take into account

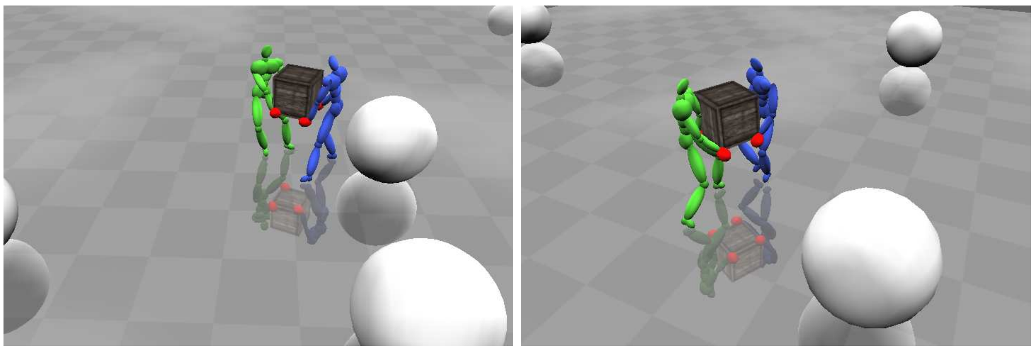

Figure 5.12: Screen shots of the characters controlled to avoid the ball while holding a box.

the full status of the opponent. However, the subspace that meaningful samples exist is biased. By searching samples in the high density subspace, it is possible to compose a FSM called the Interaction Graph that enables intelligent control in real-time.

In our method, as the criteria to determine the subspace to explore is independent of the reward function that evaluates each action, the Interaction Graph can be used for different rewards, and it can be adjusted even during run-time. As a result, we can simulate the movements of different styles of interactions, such as infighters or outboxers who have different preferences for attacks and defences.

5.6.2 Action Evaluation and Selection

The proposed method is deterministic; the action to be selected is determined based only on the min-max score computed over the Interaction Graph. The system can easily be switched to a probabilistic system, as done in previous works [Lee & Lee (2004)]. We can set the probability that the character selects each action according to the min-max score, and use the Russian roulette approach to determine the action. We can also decrease such probability according to the number of times that that action has been selected at that state. For applications such as games, this approach can be a good method to randomly determine the actions of the computer controlled character.

In our experiments, we do not include the discount factor when evaluating the actions at each state. This is because the depth of the search is only up to five levels, and the predictions are reasonably accurate. If a deeper search is required, we can use the discount factor when recursively computing $V_{i,d+1}$ in the Interaction

Graph at Algorithm 1. In such a case, Equation 5.5 will be calculated as $V_{i,d+1} = \{max/min\} \, (W_{i,j} + \gamma V_{s(i,j),d})$ where γ is the discount factor.

5.6.3 Game Theory Related

Since the Interaction Graph can be applied in computer games, there is a demand to model the behavior of the human player and apply opponent-model search [Carmel & Markovitch (1996)] or probabilistic opponent-model search [Donkers et al. (2001)] instead of min-max search. By this way, the computer controlled characters can adapt to different human players. However, as there is a strong relationship between the actions launched and the corresponding states, modeling the human player is tough due to the high dimensional state space. Further research will be required to generalize the history of launched actions in the state space, and model the player with minimal resources.

5.6.4 Comparison to Reinforcement Learning

The Interaction Graph is a machine learning based approach, and has a lot of similarities with reinforcement learning [Sutton & Barto (1998)]. Like temporal different learning, we rely on dynamic programming to evaluate the state values. Moreover, similar to off-policy temporal different learning methods such as Q-learning, we maintain two policies. The behavior policy is used to explore the state space while the estimation policy is used to estimate the state values. The major difference is that in our system, the state sampling process and the state evaluation process are completely independent. We do not revise the estimation policy until sampling is completed. This framework has an advantage on simplicity when we wish to change the estimation policy to model different behaviors.

One popular approach to deal with high dimensional state space is to apply Monte Carlo Reinforcement Learning [Fishman (1996), Bouzy & Chaslot (2006)]. This approach is shown to be effective in Go, where the complexity is up to $10^{3}60$. Instead of explicitly sampling the whole state space in order to evaluate the value of a state, we may randomly assign a actions to such state and observe the average rewards. By repeating this process, it is possible to get an accurate estimation of the state values. Unfortunately, such an approach is not feasible in our system. This is because unlike Go, the action evaluation functions in our system are far more complex, and involve computational costly processes such as the collision detections functions. As a result, evaluating the state value by Monte Carlo approach will take even longer than sampling the state space.

Another popular approach to deal with high dimensional state space in reinforcement learning is to set up a hierarchical system with options [Stolle & Precup (2002)], hierarchies of abstract machines (HAMs) [Parr & Russell (1998)], or the MAXQ framework [Makar et al. (2001), Shen et al. (2006)]. We will discuss these three methods with respect to our problem.

First, reinforcement learning with options creates temporally extended actions. With a set of starting states and a set of destination states, multiple successive actions are combined as an option [Stolle & Precup (2002)]. One example is that when training a robot to navigate a huge room, we may divide the room into different areas, and train the robot to move from one area to another. Each option contains a starting area, a destination area, and a series of actions for the robot to perform. By this way, a complex problem can be decomposed into a set of smaller problems, and the optimal solution of each smaller problem is learned with reinforcement learning. However, in our fighting problem, since the rewards function depends on collision detection of attacking body parts, a small change in state may result in a large change in reward. This creates a highly irregular state space. Using the robot example, our environment is full of tunnels and warping points. It is therefore difficult to effectively divide the state space into systemic areas and apply options.

Second, the hierarchies of abstract machines (HAMs) constrains the actions that an agent can take in each state, and provides a hierarchical means of expressing these constraints at different levels of detail [Parr & Russell (1998)]. For example, when training a robot to navigate a narrow corridor, instead of allowing the robot to move in all direction, we can tell the robot to only move forwards and backwards. By this way, we can effectively reduce the action space. In our system of fighting, the hard constraints serve a similar propose. Instead of allowing the character to do any action, the actual actions a character can do is limited by different conditions defined in the state space. For example, for all states in which the opponent is attacking, the character should do nothing other than defense. Similarly, in those states where the opponent is far away, the character should not attack nor defense.

Finally, MAXQ represents a problem with subtasks and subgoals, and corresponds the values of parent tasks with those of subtasks [Makar et al. (2001), Shen et al. (2006)]. A typical example is to train a robot picking up rubbish and putting the rubbish into rubbish bins in a given environment. Instead of training the robot in the full state space, we can divide the task into subtasks such as navigating, picking up rubbish, and putting rubbish into a bin. Each subtask is trained using standard reinforcement learning, and a global controller is trained to assign the correct task to the robot. In our moving luggage system, we apply similar concepts to decompose a huge task. Instead of training the characters to avoid any obstacles with the luggage, we train the characters to move to one of the eight quantized directions, such that the training becomes a lot simpler. The animator acts as a global controller to assign movement directions to the characters such that they avoid the obstacles.

5.6.5 Usage Complexity

We create simple user interfaces for the animators, such as the game-style control panel, to ease the animation generating process. With these interfaces, it is easy to give high level commands to the characters in the scene. However, if lower level controls are required, our interfaces may fall short. In fact, it is a tough problem to enable the animators to control the whole scene in different level of details, especially in a real-time basis. Such an issue would require further research.

The criteria to determine the quality of interactions during the behaviors of the characters must be specified by the animators. As the animators are only allowed to give an abstract idea of the interaction, this might not be a difficult task. Especially in case the interactions are competitive, an abstract idea of the way the two characters compete can already become a good hint for the criteria.

Once the criteria to evaluate the quality of interactions are specified, generating the Interaction Graph is an automatic process. However, evaluating the quality of the graph is difficult before this process is completed, and the graph requires hours to be built. Thus, it is better to be careful when setting up the criteria. This is a general problem for learning based approaches.

Similar to the tree expansion method in Chapter 4, the animators need to specify how to reward the actions. This again depends on the nature of the interactions, and the animators need to adjust it to obtain a satisfactory scene. Since the topology of the graph is independent of the reward, the animators can interactively edit the function and view the effect in the animation in a few seconds.

5.6.6 Possible Extensions

Our method can be easily combined with existing FSM frameworks to control characters in computer games such as wrestling. In such games, usually there is one FSM for each individual character. We can let the users freely control the characters based on the individually prepared FSM when they are apart from each other, and let them go into the Interaction Graph only when the dense interactions start. Some game designers might prefer to manually design such motions so that they appear plausible. It is also possible to design a coupled state and insert it into an Interaction Graph.

This research can also be extended to simulate the behavior of real humans when they are competing with each other. In scenes of competition, usually the person does not have full knowledge about the opponent, but gradually learns it through the interactions. The person also takes advantage of such a condition by launching fake actions to trick the opponent. By enabling the system to simulate such behaviors, it will be possible to create a virtual reality system that the athletes can use to train their skills and simulate matches with other athletes.

5.7 Summary

We presented a real-time approach to simulate scenes in which multiple characters are densely interacting with each other. We proposed a method to precompute the complex interactions of the characters by favoring states that result in more interactions with the character and that have higher connectivity with other states.

Using our method, it is possible to cope with problems with high dimensionality, such as fighting. Our method can be used to control NPCs in 3D computer games as the optimal action at every state is precomputed. We can even simulate various styles of interactions as the samples collected are independent of the cost function used to select the optimal action.

Chapter 6

Interaction Patches

Scenes of battlefields, panicked crowds and team sports in movies and TV programs involve a huge number of interactions of multiple characters. Existing methods have problems creating such interactions. Manually composing the scene using singly captured motions or keyframed motions requires a huge amount of labor by the animator. Flocking-based methods [Reynolds (1987), Helbing et al. (2000)] have problems simulating close interactions that involve a lot of kinematic constraints. Previous optimization-based methods [Lee & Lee (2004), Treuille et al. (2007)] suffer when creating artistic interactions, as the objective functions are designed just to benefit each character.

The Interaction Graph in Chapter 5 can generate a large number of characters with realistic interactions in real-time. However, the method suffers from two problems. First, due to the limitation in the state space, the Interaction Graph can only simulate interactions between two characters. Although we can extend the algorithm and put more characters in the scene, each character only consider one opponent during interaction at a time. Second, the graph generates a lot of nodes and edges, making it difficult to monitor the quality of interactions, as well as adjust the graph manually. We need an algorithm to model the interactions among a large number of characters effectively.

When we watch fighting scenes in movies, we immediately realize that there are a variety of interactions appearing stylized; artistic and logically clear as if they are designed by an artist. At the same time, we also realize that the patterns of interactions are very simple. For example, in a scene where a main character fights with many background characters, most interactions between them follow the rule of “background character: attack”, “main character: avoid”, “main character: counter attack" and "background character: knocked down".

This observation leads us to develop an algorithm that is flexible enough for the user to design his/her favorite interaction, while sufficiently automated so that the user can create a large-scale animation involving a number of characters with the least effort. Our system simulates the minimal unit of interactions between two characters based on abstract instructions given by the user, and stores the result as structures called Interaction Patches. The Interaction Patches are spatio-temporally concatenated to compose a large-scale scene in which the characters interact with each other, such as one person fighting with many enemies (Figure 6.9), a group of characters falling down onto each other like dominos (Figure 6.11), an American football player holding a ball and escaping from tackling defenders (Figure 6.13) and a group of people passing luggage one to another (Figure 6.16).

Our work is inspired by the idea of Motion Patches [Lee et al. (2006)], where the large-scale scene is composed of building blocks. Using their approach, it is possible to generate an animation where the characters interact with the environment. However, it is not possible to generate an animation where multiple characters densely interact with each other. In this research, we precompute the complex interactions of multiple characters and use them as the building blocks to compose the final scene.

6.1 Contributions in This Chapter

- We propose a method to synthesize realistic interactions between characters by expanding the game tree, based on the pattern of interactions specified by the user. Since the pattern is specified, the number of combinations is small, and we can obtain realistic interactions with a limited amount of computation. These interactions are saved as Interaction Patches to be used during runtime.

- We propose a new algorithm to synthesize a large-scale scene in which the characters densely interact with each other. The precomputed Interaction Patches are spatio-temporally concatenated to compose a large-scale scene.

6.2 Outline of Method

The outline of Interaction Patches is shown in Figure 6.1. It consists of five steps:

1. Capture the motion of a single person using a motion capture system.
2. Create the action level Motion Graph, in which the actions are all annotated.
3. Compose the set of minimal units of interactions, which we call the Interaction Patches, by specifying the pattern of interactions and expanding the game tree.
4. Generate two tables that list how each Interaction Patch can be temporally and spatially concatenated with other Interaction Patches to compose large-scale scenes.
5. Compose a scene by concatenating the Interaction Patches. This is the only online process, which allows the user to optionally give high-level commands and see what they can get immediately.

Steps 1 to 4 are performed during the preprocessing stage. The only run-time process is the low computational cost process at step 5. As a result, we can simulate a huge number of characters in real-time.

6.3 Interaction Patches

The Interaction Patch is composed of the initial condition of the two characters and the list actions performed by each of them. The initial condition includes the distance between the two characters (r), the relative orientation of each character with respect to the other (θ^1 and θ^2), and the delay in either of the characters to start the first action (t_{diff}).

In the rest of this section, we first explain how we preprocess the motion capture data, and then explain how the Interaction Patches are generated. Finally we explain how they are evaluated.

6.3.1 Preprocessing Motion Data

We assume the motion data is preprocessed and stored as an action level Motion Graph. The list of annotations used in this research is shown in Table 6.1.

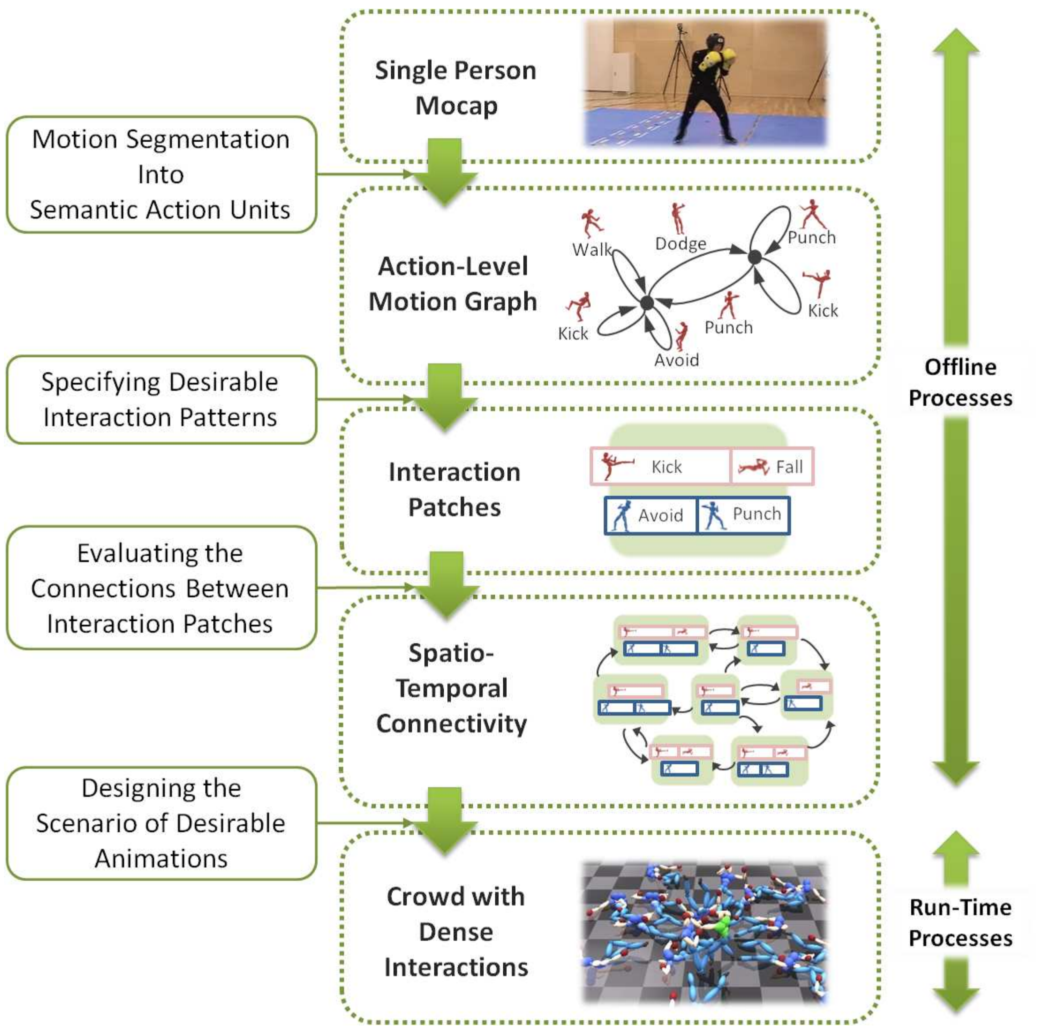
Motion Segmentation Into Semantic Action Units
Specifying Desirable Interaction Patterns
Evaluating the Connections Between Interaction Patches
Designing the Scenario of Desirable Animations
Single Person Mocap
Action-Level Motion Graph
Walk
Dodge
Punch
Kick
Punch
Kick
Avoid
Interaction Patches
Kick
Fall
Avoid
Punch
Spatio-Temporal Connectivity
Crowd with Dense Interactions
Offline Processes
Run-Time Processes

Figure 6.1: The outline of the Interaction Patches.

6.3.2 Composing Interaction Patches

The process of composing Interaction Patches is to let the user specify the pattern of actions, sample the initial condition of the two characters and simulate the interactions between them. An overview, showing the composition of an Interaction Patch is shown in Figure 6.2. Each process is explained in the following subsections.

6.3.2.1 Specifying Pattern of Interactions

A user first gives a list, defined here as a *PatternList*, that describes the pattern of the interaction between two characters: $PatternList = \{ (CharID_1, Annotation_1), \ldots, (CharID_n, Annotation_n) \}$, where $Annotation_i$ is the annotation embedded in the action level Motion Graph, $CharID_i$ is the identity of the character who performs this action, which is either 1 or 2, and n is the total number of actions in the pattern. In our system, multiple actions may share the same annotation. Therefore, an annotation represents a cluster of actions, rather than a specific action. Figure 6.2 (upper left) shows an example of *PatternList*. It should be noted that the list defines only the starting order of the actions, and does not mean each character has to wait for the other character to finish its action to start a new action.

6.3.2.2 Sampling Initial Conditions

Once the pattern of interaction is determined, the initial conditions of the characters are sampled based on the annotation of the first actions for each character (Figure 6.2, middle left). For most of the actions, there is a range in the initial condition parameters r, θ^1, θ^2, t_{diff} when the action becomes successful. For attacks or tackles, the other character must be in the front at some distance and the valid range is relatively narrow. On the other hand, avoiding actions are valid as far as the character can get away from the opponent, which means the range is larger. We predefine the valid range of each parameter for each annotation. The system computes the intersection of the valid range for the characters' first actions, and performs uniform sampling in the intersection. In our system, distance is sampled every $20cm$, angles are sampled every 20°, and time difference is sampled every $0.1s$.

6.3.2.3 Expanding Game Tree

When simulating the interactions between the two characters, each character is controlled by its own action level Motion Graph. Starting from the sampled initial

Scene	Annotations
Fighting	punch, kick, avoid, dodge, transition, falling
American Football	run, jump, avoid, tackle
Rat Avoiding	avoid, pushed
Crowd Falling	falling
Luggage Carrying	carry, walk, hand, receive, turn

Table 6.1: The table of annotations used to annotate captured motions

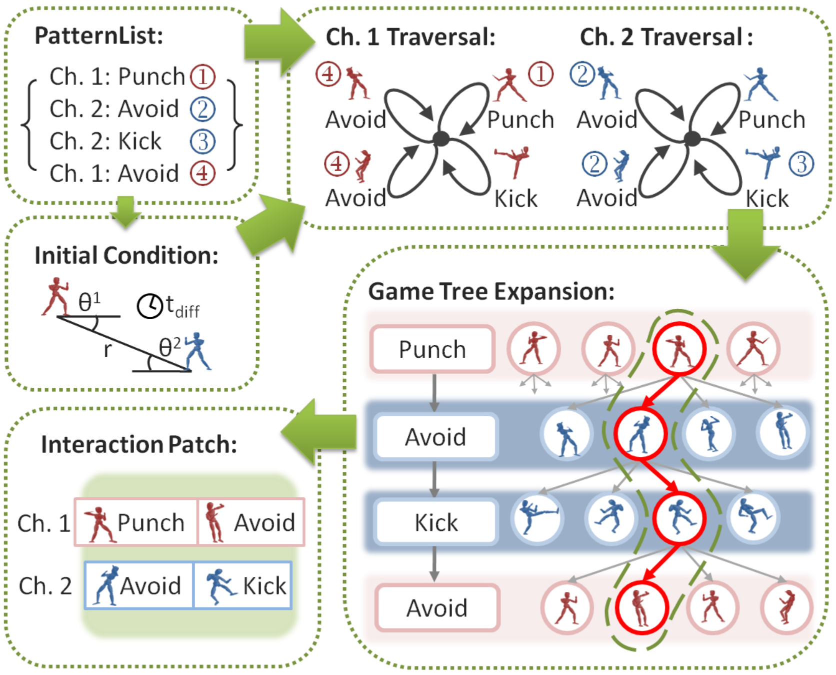

Figure 6.2: Given the *PatternList* (upper left), the system sets the initial condition (middle left). Using these data, the action level Motion Graphs are traversed by both characters (upper right). The traversal process is equivalent to expanding the game tree (lower right) as there are multiple choices for the same annotation. The good interactions are stored as Interaction Patches (lower left).

condition, each character traverses its own action level Motion Graph according to the pattern of annotations given by the *PatternList* (Figure 6.2, upper right). As the annotation represents a cluster of actions, we have multiple choices of actions for each annotation. Since *PatternList* contains a list of annotations, there are exponential combinations of instances per *PatternList*. The process to evaluate all possible combinations is equivalent to expanding a game tree (Figure Figure 6.2, lower right). In this game tree, each node represents an action to be launched by the corresponding character, and each edge directs the subsequent action by either character.

When expanding the game tree and evaluating the sequence of actions, some combinations are considered invalid for the following reasons:

- **Invalid distance:** We avoid interactions in which the characters stand too close, as they can cause serious penetrations.

- **Incorrect order of actions:** As the duration of each action is different,

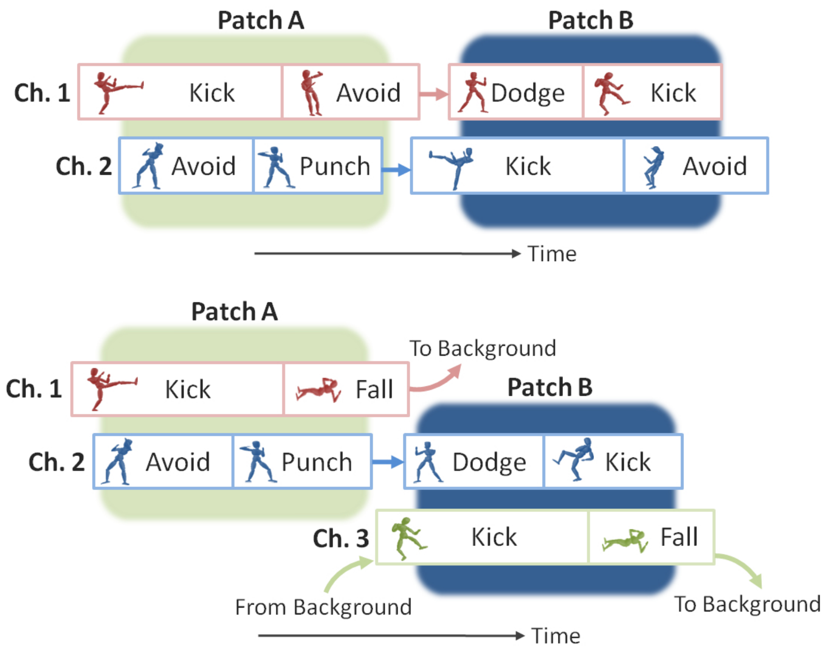

Figure 6.3: Two cases of temporal concatenation of Interaction Patches. Two characters finishing the previous Interaction Patch rejoin in the next patch (upper). One character starts to interact with a different character after finishing the previous patch (lower).

sometimes the overall order of the actions does not coincide with the pattern; such series of actions are discarded.

Close interactions involve a lot of close contacts of body segments. We need to evaluate whether the segments collide or not. We represent the body segments with rectangular bounding boxes and check if any segments are overlapping. If the colliding segment has large linear or angular momentum, response motion of being pushed or falling down is immediately launched. We compare every posture of the response motion with the posture at the moment when the impulse is added to the body. The best matching frame is used as the starting frame of the response motion [Zordan et al. (2005)]. If the segments unintentionally collide, such as when a character is supposed to successfully avoid the attack according to the given pattern but gets hit, this sequence of actions is discarded.

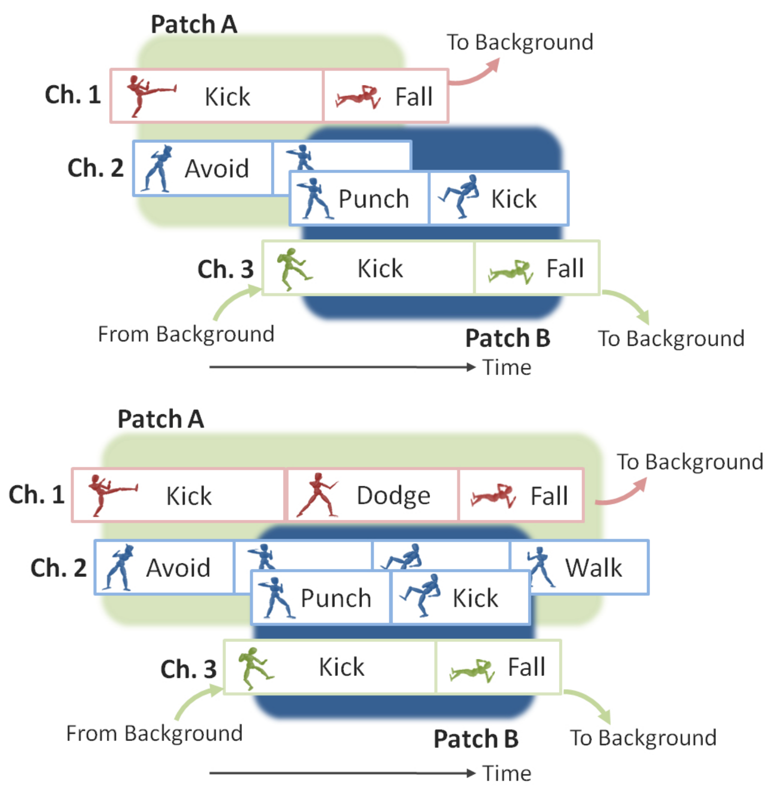

Figure 6.4: The condition for applying the spatial concatenation to the Interaction Patches: Either the series of actions in the initial and final part of the patches must overlap (upper) or the whole series of actions of one interaction patch overlaps with part of the other Interaction Patch (lower).

6.3.3 Evaluating the Interactions

After expanding the game tree, we evaluate the interactions using a cost function. Any paths connecting the root and leaf nodes of the game tree form a series of interactions between the two characters. The set of interactions with a score above a threshold are stored as Interaction Patches. The design of the evaluation scheme is specific to the type of interactions. We used the linear combination of the following objective functions in our experiments.

- **Contact criterion:** For some actions such as holding the hand, punching the face, and tackling the body of the other person, some parts of the bodies must contact either for a moment or throughout the timeline. Better scores are given to a series of actions that result in desired contacts.

- **Relative distance / orientation criterion:** For actions such as dancing, the characters need to stay close and face each other for some period. Similarly, for interactions such as one character punching and the other avoiding, the defender should get away from the punch, but needs to face the attacker while avoiding it. For these interactions, there are desired distances and relative orientations of the root of the body at some moment / throughout the motion. We can evaluate the interactions based on the difference of the resulting values and the desired values.

- **Timing criterion:** Some combinations of actions performed by both characters need to be well synchronized. We consider those interactions with small timing differences to be better.

All the interactions designed in our experiments are modeled by different combinations of the above functions. The blending ratios are manually tuned for each example.

6.3.4 Computational Efficiency

Since the process of constructing the Interaction Patches involves game tree expansion, the computational cost is of exponential order. In general, when fully expanding the game tree to evaluate the interactions of characters, the computational cost is A^D, where A is the average number of available actions, and D is the depth of the tree to be expanded. However, we can greatly reduce the cost by making use of the following features:

1. **As the patterns of actions are given**, the number of actions to be expanded at each level is much fewer than that of doing a full search. Assuming the actions are evenly divided into N types of annotation, the computational cost will be reduced to $(\frac{A}{N})^D$. At the same time we can get high quality samples, as the pattern of interaction is a very important factor to determine the realism of the interaction.

2. **As the *PatternList* is short**, the depth of the expanded tree, D, is limited. This is because only short Interaction Patches are required in our system. We can generate longer interactions, and those of more than two characters, by concatenating the Interaction Patches based on the method explained later in Section 6.4.

6.4 Connecting Interaction Patches

We compose large scale scenes by connecting the Interaction Patches. Long series of interactions can be created by temporally concatenating the Interaction Patches. Animations of more than two characters concurrently interacting can be composed by spatially concatenating the Interaction Patches. We check if such concatenations are possible for every pair of Interaction Patches, and save this information in a table. The details of checking the eligibility of temporal and spatial concatenations are explained in the following subsections.

6.4.1 Temporal Concatenation of Interaction Patches

Two Interaction Patches A and B can be temporally concatenated if (1) both of the characters finishing patch A start interacting again in patch B (Figure 6.3, upper), or (2) one of the characters finishing patch A joins patch B and starts to interact with a different character (Figure 6.3, lower).

The patches must satisfy two further conditions to be temporally concatenated: Firstly, the motions when switching from patches A to B must be continuous; this can be examined by checking the continuity of actions in the Motion Graph. Secondly, if the characters in the two patches are different, as in Figure 6.3 (lower), we must make sure the leaving character in patch A does not collide with the joining character in patch B. The leaving character either leaves the scene or joins another Interaction Patch with another character. For example, in Figure 6.3 (lower), after patch A, character 1 goes away and character 3 joins in patch B. Collision detection based on the two bounding boxes that surround character 1 and character 3 is carried out for all actions in the patch. Only if there is no collision can patch A and B be temporally concatenated.

6.4.2 Spatial Concatenation of Interaction Patches

The animator might need a scene where more than two characters concurrently interact; we can compose such a scene by spatially concatenating Interaction Patches of two characters. For example, the animator might need a scene in which a football player jumps up and avoids tackles from two opponents, one from the left and another from the right. This scene can be composed using two Interaction Patches, in which (1) a character jumps and avoids the tackle from the left, and (2) a character jumps and avoids the tackle from the right. There are two conditions for such a concatenation (Figure 6.4). First, the two uncommon characters in the two patches (character 1 and 3 in Figure 6.4) must not collide into each other. This condition is the same as the one in temporal concatenation. Second, the common character in the two patches (character 2 in Figure 6.4) must conduct the same series of actions for a continuous duration. The duration of overlap does not have to cover the whole Interaction Patch. If the ending part of one patch and the initial part of another patch overlap (Figure 6.4, upper) or if the whole series of actions in the shorter patch completely overlaps with a part of the longer patch (Figure 6.4, lower), this condition is satisfied.

6.5 Scene Composition

Once we know the set of Interaction Patches that can be concatenated, we can automatically compose large-scale scenes by spatio-temporally concatenating the patches. In this section, we explain the process of composing the scene: First, we explain the criteria for selecting the next Interaction Patch among all the available ones, and then explain how these criteria are applied to generate the scene. Finally, we explain how to reuse characters that exited Interaction Patches for other Interaction Patches later in the scene.

6.5.1 Selecting Patches

Among all the patches that can be connected to the currently played one, our system excludes those which result in collisions, and then selects the best one among the rest based on an objective function explained in this subsection.

First, we exclude the patches that result in collisions. If a patch requires the system to add a new character to the scene, we need to ensure that the newly added character does not collide with any other characters present in the scene. This is done by representing each character as a bounding box and checking if the new character overlaps with those in the scene. Then, we evaluate the Interaction Patches based on the following factors:

- **Density of characters:** Because there are going to be a large number of characters involved in the interactions, we favor patches that allocate characters in open space. This is evaluated as follows:

$$s_d(p) = \frac{1}{d_p + 1}$$

where d_p is the current density of characters at the region where the candidate Interaction Patch p will occupy.

- **Frequency of the usage:** As we prefer the characters not to keep repeating similar movements, lower scores are given to patches which have been recently used. We define a parameter f_p to represent the usage of the patch p; once a patch is used, its corresponding f_p value is increased by one. On the other hand, the value is decreased by 10% each time other patches are selected. The usage score of the patch is calculated as follows:

$$s_f(p) = (1 - min(f_p, 1))$$

- **User preference:** We provide a simple interface for the user to select the preferred type of actions represented by action annotations. The patches that include such types of action are given better scores: $s_u(p) = 1$ if the action satisfies the user's preference and $s_u(p) = 0$ if it does not.

The final score of a patch is defined as the weighted sum of the above factors:

$$S(p) = w_d s_d(p) + w_f s_f(p) + w_u s_u(p) \tag{6.1}$$

where p is the patch to be evaluated, w_d, w_f, w_u are the weights for each factor, which we set as $w_d = 10, w_f = 1000$ and $w_u = 10000$. The patch that returns the highest score is selected.

6.5.2 Concatenating Interactions

Here we explain how to generate scenes of continuous interactions involving many characters by concatenating the Interaction Patches.

When an Interaction Patch is about to end, we automatically select the patch that can be temporally concatenated by evaluating all the connectable patches using Equation 6.1. If there are any patches which are spatially connectable, such patches are also evaluated by Equation 6.1 and the one with the best score is concatenated.

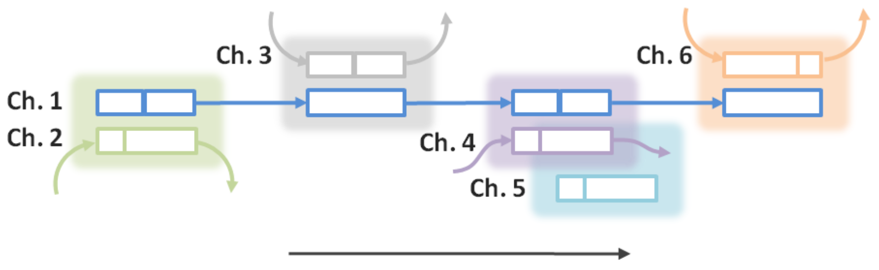

Figure 6.5: Creating scene by applying spatio and temporal concatenations on Interaction Patches. We can generate scenes in which a main character interacts with many background characters. We assume the background characters comes from background before the interactions, and return to background after the interactions.

Then, the movements before and after the interactions for the characters are generated by a locomotion engine that controls the character in a greedy manner. The locomotion engine selects a movement which is collision free and transfers the character as close as possible to the target position. The movements of the characters are determined backward and forward in time starting from the moment of the interaction. For those characters that appear from the background, the starting point is set at a location outside the scene in the radial direction. The motions of the characters whose interactions happen first are decided first. Therefore, when deciding the locomotion of each character, we only need to avoid the characters that are already in the scene. Although more elaborate locomotion engines based on model predictive control [Lau & Kuffner (2005)] or reinforcement learning [Lo & Zwicker (2008)] might perform better, our controller works well for the scenes we simulated.

An example of an overall time line is shown in Figure 6.5, in which character 1 (Ch.1) interacts with character 2, 3, 4 and 6 (Ch.2, Ch.3, Ch.4 and Ch.6) with temporal concatenation. The Interaction Patch shared by Ch.1 and Ch.4 is spatially concatenated with another patch shared by Ch.4 and Ch.5. A corresponding fighting scene is shown in Figure 6.6. Ch.1 (blue) first attacks Ch.2 (green) at the right side of the image, and next Ch.3 (grey) at the top, then Ch.4 (violet) at the left, and finally Ch.6 (orange) at the bottom. When Ch.4 falls down, this motion is spatially concatenated with another Interaction Patch, in which it falls over character Ch.5 (cyan). Once the Interaction Patches are fixed, the motions of the characters entering the scene are decided.

6.5.3 Recycling Characters

When multiple characters continuously interact, they need to repeatedly enter and exit Interaction Patches (character 1 to 3 in Figure 6.7). For instance, if we want to design such a scene for two characters, both characters going out from a patch need to rejoin in the next patch. However, sometimes these kinds of patches cannot be found due to the distinct initial condition to start an Interaction Patch. We solve this by giving the characters the degrees of freedom to adjust their locations, orientations and postures.

First, we introduce the concept of standard pose, which is a pair of postures for two characters, from where the two characters can easily find ways to enter various Interaction Patches (Figure 6.8). This corresponds to the hub nodes [Gleicher et al. (2003)] in the Motion Graph. We first categorize the initial and final postures of the Interaction Patches according to their relative distance, orientation and postures. The average poses of all the categories are computed and they become the standard poses. Then, we can concentrate on planning how to reach the standard poses. We use the locomotion engine for moving the characters to the desired locations when it is far away from the standard pose. The characters move towards the nearest standard pose to start another Interaction Patch.

We define a distance function that evaluates the difference between the current pose (P_c) and each standard pose (P_s) as follows:

$$F(P_c, P_s) = \left(\frac{r_c - r_s}{r'}\right)^2 + \left(\frac{\theta_c^1 - \theta_s^1}{\theta'}\right)^2 + \left(\frac{\theta_c^2 - \theta_s^2}{\theta'}\right)^2 \tag{6.2}$$

where r_c is the distance between the characters, θ_c^1 and θ_c^2 are the angles between the line connecting the two characters and the direction each character is facing, r_s, θ_s^1, θ_s^2 are the corresponding values in the standard pose. The constants r' and θ' are used to normalize the effects of distance and angle, and are set to $300cm$ and 180° respectively. The distance between the current status of the characters and each standard pose is calculated and the one with the smallest distance is selected:

$$argmin_{P_j} F(P_c, P_j) \tag{6.3}$$

where P_j is the j-th standard pose, and P_c is the current status of the two characters.

Once the target standard pose is selected, each character approaches the character it is to interact with by using the locomotion engine. When the characters are at the required relative distance and orientation, each character expands the game tree to find the action that brings its posture to that in the standard pose. Since (1) the connectivity of the action level Motion Graph is high, and (2) the posture of each character in the standard pose is a commonly used posture, we can usually arrive at the target pose in one step. If the graph connectivity is low, and complex path planning is required for arriving at the standard pose, it is possible to apply dynamic programming to find the path in real-time.

As a result, even if there is no available Interaction Patch that can be immediately launched, the characters can move around and adjust their poses to start the next desirable Interaction Patch. As for timing, if one character arrives at the corresponding posture in the standard pose slightly earlier than the other character, we let the character wait there so that it is synchronized with its opponent before launching the next Interaction Patch.

6.6 Experimental Results

Using our method, we have simulated two types of scenes, which are generated by (1) only concatenating Interaction Patches, and (2) using the standard poses to let the characters continuously interact. The first group of scenes can be generated by the method explained in Section 6.5.2, and the second group of scenes further requires the techniques explained in Section 6.5.3. The set of *PatternList* used to generate the Interaction Patches are shown in Table 6.2.

A particle system was designed to handle unexpected collisions among the characters and adjust the actions performed. Further information can be found in Appendix A.

6.6.1 Scenes Generated By Concatenating Interaction Patches

We simulated scenes where (1) a main character fights with many background characters (Figure 6.9, Figure 6.10), (2) a group of people fall down over each other like dominos (Figure 6.11, Figure 6.12), (3) an American football player holding the ball avoids the defenders and runs towards the goal (Figure 6.13), and (4) a mouse runs into a crowd and the frightened people avoid it and bump onto each other (Figure 6.14). Although our system can automatically select all the actions for all the characters, usually the user prefers to give high level commands. Therefore, for scenes (1), (3) and (4), we have prepared an interface for the user to provide basic commands such as transition and rotation of the character, as well as field-specific commands such as punch, kick, and avoids. The commands correspond to $s_u(p)$ in Equation 6.1.

6.6.1.1 A Character Fighting With A Lot Of Enemies

We created a scene in which a stronger character fighting with a lot of weaker enemies (Figure 6.9). During the animation, the stronger character avoids or dodges the attacks of the weaker enemies, and counter attacks the enemies. One features of this animation is that when designing the Interaction Patches, we require the enemies die after the interaction. By this way, we do not need to handle the motions of the enemies after the interactions.

Apart from the interactions between the stronger character and its enemies, we also apply Interaction Patches to handle the motions while weaker enemies bumping onto other weaker enemies. These Interaction Patches are designed by combining PD control and motion of being pushed away or falling down [Arikan et al. (2005), Zordan et al. (2005)]. More specifically, when expanding the game tree to generate such Interaction Patches, the appropriate reactive motions are selected and blended to the motions of the characters whenever the two of them collide.

Although our system can generate the fighting scene automatically, some users may have specific preference on how the scene looked like. Therefore, we design a user interface for the user to control the actions of the stronger character. Such user defined actions will then act as constraints when selecting the Interaction Patches, which correspond to $s_u(p)$ in 6.1. Then, our system automatically searches for the appropriate Interaction Patches to plan the movements of the weaker characters (Figure 6.10).

6.6.1.2 Characters Falling Onto One Another

A scene in which a large number of characters falling onto one another like dominos was created (Figure 6.12). In this experiment we only use the Interaction Patches in which one character falling onto another character. During run-time, spatio concatenation is used in two ways. First, we concatenate patches such that one character falls onto two or more characters. Second, Interaction Patches are also concatenated such that the newly falling characters continue to fall onto other characters.

The Interaction Patches are automatically concatenated so that the area specified by a given bitmap on the floor is filled with characters falling to the ground. As the interactions between the characters are precomputed, even for large numbers of characters, we can obtain the results in real-time. After planning the motions for the falling down characters, the standing characters are added to fill up unused areas in the scene.

We also generated a scene in which hundreds of characters falling onto each other (Figure 6.12). In this case, we create one starting point of falling for each letter on the floor, and define the time for which to being.

6.6.1.3 American Football

We created a scene in which an American football player runs to the goal line while avoiding tackling defenders (Figure 6.13). Each Interaction Patch contains a short clip of running motion for the offensive player, the tackling motion of the defensive player, and the avoid motion of the offensive player. By repeatedly temporal concatenating the Interaction Patches based on the offensive player, we can simulate a continuous running and avoiding motion.

We allow the user to indicate the running directions of the offensive player, as shown in the arrow on the floor. The system then chooses the appropriate patch with the most similar running direction.

6.6.1.4 Characters Avoiding A Rat

We synthesized a scene in which a group of characters avoid a rat and bump into each other. We defined two *PatternList*. The first one defines the interactions between the rat and the avoiding character, while the second one defines the bumping interactions between two characters.

In our system, the rat is a simplified character with only one joint for the body. We let the user define the movement trajectories of the mouse, which consists of a lot of shorter movement steps. Then, the scene is generated automatically by applying Interaction Patches that contain the movement steps of the rat.

6.6.2 Scenes Where Characters Are Recycled

We simulated scenes where (1) two characters are continuously fighting (Figure 6.15) and (2) a group of characters are passing luggage one after another to the characters next to them (Figure 6.16). Since for both of the experiments, we require the characters to interact more than once, we need to reuse the character in the scenes.

6.6.2.1 Two Characters Continuously Fighting

When generating a scene in which two characters continuously fighting (Figure 6.15), after finishing an Interaction Patch, the characters either immediately enter another patch, or search for a standard pose which leads them to a set of other patches. A fighting scene where the characters keep on attacking and defending can be generated. Although the temporal tree expansion and Interaction Graph approaches can generate scenes of continuously fighting as well, those generated by Interaction Patches are more stylized due to the uses of *PatternList*. For example, due to the use of Interaction Patches, we can generate a scene with intensive and well synchronized interactions.

6.6.2.2 Many Characters Carrying Luggage Cooperatively

We created a scene in which many characters carrying luggage cooperatively (Figure 6.16). Each character continuously interacts with one of its neighbors when the luggage arrives. Each Interaction Patch includes the motion of the first character standing, walking to receive the luggage, carrying and handing it to the second character, and going back to the original location. Spatio concatenation is used such that when one character receives the luggage with the first patch, it passes the luggage to another character with the second patch. Different Interaction Patches are selected according to the size and the weight of the luggage. We define a set of standard poses which are suitable for passing and receiving luggage. Using these Interaction Patches and standard poses, we have generated a scene where a large number of characters pass luggage one after another to the next person.

6.6.3 Computational Costs

The computational speed and the number of actions and patches of each experiment are shown in Table 6.3. The computer used comes with a Pentium 4 Dual Core 3.0 GHz CPU and 2 GB of RAM. The reason for large numbers of Interaction Patches in the “Mouse” and “Crowd falling” demo is that we need to generate characters colliding from all directions for different orientations of the characters.

Excluding the rendering, all the animation can be generated in real-time, once the instructions from the user are given.

6.7 Discussions

6.7.1 Patches Creation

Although we use the *PatternList* to generate the Interaction Patch, animators can manually design the Interaction Patches action by action. In fact, many animators prefer to design complex elaborate interactions between the characters which are difficult to be generated automatically. The design of the Interaction Patches is simple, because they are short and only involves two characters. The designed Interaction Patches can be added into the database together with the automatically generated ones.

Interaction Patches can also be created by capturing the motion of two persons. This method works for sparse interactions such as two persons avoiding each other while walking, and two persons shaking hands. However, it is not recommended for dense interactions such as fighting due to the technical difficulties of motion capturing devices. Another disadvantage is related to spatio concatenation, in which two patches can only be concatenated when part of the first patch is effectively the same as part of the second one. While it is difficult to perform an action exactly the same in two trials, captured patches suffer from the low rate of successful spatio concatenation.

When generating the Interaction Patches, although we limited the number of characters in each Interaction Patch to two, those of three or more characters can also be generated. In that case, we can generate the Interaction Patch of multiple characters by expanding the game tree for all of them, and then concatenating them as done in this research. However, more computational time will be needed to expand the game tree. We believe spatio concatenation is a more efficient method to generate multi-character interactions.

We assume the interactions between the characters are short. In our experiments, we limit the total number of actions inside the Interaction Patches to be four. This condition helps the system from two aspects: Firstly, the computational cost to generate the Interaction Patches can be minimized. We will be able to scan a huge number of combinations which increases the chance of finding plausible interactions. Secondly, shorter interactions are better to be fit into the scene, as they are more compact and less influential to the other extra characters in the scene. This helps the final process to compose the scene, as long interactions become a constraint for the other characters. To generate longer sequences of interactions, we suggest the use of temporal concatenation.

6.7.2 Scene Generation and Controllability

For scenes where the main character interacts with many the background characters, we assume the interaction between two different characters occur only once. Therefore, the number of characters appearing in the scene is proportional to the number of interactions, which may cause the scene to be filled up by characters. Although the collisions of characters will not occur as the motions of the characters are subsequently determined, there is a potential risk that they behave in an unnatural way due to the lack of space. This problem is avoided by letting the background characters fall down onto the ground or disappear from the scene after the interaction. The fact that the interactions between two characters continues at least for a few seconds also helps to keep the number of characters appearing in the scene to be limited. We can also reuse some characters if the scene is too crowded. As a result, we do not face problems of unusual movements even though we do not explicitly implement any function to control the number of characters in the scene.

There are two possible extensions to enhance the controllability of the characters. The first method is to greatly increase the number of Interaction Patches and introduce a hierarchical structure to store the patches. In that case, according to the input by the animator, the corresponding cluster will be selected first, and then the best patch in the cluster will be selected subsequently. The second method is to introduce parametric techniques to deform and interpolate the existing patches. Using such a method, we will be able to produce a large number of variations from a small number of patches.

6.7.3 Usage Complexity

The patch generation process is fully automatic except the need of a pattern list. It is easy to define a valid pattern list during the patch generation process. However, to generate artistic interactions, it is more about arts than technology. The animators must be familiar with the movie industry and have good creativity.

Similar to the tree expansion method in Chapter 4, the objective functions to evaluate the quality of a patch are problem specific. Implementing extra objective functions will require in-depth understanding on the system. One possible solution is to manually select good-looking patches. If only a few tens of patches are required, the selection process is in fact not time consuming.

We created different interfaces for the animators to control the patch concatenation process. These interfaces are very easy to be used. However, similar to the experiments of the Interaction Graph in Chapter 5, since there may be a lot of characters in the scene, it is difficult to control every single detail. For example, when making the one-to-many fighting scene, our system automatically concatenate patches in which one character falling onto another. This is because we wish the animator focus on the main characters.

Generating the scene can be a trial and error process, mainly because our algorithm runs in real-time. For inexperienced animators, they can simply randomly press some buttons in our control interface and see the resultant scene. Usually, after a few trials, the animators would understand how the scene is created, and could generate the scene they want.

6.7.4 Comparisons On Different Control Systems

The Interaction Patches system requires far fewer samples than other optimization-based systems. For example, in Interaction Graph explained in Chapter 5, the number of samples produced is over 50,000. With this large number of samples, it is difficult to monitor the quality of the interactions. For the experiments in this research, fewer than 300 Interaction Patches are needed to create a stylized fighting scene. Methods such as the Interaction Graph are targeted for real-time applications such as computer games. In order to make the computer-controlled character strong, the controllability of the character must be high, which means the character needs to be able to launch various kinds of movements, including subtly different stepping and attacks. This results in dense sampling of the state space. On the other hand, the objective of this research is to create a stylized animation of characters interacting. The system does not need high controllability of the characters, but only needs to be able to satisfy the high level commands given by the animator. In addition to that, as our system first determines how the characters are going to interact, the characters have a lot of degrees of freedom to adjust their movements before and after the interactions. As a result, we can greatly reduce the number of interaction samples.

We can evaluate the systems in terms of accuracy on control. That is, given the control signals by the animators, how accurate can the characters follow the signals. In this case, the Interaction Patches system performs poorly. Because of the small number of available patches, it is difficult to tell the characters perform exactly what the animators want, such as moving to a predefined position. On the other hand, with the Interaction Graph explained in Chapter 5, the animator can control the characters much more accurately. This is because we can quantize the possible control signals can compute the optimal policy for every quantized signal. Still, the accuracy depends on the level of quantization. If we quantize the control signals heavily, the accuracy will be low. Otherwise, with a set of finely quantized control signal, the accuracy will be high, but the system will require a huge amount of memory. The temporal tree expansion method explained in Chapter 4 performs the best in terms of accuracy on control. This is because the method is essentially a short horizon optimization system. That is, we always plan for a short duration based on the exact current situation. Hence, we do not need to quantize the control signals at all. An example is shown in Figure 6.17. Suppose the animators require a pair of fighting characters to follow a predefined path, the characters controlled by temporal tree expansion will follow the most accurately. Those controlled by the Interaction Graph can follow the path, but the accuracy depends on the level of quantization. Those controlled by the Interaction Patches cannot follow the path well since there are only a small number of patches.

When comparing the computation cost of different systems we proposed, the temporal tree expansion method performs the worst. This is because the cost of every action selection increase exponentially with respect to the depth of the game tree, and usually we require three levels of tree expansion to create a smart character. Both Interaction Graph and Interaction Patches use precomputed tables for action selection, and thus the computation cost increase linearly with respect to the size of the tables. Since the Interaction Patches method use a far smaller table, it is even more computational efficient than the Interaction Graph. Another advantage of the Interaction Patches method is that each patch defines multiple actions for two characters. Therefore, when planning a scene with a predefined number of characters and duration, the Interaction Patches requires much fewer table lookups when comparing to the Interaction Graph.

If we wish to implement or extend the character control systems in this thesis, the temporal tree expansion method requires the smallest effort. We simply need to implement the objective functions and the min-max framework. Tuning the objective functions can be a trial and error process, as the effects are shown immediately. Both Interaction Graph and Interaction Patches are built on top of the temporal tree expansion method. The Interaction Graph requires much more effort to be implemented. This is because even if the framework is available, training the graph takes a lot of time, and in case things goes wrong, we will need to retrain the graph. This is a general problem of learning based methods. However, once the graph is successfully trained, we can tune the way we evaluate the actions and see the effect almost immediately. For example, when creating different style of fighting, we simple need to tune the parameter. The whole graph can be updated in seconds, and the resultant animation can be generated in real-time. Finally, implementing the Interaction Patches method requires building the patch concatenation framework. Once the framework is completed, we can easily create Interaction Patches by tree expansion. The framework works well with only a few patches, and hence building the patches requires only a short time.

The Interaction Patches system can become an alternative to creating realistic interactions by using infinite horizon optimization methods such as reinforcement learning. In theory, it is possible to produce realistic interactions between characters if each of them select motions based on what benefits them the most. However, in practice, such smartness can make the scene less stylized as the characters will never conduct actions that do not benefit them. The characters become too careful and as a result, they will never launch risky movements that can make the interactions more attractive. On the other hand, the animators or the audience want to see energetic movements. It is much easier to produce such interactions by using our short-horizon method as the users can explicitly specify the pattern of interaction they want to see. Another advantage is that the computational cost is limited by the short depth of the game tree.

6.7.5 Limitations

There are some limitations with our method. First of all, the process of specifying the pattern can cause problems if the actions by the characters are abstract and aimless as they are difficult to annotate. Our method is more suitable for actions which are easy to annotate. Secondly, we have limitations in generating scenes where multiple characters continuously interact. In the examples shown, the characters were allowed to adjust their movements without a time limit. If the time and locations of the interactions are strictly constrained, a global planner that can plan the sequence of all the characters at once will be required. Solving such a problem using discrete optimization is one of the possible solutions.

6.8 Summary

We proposed a method to develop large-scale animations where characters have close interactions. The user can obtain stylized interactions between the characters by simply specifying the pattern of interactions. The interactions between the characters are saved by data structures called Interaction Patches. The Interaction Patches are spatio-temporally concatenated to compose large-scale scenes. Once the Interaction Patches are prepared, the process of composing the scene is fully automatic. At the same time, the users can control the scene using our control interface.

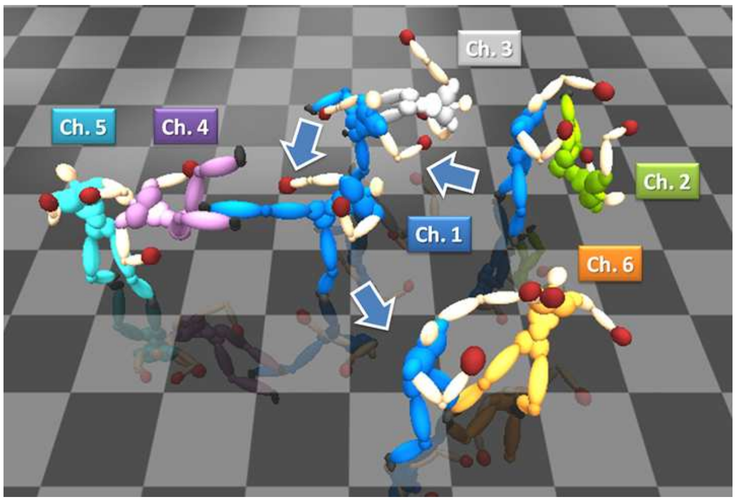

Figure 6.6: The scene that corresponds to the data flow shown in Figure 6.5 upper. The blue character (Ch.1) sequentially interacts with Ch.2, Ch.3, Ch.4 and Ch.6. This sequence of interactions is composed by temporal concatenation. Ch.4 falls over Ch.5. This interaction is produced by spatial concatenation.

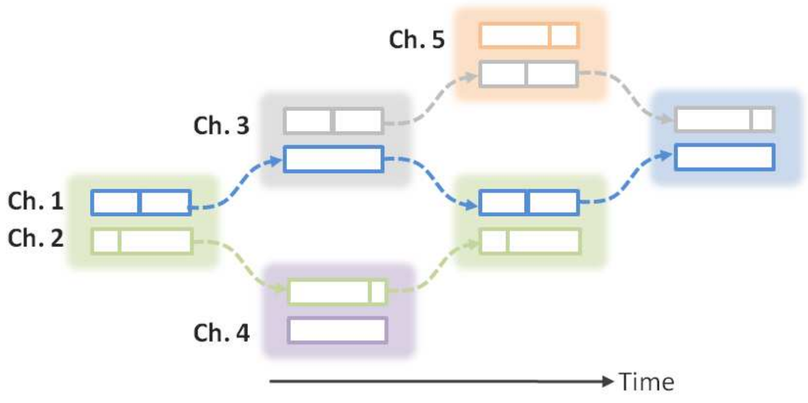

Figure 6.7: With characters recycled, we can create scenes in which characters continuously interact with other characters. The dotted lines indicate that adjustment motions may be required to connect two patches.

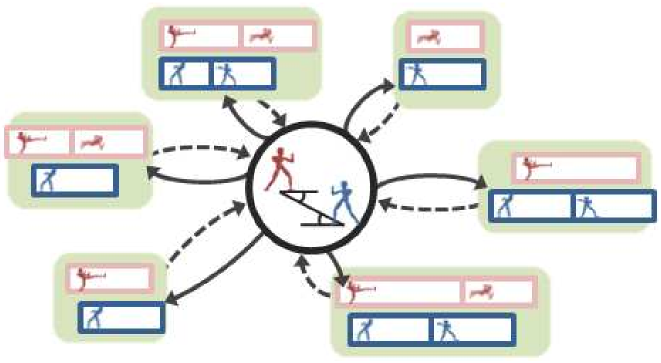

Figure 6.8: The standard pose (the circle at the center) acts as a hub to connect different Interaction Patches. The dotted lines indicate that the characters in the patches may need to adjust their locations and orientations for getting back to the standard pose.

Scene	***PatternsList***
Fighting (one-to-many)	{attack, **defence**, **attack**, fall},
	{**attack**, fall},
	{attack, **attack**, fall},
	{arbitrary motion, **fall**, fall}
Fighting (one-to-one)	{attack, **defence**},
	{**attack**, fall}
American Football	{run, **tackle**, avoid}
Rat Avoiding	{arbitrary motion, **avoid**, pushed away},
	{arbitrary motion, **pushed away**, pushed away},
	{run, **avoid**},
	{arbitrary motion, **avoid**, fall}
Crowd Falling	{arbitrary motion, **fall**, fall}
Luggage Carrying	{carry, **walk**, hand, **receive**, turn, **carry**}

Table 6.2: The *PatternList* used to compose the Interaction Patches (The actions of the second character are shown in bold font). Attack includes punch and kick, and defence includes dodge and avoid.

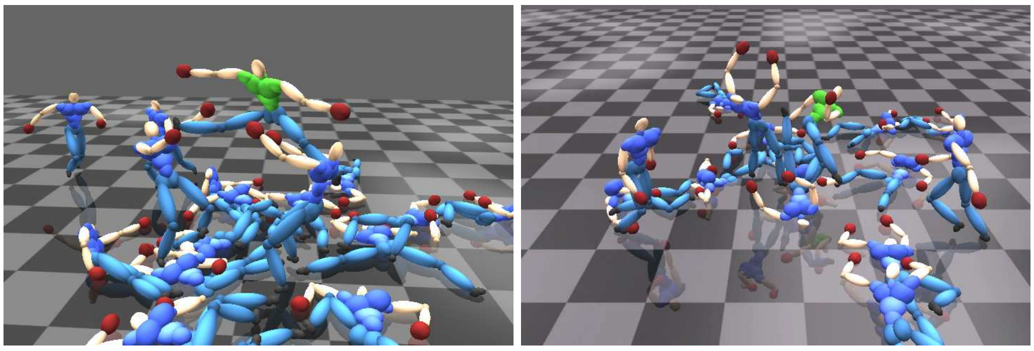

Figure 6.9: Animation of one person fighting with many enemies generated by our system. Interaction Patches are used to produce the fighting interactions between the main character and the enemies, as well as the bumping interactions among the enemies.

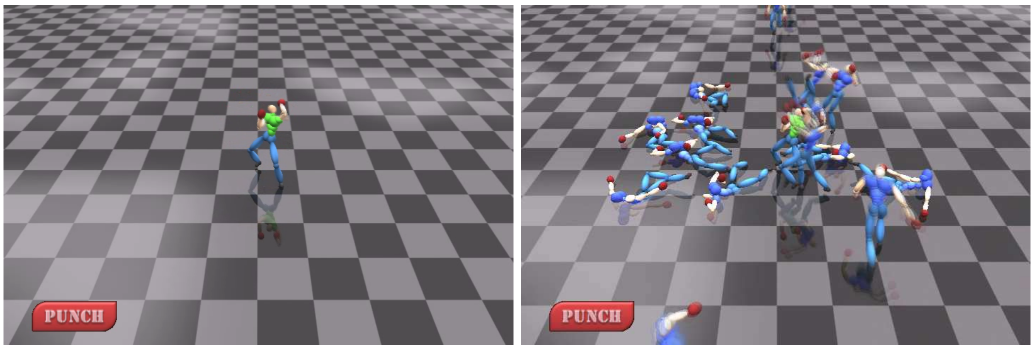

Figure 6.10: Apart from automatic simulation, we also designed a user interface let the user synthesize one-to-many fighting semi-automatically. The user first controls the movement of the main character. Then, the system will plan the motion of the enemies with Interaction Patches.

Scene	Speed (fps)	Actions	Patches
Fighting (One-to-Many)	87	162	157
Fighting (One-to-One)	104	162	279
American Football	194	217	21
Rat Avoiding	78	65	3716
Crowd Falling	72	39	4091
Luggage Carrying	162	108	72

Table 6.3: The computational speed, number of actions and number of Interaction Patches of each experiment (Computational speed above 60 frame per second (fps) is real-time)

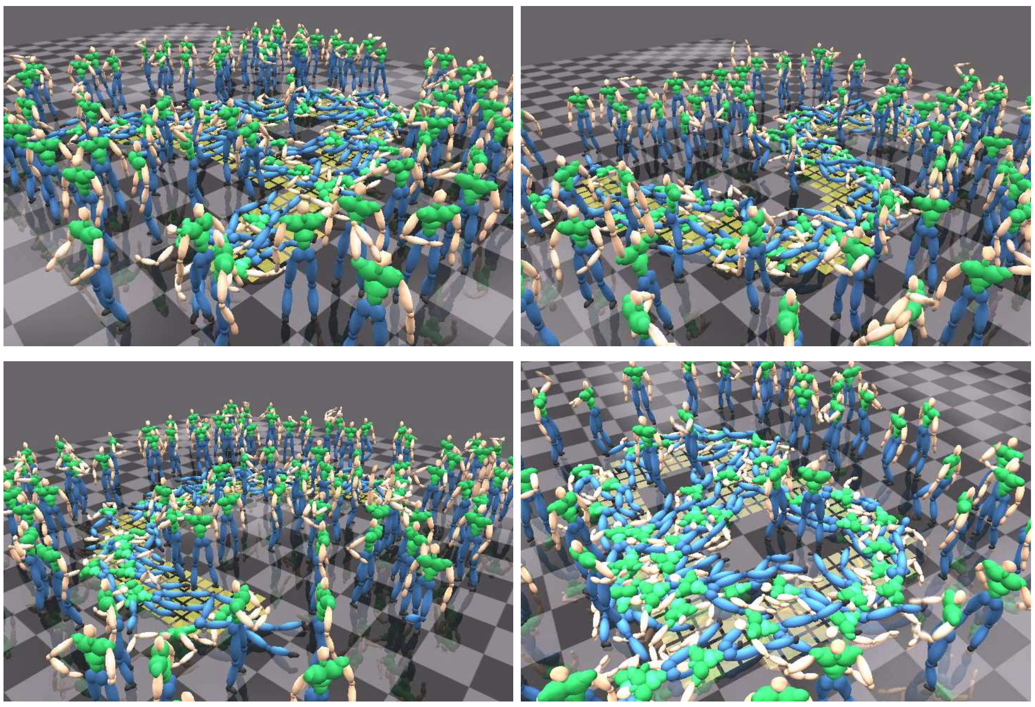

Figure 6.11: Animation of character falling onto one another generated by our system. The user can control the overall pattern of falling by designing a bitmap as shown on the floor. Characters standing on the pattern will fall while the others remain standing.

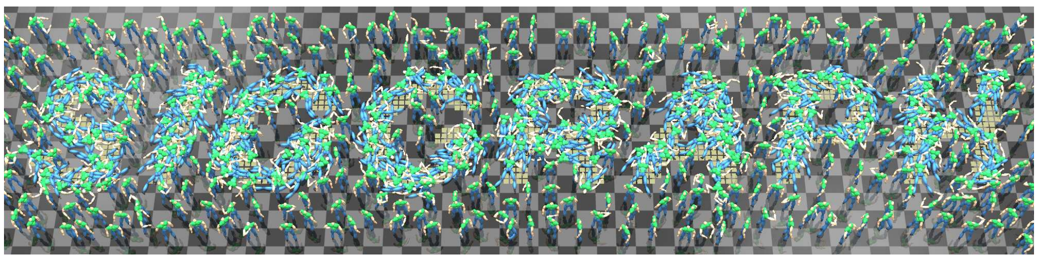

Figure 6.12: Animation of hundreds of character falling onto one another generated by our system. Despite of the large number of characters, the motions are planned in real-time.

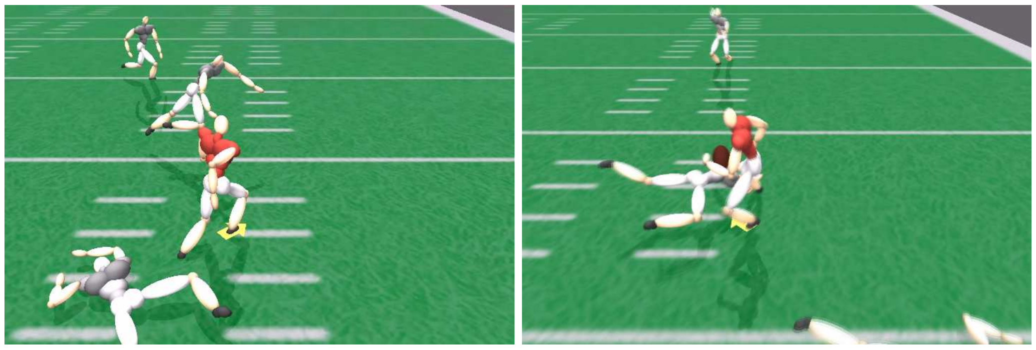

Figure 6.13: Animation of American Football generated by our system. Interaction Patches are used to produce the interactions when one character running while avoiding another tackling character. The user can control the direction of running for the main character as shown in the arrow on the floor.

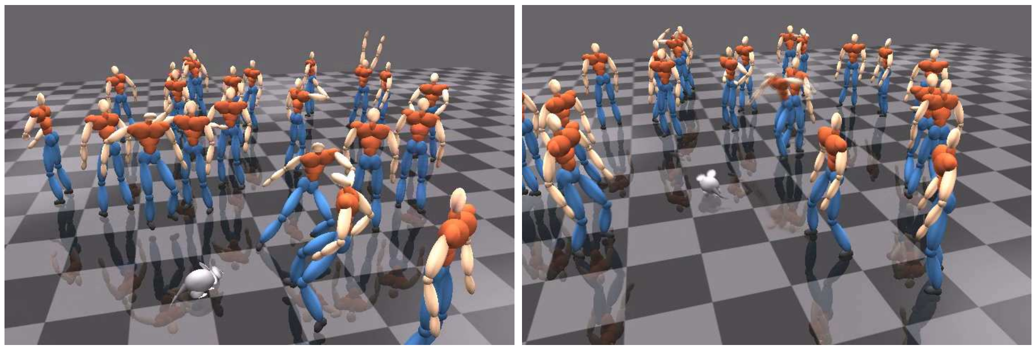

Figure 6.14: Animation of people avoiding a rat and bumping onto each other generated by our system. Interaction Patches are used to produce the avoiding interactions between the rat and the character, as well as the bumping motions between the characters. The trajectory of the rat is controlled by the user.

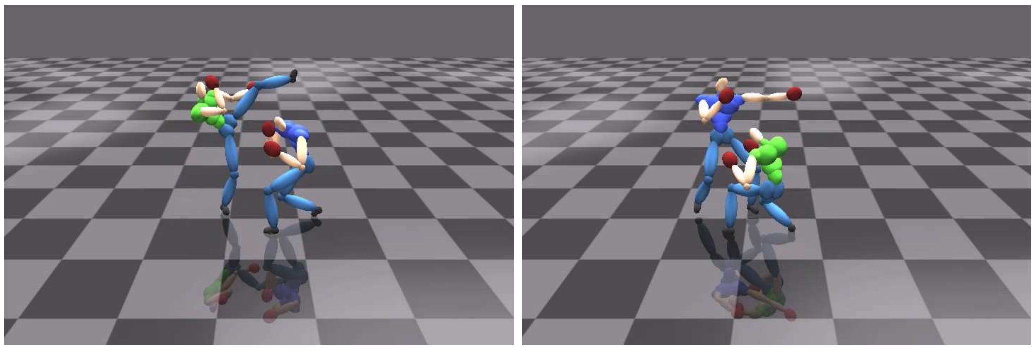

Figure 6.15: Animation of two characters continuous fighting with each other generated by our system. We continuously apply temporal concatenation and reuse the two characters such that they continue to interact.

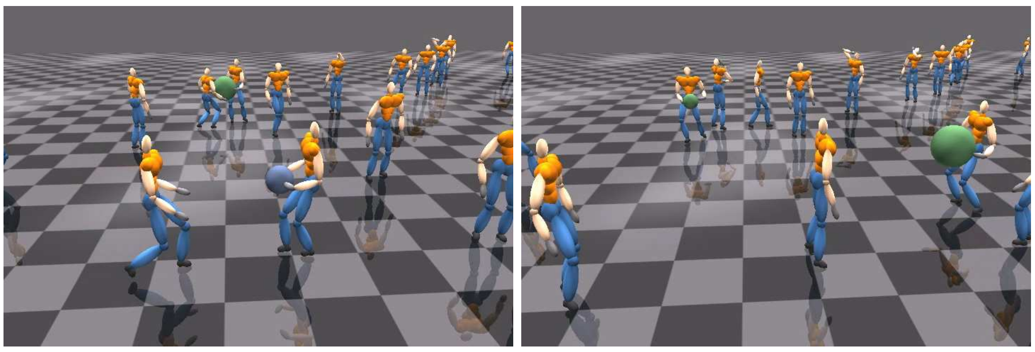

Figure 6.16: Animation of characters moving luggage cooperatively generated by our system. The characters are reused when planning the scene such that they interact multiple times when different luggage arrives.

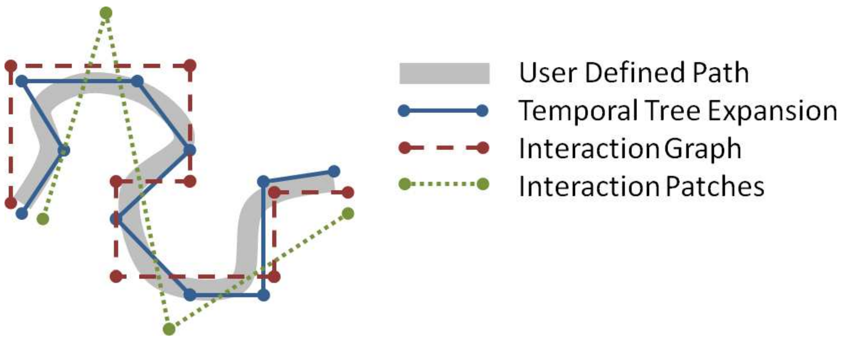

Figure 6.17: With a predefined path to follow (grey thick line), the characters controlled by the temporal tree expansion method can follow the most accurately (blue solid line), those controlled by the Interaction Graph suffer from quantization error (red dashed line), and those controlled by the Interaction Patches cannot follow the path well (green dotted line).

Chapter 7

Conclusions

In this thesis, we presented our researches to synthesize the interactions among multiple characters. Character interaction is an essential field in the gaming and movie industries, but is still unsolved due to the difficulties to simulate the complex behaviors during interactions. We successfully modeled the interactions among characters, and designed artificial intelligence algorithms to synthesize cooperative and competitive interactions among multiple characters. We demonstrated our system with high quality scenes involving characters interacting with each other like real humans.

Our system first segments the raw captured motions into semantic actions (Section 3.1), and creates the action level Motion Graph (Section 3.3), which indicates possible transitions between actions. Based on the graph, we designed the artificial intelligence algorithms to control virtual characters interacting with each other.

We modeled the interactions between two characters as Markov decision processes. Inspired by game theory, we applied game tree expansion to predict the future states of interaction, in order to select the optimal actions for a character to interact with its opponent. Pre-defined objective functions were used to evaluate the reward of launching an action at a given state. We observed that most interactions involve both competitive and cooperative natures. This leaded us to design a multi-modal character controller by embedding the cooperative features into the min-max search framework. For example, in our experiments, the characters can fight with each other competitively while following the high-level instructions from the user cooperatively. We showed that the objective functions can be updated during run-time to simulate different styles of interactions. Furthermore, by expanding game trees of different levels, we can simulate characters with various intelligence levels.

Since the major computational overhead of the temporal tree expansion method is performed during run-time, the system is too slow to be used in real-time applications like games. In theory, we can precompute the optimal actions for all possible situations, which are known as states, for a character when it is facing an opponent. However, due to the complexity of interactions, the state space is too large to be exhaustively precomputed. We observed that the active areas of the state space are small compared to the whole space, and designed an off-policy approach to sample the states that involve high quality interactions (Section 5.3). Based on the sampled states, we created a structure called Interaction Graph, which is a finite state machine with the nodes representing states and edges representing actions (Section 5.4). We precomputed the immediate rewards for all state-action pairs in the Interaction Graph, and applied dynamic programming or min-max search to evaluate the optimal actions that benefit the character the most in the future. As a result, the computational cost during run-time is minimal, and we can create controllable characters for cooperative and competitive interactions in applications such as 3D computer games (Section 5.5).

Although the precomputation algorithm in Interaction Graph can simulate character interactions in real-time, it only generates one-to-one interactions due to the limitation in the state space. We explored the possibility of combining one-to-one interactions to form many-to-many interactions such as those appear in a war scene, and introduced Interaction Patches (Section 6.3), which are precomputed short clips of interactions between two characters. The major advantage of Interaction Patches is that they can be concatenated temporally to form longer interactions, and concatenated spatially to form interactions involving more characters (Section 6.4), with minimal computational overhead. We designed a system to synthesize the Interaction Patches off-line, and concatenate them with optional user instructions during run-time. We demonstrated our system by generating scenes that involve tens to hundreds of characters, including those of fighting, sports, and crowd simulation where characters fall onto each other (Section 6.6).

7.1 Summary of Contributions

In this section, we summarize the contributions in this thesis.

- We propose an algorithm to simulate dense interactions of two characters by applying game theory. We model interactions as a Markov decision process, apply temporal tree expansion to predict future states of interaction, and use min-max search to select the optimal actions. (Chapter 4)

- We propose a multi-modal approach to create competitive characters with cooperative features. We embed both competitive and cooperative objective functions into the min-max frameworks such that the characters can compete with each other while achieving common goals. (Chapter 4)

- We propose an off-policy approach to sample the huge state space of interactions between two characters. This is achieved by sampling the space with criteria that favor states with good connectivity and more interactions. The samples are general enough to be used in different control policies. (Chapter 5)

- We propose a finite state machine called Interaction Graph to precompute the optimal action for a character to collaborate or compete with one another, such that the character interactions can be simulated in real-time applications such as 3D computer games. (Chapter 5)

- We propose a method to precompute realistic interactions between two characters for a short duration and save in a data structure called Interaction Patches. The interactions are simulated by expanding the game tree with a predefined pattern of interactions specified by the user. Such a simulation is fast and easy to control. (Chapter 6)

- We propose an algorithm to synthesize a large-scale scene in which the characters densely interact with each other by concatenating Interaction Patches. We apply temporal concatenation to create longer interactions, and spatial concatenation to create interactions involving more characters. (Chapter 6)

7.2 Future Research Directions

In this section, we outline some possible future research directions, which are out of the scope of this research.

7.2.1 Group Interactions

One future direction of this research is to simulate multiple groups of characters interacting intelligently. For example, in a football match, each character must cooperate with its teammates while counteracting the characters in the opponent team. With the Interaction Graph described in Chapter 5, we can simulate a character interacting with one opponent by maximizing the long term rewards. However, when considering the optimal action of every team member in a group to interact with another group, the state space must include all the characters in the scene. Such a state space is too large to be handled.

As a result, instead of searching for the optimal actions for all characters, we have to simplify the problem to generate suboptimal results. One possibility is to apply two-level planning. In the higher level, we can plan the strategy of the whole team, assuming each team member can achieve a predefined set of objectives, such as moving to a nearby position. The planning in such team level should take into account the opponent team, and hence concepts like temporal tree expansion and min-max in Chapter 4 can be applied. Then, in the lower level, each character tries to complete the assigned objective, taking into account only local information such as opponent nearby. By this way, we can simulate two teams of characters interacting with each other, and when focused on individual characters, we can see they interact intelligently as if they are real humans.

7.2.2 Intuitive User Interfaces

The theme of this research is to simulate interactions among multiple characters automatically rather than creating user control interfaces. One of the future research directions is to provide intuitive crowd controlling interfaces, such as those used to adjust the movements of a group of characters, and automatically generate realistic underlying interactions among the characters. Such interfaces are important for controlling crowds in computer games such as those involving war scenes, and can be used to ease the production processes of crowd animations.

Recent researches create convenience user interface to design the movements of a crowd [Kwon, Lee, Lee & Takahashi (2008), Takahashi et al. (2009)], but they do not consider the dense interactions among characters. Recently, some researchers regard interactions as spatio-temporal constraints and apply space-time optimization to synthesize the movements of the characters [Kim et al. (2009)].

However, such constraints have to be defined by the user and the run-time cost for the optimization is still too high for a large crowd. When there are a lot of interactions, such as in a fighting scene between two armies, it is impossible to explicitly indicate the interactions required and solve for the movements during run-time.

We believe that by combining these crowd controlling techniques with Interaction Patches, we can create an intuitive user interface for controlling crowds with dense interactions in real-time. The movements of the crowd are controlled by the user and optimized with Laplacian transformation. The system then monitors the situation of the characters during run-time, and applies Interaction Patches to generate the underlying interactions among characters automatically.

7.2.3 Hierarchical Character Controller

Precomputing the optimal actions in different situations is useful for real-time applications. However, such method requires a manageable size of state space. For example, in the Interaction Graph explained in Chapter 5, the system has to be carefully designed to limit the complexity of the problem, as the state space increases exponentially with such complexity in general.

One possible way to solve the problem is to design a hierarchical state space for complex problems. The lower level state space can provide detail information, but we only consider the subset of the space that is frequently visited to limit the complexity during training. Whenever the character comes to a state that is not considered in the lower state space, we can refer to the higher level state space, which is simpler and acts as an abstraction of the lower one. One example is to represent the higher state space as a general locomotion controller, while the lower state space as a specific interaction controller. For locomotion, only a few information need to be considered, and hence the state space is very simple. On the other hand, interactions require detailed description on the environment and nearby opponents, which leads to a huge state space. Fortunately, a lot of states in such a space are not useful, such as those in which there are no nearby opponents. By combining the two spaces we can create a simpler controller for faster training and lower memory requirement. Although the hierarchy is manually designed in the previous example, it is worth to research on automatic methods to evaluate the optimal hierarchy.

7.2.4 Run-time Learning

The advantage of machine learning algorithms such as reinforcement learning is that we do not need to explicitly design an algorithm to control a character. Instead, we only need to setup an environment and let the character to experience the consequence of conducting different actions. During such training phase, the characters can learn from experiences to perform actions that benefit them the most in the future. However, the training phase takes too long to be an on-line process especially for complex problems. The behaviors of characters have to be precomputed and applied during run-time. As a result, the behavior of the trained character will not be updated during run-time. This is especially important in applications like games, in which there may be unexpected user behavior, and the trained character will continuously perform sub-optimal actions.

In theory, it is possible to run the training process during run-time. However, in practice, we can only get a few numbers of training samples during run-time for a reasonable duration. It is important to research on how to generalize the training process such that it can be performed in run-time. One possible solution is to parameterize the reward function. Instead of updating a subset of state, a training sample will update the parameter of the reward function that affects the whole space. By this way, the character can learn from real experience based on run-time information and behave intelligently.

7.2.5 Interaction Adjustment

Our research focuses on synthesizing realistic interactions rather than adjusting existing interactions. The Interaction Patches in Chapter 6 represent realistic segments of interactions, but how much we can adjust the patches while maintaining the context of the interactions requires further researches.

In general, there could be two levels of adjustments. For the basic level of adjustments, we target for adjusting the position and the orientation of the characters while maintaining the features of the interaction. Recent researches suggest that interpolation of motions in the latent space can create realistic motions that satisfy low dimensional task constraints [Bitzer et al. (2008)]. It would be interesting to investigate the possibility to apply similar methods for the interactions of two characters. On the other hand, the advance level of adjustments requires switching the actions during the interaction in order to keep the context of the interaction.

For example, suppose there is an Interaction Patch in which the attacker punches the opponent. When the patch is edited and the attacker can no longer reach the opponent, the attacker may need to switch the punch with a kick in order to attack its opponent. While it is still unclear how to represent the logical similarity between discrete actions during interactions, further research is required to create a general algorithm that can switch actions to maintain the context of interactions.

7.3 Publications

The concepts related to temporal tree expansion in Chapter 4 are included in:

- Hubert P. H. Shum, Komura Taku & Shuntaro Yamazaki (2007), 'Simulating competitive interactions using singly captured motions', *in* 'VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology', ACM, New York, NY, USA, pp. 65-72

The concepts related to Interaction Graph in Chapter 5 are included in:

- Hubert P. H. Shum, Komura Taku & Shuntaro Yamazaki (2008), 'Simulating interactions of avatars in high dimensional state space', *in* 'I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games', ACM, New York, NY, USA, pp. 131-138

The concepts related to Interaction Patches in Chapter 6 are included in:

- Hubert P. H. Shum, Komura Taku, Masashi Shiraishi & Shuntaro Yamazaki (2008), 'Interaction patches for multi-character animation', *in ACM Trans. Graph.* **27**(5), pp. 1–8

7.4 Commercialization

As our research is closely related to the gaming and movie industries, it is important to demonstrate the possibility in applying the works in practical applications. This research is funded by the Initiating Knowledge Transfer Fund from the University of Edinburgh to commercialize the concepts related to Interaction Patches. A patent has been applied and is now pending. We started a commercialization project in 2009 to create a user interface in Maya, one of the most popular 3D computer graphics software, for generating crowd with dense interactions. The project is on-going and a demonstration program is expected to be available in early 2010.

Appendix A

Runtime Synthesis

Although we use motions captured by real human to generate the animation, when applying them for interaction simulation, the quality the resultant animation is usually below standard. There are artifacts due to the transition periods of the motion graph, unexpected collisions of body parts, and the change of foot contact states. Furthermore, instead of simply displaying the captured motions, we wish to make adjustments to create more realistic interactions. For example, when a character is being hit, we push the colliding parts backwards to emphasize the impact.

We create a particle system based on the Open Dynamic Engine (ODE) [Smith (2008)] framework to synthesize the body postures during runtime, with reference to the captured motion and the adjustment required to simulate the poses. The ODE framework provides physical simulation, while we indicate the appropriate forces and torques to be applied for each joint of the body. Our system can simulate smooth, realistic motions capable of minor adjustments to the body posture.

A.1 Character and World Modeling

In our system, the body segments of a character are modeled by simple rectangular polygons for faster collision detection. The body segments are connected with ball joints, indicating that each body segment has 3 degrees of freedom in rotation. Each character is represented by 25 joints and 19 segments. In our particle system, each joint acts as a particle and each body segment act as an elastic spring. Since the sizes of the body segments are constant, the elasticity of all the springs is zero.

We create an infinity large plane in the ODE world as the floor plane, which provide supporting force to the characters. Gravity is implemented such that when no control force is applied, the character falls onto the ground as a rag doll. This feature is used to model characters dying when being hit or pushed.

A.2 Soft Posture Constraints

In this section, we explain the process to define the required posture by soft posture constraints. These soft constraints are presented as the target positions of all particles, which represent the joints of the character body. Notice that we only consider the positions of the particles and do not enforce segment lengths between them. Hence, the positions may refer to a physically invalid posture and require the posture solve explained in Section A.4 to produce a valid posture. Furthermore, the soft posture constraints only define the desired posture. It is not guaranteed that such constraints will be met in the final result.

The target position of each joint is initialized with the posture of the next frame in the captured motion data. Since the motions are stored as joint angles, forward kinematics is use to calculate the joint positions. The results are used as the initial target positions of the particles. The advantage of using joint positions rather than joint angles to describe the soft posture constraints is that positions are more trivial to human understanding, and hence ease the process to adjust the required postures.

Then, we adjust the target positions based on the simulation requirements, such as changing the hitting position of a punch. We only need to adjust the subset of joints that are explicitly related. For example, when adjusting a punch, we only need to adjust the target position of the hand, although the upper arm and lower arm should be adjusted accordingly as well. Such implicit related joints will be handled by the posture solver in Section A.4. We apply adjustments of target particle positions in four ways:

- During boxing, when one character hit its opponent, the attacking joint may not be accurately landed to the opponent. In such situations, we update target positions of the attacking joints, mostly hands and feet, to the opponent for the frames before the hit. Our posture solver will then produce an ease-in and ease-out effect such that the attacking joints gradually move towards the opponent before the hit and move away after the hit.

- In the chasing and catching simulations, to simulate a character catching another, we set the target positions of the hands to the body of the opponent when the character is close to its opponent. When we solve for the posture, the whole arms will be pulled towards the opponent.

- We apply joint adjustment to simulate parry motion in boxing. Whenever a character is being attacked, the character should try to protect itself if possible. We simulate parry motions for the arms when any of them free, that is, when the character is not using the arms to attack nor defence. When the attacking joint of the opponent becomes close to the character, we set the positions of the forearms to the position of that attacking joint. The forearms will then move towards the attacking joint as if blocking the attack, and restore to the original positions after the attack.

- In the American football simulations, we control the left arm of the offensive player such that it holds the ball. In such situation, the particle based on a fixed location referencing to the local coordinate of the character. More specifically, we define the joint positions for the whole left arm relative to the pelvis position and orientation of the character during the simulating, such that the left arm is bent towards the body as if holding a ball.

A.3 Hard Posture Constraints

Hard posture constraints are also defined by the target positions of particles. Unlike the soft posture constraints, the hard constraints are only defined for a subset of particles, and are guaranteed to be met in the final postures produced by the posture solver in Section A.4.

The hard posture constraints are used to constraint the position of particles. To define a new hard constraint, we create a virtual ball joint between the specified particle and the space, indicating that the particle cannot be translated. Alternatively, we can create a virtual ball joint between two particles, indicating that they are connected. The hard constraints are applied in two ways:

- The supporting feet pattern of all actions are precomputed as explained in Section 3.2. During run-time, if a foot is supported, we add a hard constraint to fix the foot on the floor. When it becomes unsupported, the constraint is removed to resume the movement of the foot.

- When we simulate a character holding luggage, we fix the hands to the surface of the luggage with hard constraints such that they maintain contact with the luggage. Using the same carrying motion with different hard constraint definition, we can simulate a character carrying objects of different sizes. However, when there is a dramatic change in the size of the objects, we have to capture new motions since the whole body movement should be different.

Notice that if multiple hard constraints are created, there may be incompatibility among them, such as fixing the feet while requiring the hands to unreachable positions. In such situation, the posture solver will fail and the resultant posture will appear broken as the segment lengths can no longer be maintained. Therefore, we try to apply as few hard constraints as possible, and make sure the required positions are valid.

A.4 Posture Solver

In this section, we explain the process to solve for a valid posture with the given soft and hard posture constraints.

Our posture solver is a particle system based on the ODE framework. We define the control forces for each individual particle based on the soft posture constraints. On the other hand, the ODE maintains the segment length and segment connectivity defined by joints while applying the control forces. Virtual joints defined by hard posture constraints are also maintained. Since the segment length is fixed, when a force is applied to a particle, the rest of the particles will be dragged to such direction. For each time step, the control force for each particle is calculated by PD control:

$$F = K_e(P_{target} - P_{current}) + K_d(P'_{target} - P'_{current}) \tag{A.1}$$

where P_{target} is the target position of the particle as defined in the soft posture constraints, $P_{current}$ is the current position, P'_{target} and $P'_{current}$ are the respective derivative, K_e is the elasticity gain and K_d is the damping gain. A high K_e can improve the responsiveness of the character, while a high K_d produce more stable movements. We manually tune the smallest possible K_e and K_d as a particle system with high control forces is not stable. Furthermore, the magnitude of the resultant force F is bounded by a predefined value to avoid unexpected high control force while the target values are very different from the current ones. In case a character is being pushed or hit, extra control force is applied to simulate the impact. The force is added to the particles being disturbed by referencing the velocity of the disturbing particles.

One problem for our particle system is that because the particles are generated directly using joint positions, we cannot represent the rotational movements along the body segments. One solution is to sample multiple particles based on fixed offsets from each joint. However, this increases the complexity of the system unnecessarily. Instead, we construct a hybrid system taking into account both controlling forces and rotational torques, with the latter being a supporting element. The controlling torque of a particle is calculated as:

$$T = K_{\varepsilon}(\theta_{motion} - \theta_{current}) + K_{\delta}(\theta'_{motion} - \theta'_{current}) \tag{A.2}$$

where θ_{motion} is the orientation of the joint defined in the source motion data, $\theta_{current}$ is the current orientation of the corresponding particle, θ'_{motion} and $\theta'_{current}$ are the respective derivative, K_{ε} and K_{δ} are the hand tuned elasticity gain and damping gain. Similar to the force calculation, the torque T is bounded by a predefined value.

Finally, a resultant posture is generated by the physical simulation engine of the ODE. Collision detection is carried out between body segments such that they do not overlap when the control signals are applied. The resultant posture is the equilibrium state of the particles when all control forces are applied. Thus, it represents the posture that can satisfy most of the soft constraints while maintaining the hard ones.

A.5 Summary

We designed a particle system based on the ODE framework. By defining the constraints of the required posture based on the motion data and simulation requirements, we can calculate the control force required for each particle. The control forces are applied to the physical simulation engine of the ODE to solve for the resultant posture. Our system can synthesize smooth and realistic movements during run-time. We can also adjust the movement of a specific joint, and constraint joints such as the supporting feet.

Bibliography

Abe, Y., da Silva, M. & Popović, J. (2007), Multiobjective control with frictional contacts, *in* ‘SCA ’07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 249–258.

Abe, Y. & Popović, J. (2006), Interactive animation of dynamic manipulation, *in* ‘SCA ’06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 195–204.

Adriano Macchietto, Victor Zordan, C. R. S. (2009), ‘Momentum control for balance’, *ACM Trans. Graph.* .

Arikan, O. & Forsyth, D. A. (2002), ‘Interactive motion generation from examples’, *ACM Trans. Graph.* **21**(3), 483–490.

Arikan, O., Forsyth, D. A. & O’Brien, J. F. (2005), Pushing people around, *in* ‘SCA ’05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation’, ACM, New York, NY, USA, pp. 59–66.

Avis, D., Rosenberg, G., Savani, R. & Stengel, B. (2010), ‘Enumeration of nash equilibria for two-player games’, *Economic Theory* **42**(1), 9–37.
URL: *http://ideas.repec.org/a/spr/joecth/v42y2010i1p9-37.html*

Barbič, J., da Silva, M. & Popović, J. (2009), ‘Deformable object animation using reduced optimal control’, *ACM Trans. Graph.* **28**(3), 1–9.

Barbič, J. & Popović, J. (2008), ‘Real-time control of physically based simulations using gentle forces’, *ACM Trans. Graph.* **27**(5), 1–10.

Beaudoin, P., Coros, S., van de Panne, M. & Poulin, P. (2008), Motion-motif graphs, *in* ‘Symposium on Computer Animation 2008’, pp. 117–126.

Bitzer, S., Havoutis, I. & Vijayakumar, S. (2008), Synthesising novel movements through latent space modulation of scalable control policies, *in* ‘SAB ’08: Proceedings of the 10th international conference on Simulation of Adaptive Behavior’, Springer-Verlag, Berlin, Heidelberg, pp. 199–209.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, *in* ‘COLT ’92: Proceedings of the fifth annual workshop on Computational learning theory’, ACM, New York, NY, USA, pp. 144–152.

Bouzy, B. & Chaslot, G. (2006), Monte-carlo go reinforcement learning experiments, *in* ‘Computational Intelligence and Games, 2006 IEEE Symposium on’, pp. 187–194.

Brand, M. & Hertzmann, A. (2000), Style machines, *in* ‘SIGGRAPH ’00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques’, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 183–192.

Bruderlin, A. & Williams, L. (1995), Motion signal processing, *in* ‘SIGGRAPH ’95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques’, ACM, New York, NY, USA, pp. 97–104.

Carmel, D. & Markovitch, S. (1996), Learning and using opponent models in adversary search, Technical Report CIS9609, Technion.
URL: *http://www.cs.technion.ac.il/ shaulm/papers/pdf/Carmel-Markovitch-CIS9609.pdf*

Chai, J. & Hodgins, J. K. (2005), Performance animation from low-dimensional control signals, *in* ‘SIGGRAPH ’05: ACM SIGGRAPH 2005 Papers’, ACM, New York, NY, USA, pp. 686–696.

Chiu, B., Zordan, V. & Wu, C.-C. (2007), State-annotated motion graphs, *in* ‘VRST ’07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology’, ACM, New York, NY, USA, pp. 73–76.

Choi, M. G., Lee, J. & Shin, S. Y. (2003), ‘Planning biped locomotion using motion capture data and probabilistic roadmaps’, *ACM Trans. Graph.* **22**(2), 182–203.

Cooper, S., Hertzmann, A. & Popović, Z. (2007), ‘Active learning for real-time motion controllers’, *ACM Trans. Graph.* **26**(3), 5.

Coros, S., Beaudoin, P., Yin, K. K. & van de Pann, M. (2008), Synthesis of constrained walking skills, *in* ‘SIGGRAPH Asia ’08: ACM SIGGRAPH Asia 2008 papers’, ACM, New York, NY, USA, pp. 1–9.

Cortes, C. & Vapnik, V. (1995), Support-vector networks, *in* ‘Machine Learning’, pp. 273–297.

da Silva, M., Abe, Y. & Popović, J. (2008), Interactive simulation of stylized human locomotion, *in* ‘SIGGRAPH ’08: ACM SIGGRAPH 2008 papers’, ACM, New York, NY, USA, pp. 1–10.

Donkers, H. H. L. M. (2003), Nosce Hostem: Searching with Opponent Models, PhD thesis, Universteit Maastricht.

Donkers, H. H. L. M., Uiterwijk, J. W. H. M. & van den Herik, H. J. (2001), ‘Probabilistic opponent-model search’, *Inf. Sci.* **135**(3-4), 123–149.

Donkers, J., van den Herik, H. J. & Uiterwijk, J. W. H. M. (2004), Probabilistic opponent-model search in bao, *in* ‘ICEC’, pp. 409–419.

Esteves, C., Arechavaleta, G. & Laumond, J.-P. (2005), Motion planning for human-robot interaction in manipulation tasks, *in* ‘Mechatronics and Automation, 2005 IEEE International Conference’, Vol. 4, pp. 1766–1771 Vol. 4.

Fang, A. C. & Pollard, N. S. (2003), ‘Efficient synthesis of physically valid human motion’, *ACM Trans. Graph.* **22**(3), 417–426.

Feurtey, F. (2000), Simulating the collision avoidance behavior of pedestrians, Master’s thesis, University of Tokyo, Department of Electronic Engineering.

Fishman, G. S. (1996), *Monte Carlo: Concepts, algorithms, and applications*, Springer Series in Operations Research, Springer-Verlag, New York.

Flagg, M., Nakazawa, A., Zhang, Q., Kang, S. B., Ryu, Y. K., Essa, I. & Rehg, J. M. (2009), Human video textures, *in* ‘I3D ’09: Proceedings of the 2009 symposium on Interactive 3D graphics and games’, ACM, New York, NY, USA, pp. 199–206.

Fujimoto, Y., Obata, S. & Kawamura, A. (1998), Robust biped walking with active interaction control between foot and ground, *in* 'Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on', Vol. 3, pp. 2030–2035 vol.3.

Gleicher, M. (1997), Motion editing with spacetime constraints, *in* 'SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics', ACM, New York, NY, USA, pp. 139–ff.

Gleicher, M. & Litwinowicz, P. (1998), 'Constraint-based motion adaptation', *The Journal of Visualization and Computer Animation* **9**(2), 65–94.

Gleicher, M., Shin, H. J., Kovar, L. & Jepsen, A. (2003), Snap-together motion: assembling run-time animations, *in* 'I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics', ACM, New York, NY, USA, pp. 181–188.

Grochow, K., Martin, S. L., Hertzmann, A. & Popović, Z. (2004), Style-based inverse kinematics, *in* 'SIGGRAPH '04: ACM SIGGRAPH 2004 Papers', ACM, New York, NY, USA, pp. 522–531.

Heck, R. & Gleicher, M. (2007), Parametric motion graphs, *in* 'I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games', ACM, New York, NY, USA, pp. 129–136.

Helbing, D., Farkas, I. & Vicsek, T. (2000), 'Simulating dynamical features of escape panic.', *Nature* **407**(6803), 487–490.

Helbing, D. & Molnar, P. (1995), 'Social force model for pedestrian dynamics', *Physical Review E* **51**, 4282.

Ho, E. S. L. & Komura, T. (2007*a*), Planning tangling motions for humanoids, *in* 'Humanoid '07: Proceedings of the IEEE-RAS 2009 International Conference on Humanoid Robots'.

Ho, E. S. L. & Komura, T. (2007*b*), Wrestle alone: Creating tangled motions of multiple avatars from individually captured motions, *in* 'PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications', IEEE Computer Society, Washington, DC, USA, pp. 427–430.

Ho, E. S. L. & Komura, T. (2009*a*), ‘Character motion synthesis by topology coordinates’, *Computer Graphics Forum* **28**(2), 299–308.

Ho, E. S. L. & Komura, T. (2009*b*), ‘Indexing and retrieving motions of characters in close contact’, *IEEE Transactions on Visualization and Computer Graphics* **15**(3), 481–492.

Hodgins, J. (9-11 Apr 1991), ‘Biped gait transitions’, *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* pp. 2092–2097 vol.3.

Hodgins, J. K., Wooten, W. L., Brogan, D. C. & O’Brien, J. F. (1995), Animating human athletics, *in* ‘SIGGRAPH ’95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques’, ACM, New York, NY, USA, pp. 71–78.

Hodgins, J. & Raibert, M. (Jun 1991), ‘Adjusting step length for rough terrain locomotion’, *Robotics and Automation, IEEE Transactions on* **7**(3), 289–298.

Hsu, E., Pulli, K. & Popović, J. (2005), ‘Style translation for human motion’, *ACM Trans. Graph.* **24**(3), 1082–1089.

Igarashi, T., Moscovich, T. & Hughes, J. F. (2005), ‘As-rigid-as-possible shape manipulation’, *ACM Trans. Graph.* **24**(3), 1134–1141.

Ikemoto, L., Arikan, O. & Forsyth, D. (2005), Learning to move autonomously in a hostile world, *in* ‘SIGGRAPH ’05: ACM SIGGRAPH 2005 Sketches’, ACM, New York, NY, USA, p. 46.

Ikemoto, L., Arikan, O. & Forsyth, D. (2006), Knowing when to put your foot down, *in* ‘I3D ’06: Proceedings of the 2006 symposium on Interactive 3D graphics and games’, ACM, New York, NY, USA, pp. 49–53.

Ikemoto, L., Arikan, O. & Forsyth, D. (2007), Quick transitions with cached multi-way blends, *in* ‘I3D ’07: Proceedings of the 2007 symposium on Interactive 3D graphics and games’, ACM, New York, NY, USA, pp. 145–151.

Jain, S., Ye, Y. & Liu, C. K. (2009), ‘Optimization-based interactive motion synthesis’, *ACM Transaction on Graphics* **28**(1), 1–10.

Jain, T. & Liu, C. K. (2009), Interactive synthesis of human-object interaction, *in* ‘SCA ’09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics symposium on Computer animation’, ACM, New York, NY, USA.

James, D. L., Twigg, C. D., Cove, A. & Wang, R. Y. (2007), ‘Mesh ensemble motion graphs: Data-driven mesh animation with constraints’, *ACM Trans. Graph.* **26**(4), 17.

Jingjing Meng, Junsong Yuan, M. H. & Wu, Y. (2008), Mining motifs from huma motion, *in* ‘EUROGRAPHICS ’08: Proceedings of the 2008 Eurographics’.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Yokoi, K. & Hirukawa, H. (2002), A realtime pattern generator for biped walking, *in* ‘Robotics and Automation, 2002. Proceedings. ICRA ’02. IEEE International Conference on’, Vol. 1, pp. 31–37 vol.1.

Kajita, S., Matsumoto, O. & Saigo, M. (2001), Real-time 3d walking pattern generation for a biped robot with telescopic legs, *in* ‘Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on’, Vol. 3, pp. 2299–2306 vol.3.

Kajita, S., Yokoi, K., Saigo, M. & Tanie, K. (2001), Balancing a humanoid robot using backdrive concerned torque control and direct angular momentum feedback, *in* ‘Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on’, Vol. 4, pp. 3376–3382 vol.4.

Kim, M., Hyun, K. L., Kim, J. & Lee, J. (2009), ‘Synchronized multi-character motion editing’, *ACM Trans. Graph.* .

Komura, T., Ho, E. S. L. & Lau, R. W. H. (2005), ‘Animating reactive motion using momentum-based inverse kinematics: Motion capture and retrieval’, *Comput. Animat. Virtual Worlds* **16**(3-4), 213–223.

Komura, T., Leung, H., Kudoh, S. & Kuffner, J. (2005), A feedback controller for biped humanoids that can counteract large perturbations during gait, *in* ‘Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on’, pp. 1989–1995.

Komura, T., Leung, H. & Kuffner, J. (2004), Animating reactive motions for biped locomotion, *in* 'VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology', ACM, New York, NY, USA, pp. 32–40.

Kovar, L. & Gleicher, M. (2003), Flexible automatic motion blending with registration curves, *in* 'SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation', Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 214–224.

Kovar, L. & Gleicher, M. (2004), 'Automated extraction and parameterization of motions in large data sets', *ACM Trans. Graph.* **23**(3), 559–568.

Kovar, L., Gleicher, M. & Pighin, F. H. (2002), 'Motion graphs', *ACM Trans. Graph.* **21**(3), 473–482.

Kudoh, S. & Komura, T. (2003), C2 continuous gait-pattern generation for biped robots, *in* 'Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on', Vol. 2, pp. 1135–1140 vol.2.

Kwon, T., Cho, Y.-S., Park, S. I. & Shin, S. Y. (2008), 'Two-character motion analysis and synthesis', *IEEE Transactions on Visualization and Computer Graphics* **14**(3), 707–720.

Kwon, T., Lee, K. H., Lee, J. & Takahashi, S. (2008), Group motion editing, *in* 'SIGGRAPH '08: ACM SIGGRAPH 2008 papers', ACM, New York, NY, USA, pp. 1–8.

Kwon, T. & Shin, S. Y. (2005), Motion modeling for on-line locomotion synthesis, *in* 'SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation', ACM, New York, NY, USA, pp. 29–38.

Lai, Y.-C., Chenney, S. & Fan, S. (2005), Group motion graphs, *in* 'SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation', ACM, New York, NY, USA, pp. 281–290.

Lau, M. & Kuffner, J. J. (2005), Behavior planning for character animation, *in* 'SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation', ACM, New York, NY, USA, pp. 271–280.

Lau, M. & Kuffner, J. J. (2006), Precomputed search trees: planning for interactive goal-driven animation, *in* ‘SCA ’06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 299–308.

LaValle, S. (1998), ‘Rapidly-exploring random trees: A new tool for path planning’.

LaValle, S. & Kuffner, J. (2000), ‘Rapidly-exploring random trees: Progress and prospects’. In Workshop on the Algorithmic Foundations of Robotics.

Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K. & Pollard, N. S. (2002), ‘Interactive control of avatars animated with human motion data’, *ACM Trans. Graph.* **21**(3), 491–500.

Lee, J. & Lee, K. H. (2004), Precomputing avatar behavior from human motion data, *in* ‘SCA ’04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 79–87.

Lee, J. & Lee, K. H. (2006), ‘Precomputing avatar behavior from human motion data’, *Graph. Models* **68**(2), 158–174.

Lee, K. H., Choi, M. G., Hong, Q. & Lee, J. (2007), Group behavior from video: a data-driven approach to crowd simulation, *in* ‘SCA ’07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 109–118.

Lee, K. H., Choi, M. G. & Lee, J. (2006), ‘Motion patches: building blocks for virtual environments annotated with motion data’, *ACM Trans. Graph.* **25**(3), 898–906.

Lerner, A., Chrysanthou, Y. & Lischinski, D. (2007), ‘Crowds by example’, *Computer Graphics Forum (Proceedings of Eurographics)* **26**(3).

Li, Q., Takanishi, A. & Kato, I. (1992), Learning control of compensative trunk motion for biped walking robot based on zmp stability criterion, *in* ‘Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on’, Vol. 1, pp. 597–603.

Liu, C. K., Hertzmann, A. & Popović, Z. (2005), 'Learning physics-based motion style with nonlinear inverse optimization', *ACM Trans. Graph.* **24**(3), 1071–1081.

Liu, C. K., Hertzmann, A. & Popović, Z. (2006), Composition of complex optimal multi-character motions, *in* 'SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation', Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 215–222.

Liu, C. K. & Popović, Z. (2002), Synthesis of complex dynamic character motion from simple animations, *in* 'SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques', ACM, New York, NY, USA, pp. 408–416.

Liu, Z., Gortler, S. J. & Cohen, M. F. (1994), Hierarchical spacetime control, *in* 'SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques', ACM, New York, NY, USA, pp. 35–42.

Lo, W.-Y. & Zwicker, M. (2008), 'Real-time planning for parameterized human motion', *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* .

Loscos, C., Marchal, D. & Meyer, A. (2003), Intuitive crowd behaviour in dense urban environments using local laws, *in* 'TPCG '03: Proceedings of the Theory and Practice of Computer Graphics 2003', IEEE Computer Society, Washington, DC, USA, p. 122.

Makar, R., Mahadevan, S. & Ghavamzadeh, M. (2001), Hierarchical multi-agent reinforcement learning, *in* 'AGENTS '01: Proceedings of the fifth international conference on Autonomous agents', ACM, New York, NY, USA, pp. 246–253.

Marco da Silva, Yeuhi Abe, J. P. (2008), 'Simulation of human motion data using short-horizon model-predictive control', *Computer Graphics Forum* **27**(2), 371–380.

McCann, J. & Pollard, N. (2007), 'Responsive characters from motion fragments', *ACM Trans. Graph.* **26**(3), 6.

Ménardais, S., Kulpa, R., Multon, F. & Arnaldi, B. (2004), Synchronization for dynamic blending of motions, *in* 'SCA '04: Proceedings of the 2004 ACM

SIGGRAPH/Eurographics symposium on Computer animation', Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 325–335.

Mitake, H., Asano, K., Aoki, T., Marc, S., Sato, M. & Hasegawa, S. (2009), 'Physics-driven multi dimensional keyframe animation for artist-directable interactive character', *Computer Graphics Forum* **28**(2), 279–287.

Muico, U., Lee, Y., Popović, J. & Popović, Z. (2009), 'Contact-aware nonlinear control of dynamic characters', *ACM Transactions on Graphics* **28**(3).

Mukai, T. & Kuriyama, S. (2005), 'Geostatistical motion interpolation', *ACM Trans. Graph.* **24**(3), 1062–1070.

Musse, S. R., Babski, C., Capin, T. & Thalmann, D. (1998), Crowd modelling in collaborative virtual environments, *in* 'VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology', ACM, New York, NY, USA, pp. 115–123.

Musse, S. R. & Thalmann, D. (1997), A model of human crowd behavior: Group inter-relationship and collision detection analysis, *in* 'Proc. Workshop of Computer Animation and Simulation of Eurographics97', pp. 39–51.

Napoleon, Nakaura, S. & Sampei, M. (2002), Balance control analysis of humanoid robot based on zmp feedback control, *in* 'Intelligent Robots and System, 2002. IEEE/RSJ International Conference on', Vol. 3, pp. 2437–2442 vol.3.

Nash, J. (1951), 'Non-cooperative games', *The Annals of Mathematics* **54**(2), 286–295.
URL: *http://dx.doi.org/10.2307/1969529*

Neumann, J. V. & Morgenstern, O. (1944), *Theory of Games and Economic Behavior*, Princeton University Press.
URL: *http://jmvidal.cse.sc.edu/library/neumann44a.pdf*

Nishiwaki, K., Sugihara, T., Kagami, S., Inaba, M. & Inoue, H. (2001), Online mixture and connection of basic motions for humanoid walking control by footprint specification, *in* 'Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on', Vol. 4, pp. 4110–4115 vol.4.

Oshita, M. & Makinouchi, A. (2001), ‘A dynamic motion control technique for human-like articulated figures’, *Computer Graphics Forum (EUROGRAPHICS 2001, Manchester, United Kingdom, September 2001), Vol. 20* **20**(3), 192–202.

Park, S. I., Shin, H. J., Kim, T. H. & Shin, S. Y. (2004), ‘On-line motion blending for real-time locomotion generation: Research articles’, *Comput. Animat. Virtual Worlds* **15**(3-4), 125–138.

Parr, R. & Russell, S. (1998), Reinforcement learning with hierarchies of machines, *in* ‘NIPS ’97: Proceedings of the 1997 conference on Advances in neural information processing systems 10’, MIT Press, Cambridge, MA, USA, pp. 1043–1049.

Playter, R. & Raibert, M. (7-10 Jul 1992), ‘Control of a biped somersault in 3d’, *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on* **1**, 582–589.

Popović, Z. & Witkin, A. (1999), Physically based motion transformation, *in* ‘SIGGRAPH ’99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques’, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 11–20.

Raibert, M. H. & Hodgins, J. K. (1991), ‘Animation of dynamic legged locomotion’, *SIGGRAPH Comput. Graph.* **25**(4), 349–358.

Reitsma, P. S. A. & Pollard, N. S. (2004), Evaluating motion graphs for character navigation, *in* ‘SCA ’04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 89–98.

Reitsma, P. S. A. & Pollard, N. S. (2007), ‘Evaluating motion graphs for character animation’, *ACM Trans. Graph.* **26**(4), 18.

Reynolds, C. (1999), Steering behaviors for autonomous characters, *in* ‘Game Developers Conference 1999’.

Reynolds, C. W. (1987), ‘Flocks, herds and schools: A distributed behavioral model’, *SIGGRAPH Comput. Graph.* **21**(4), 25–34.

Riley, P. F. & Veloso, M. M. (2006), ‘Coach planning with opponent models for distributed execution’, *Autonomous Agents and Multi-Agent Systems* **13**(3), 293–325.

Rose, C., Guenter, B., Bodenheimer, B. & Cohen, M. F. (1996), Efficient generation of motion transitions using spacetime constraints, *in* ‘SIGGRAPH ’96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques’, ACM, New York, NY, USA, pp. 147–154.

Rummery, G. A. & Niranjan, M. (1994), On-line Q-learning using connectionist systems, Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Safonova, A. & Hodgins, J. K. (2007), Construction and optimal search of interpolated motion graphs, *in* ‘SIGGRAPH ’07: ACM SIGGRAPH 2007 papers’, ACM, New York, NY, USA, p. 106.

Safonova, A. & Hodgins, J. K. (2008), ‘Synthesizing human motion from intuitive constraints’, *Studies in Computational Intelligence* **159/2008**, 15–39.

Safonova, A., Hodgins, J. K. & Pollard, N. S. (2004), Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces, *in* ‘SIGGRAPH ’04: ACM SIGGRAPH 2004 Papers’, ACM, New York, NY, USA, pp. 514–521.

Sang Il Park, Taesoo Kwon, H. J. S. & Shin, S. Y. (2004), ‘Analysis and synthesis of interactive two-character motions’.

Shannon, C. E. (1988), ‘Programming a computer for playing chess’, pp. 2–13.

Shapiro, A., Kallmann, M. & Faloutsos, P. (2007), Interactive motion correction and object manipulation, *in* ‘I3D ’07: Proceedings of the 2007 symposium on Interactive 3D graphics and games’, ACM, New York, NY, USA, pp. 137–144.

Shen, J., Gu, G. & Liu, H. (2006), Multi-agent hierarchical reinforcement learning by integrating options into maxq, *in* ‘IMSCCS ’06: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - Volume 1 (IMSCCS’06)’, IEEE Computer Society, Washington, DC, USA, pp. 676–682.

Shin, H. J. & Oh, H. S. (2006), Fat graphs: constructing an interactive character with continuous controls, *in* ‘SCA ’06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 291–298.

Shiratori, T. & Hodgins, J. K. (2008), Accelerometer-based user interfaces for the control of a physically simulated character, *in* ‘SIGGRAPH Asia ’08: ACM SIGGRAPH Asia 2008 papers’, ACM, New York, NY, USA, pp. 1–9.

Smith, R. (2008), ‘Open dynamics engine’. http://www.ode.org/.
URL: *http://www.ode.org/*

Sok, K. W., Kim, M. & Lee, J. (2007), Simulating biped behaviors from human motion data, *in* ‘SIGGRAPH ’07: ACM SIGGRAPH 2007 papers’, ACM, New York, NY, USA, p. 107.

Solan, E. & Vieille, N. (2010), ‘Computing uniformly optimal strategies in two-player stochastic games’, *Economic Theory* **42**(1), 237–253.
URL: *http://ideas.repec.org/a/spr/joecth/v42y2010i1p237-253.html*

Stolle, M. & Precup, D. (2002), Learning options in reinforcement learning, *in* ‘Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation’, Springer-Verlag, London, UK, pp. 212–223.

Sung, M., Gleicher, M. & Chenney, S. (2004), ‘Scalable behaviors for crowd simulation’.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA.

Takahashi, S., Yoshida, K., Kwon, T., Lee, K. H., Lee, J. & Shin, S. Y. (2009), ‘Spectral-based group formation control.’, *Comput. Graph. Forum* **28**(2), 639–648.

Thore Graepel, Ralf Herbrich, J. G. (2004), Learning to fight, *in* ‘Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Educatio’.

Treuille, A., Cooper, S. & Popović, Z. (2006), ‘Continuum crowds’, *ACM Trans. Graph.* **25**(3), 1160–1168.

Treuille, A., Lee, Y. & Popović, Z. (2007), ‘Near-optimal character animation with continuous control’, *ACM Trans. Graph.* **26**(3), 7.

Tsai, Y.-Y., Lin, W.-C., Cheng, K. B., Lee, J. & Lee, T.-Y. (2009), ‘Real-time physics-based 3d biped character animation using an inverted pendulum model’, *IEEE Transactions on Visualization and Computer Graphics* .

Van De Panne, M. (Mar 1996), ‘Parameterized gait synthesis’, *Computer Graphics and Applications, IEEE* **16**(2), 40–49.

Wan-Yen Lo, M. Z. (2008), Real-time planning for parameterized human motion, *in* ‘SCA ’08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation’.

Wang, J. & Bodenheimer, B. (2003), An evaluation of a cost metric for selecting transitions between motion segments, *in* ‘SCA ’03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation’, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 232–238.

Watkins, C. J. C. H. (1989), Learning from Delayed Rewards, PhD thesis, King’s College, Cambridge, UK.

Witkin, A. & Kass, M. (1988), Spacetime constraints, *in* ‘SIGGRAPH ’88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques’, ACM, New York, NY, USA, pp. 159–168.

Yamane, K. & Nakamura, Y. (2000), Dynamics filter: Concept and implementation of online motion generator for human figures, *in* ‘ICRA: Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 1’, pp. 688–694.

Ye, Y. & Liu, C. K. (2008), ‘Animating responsive characters with dynamic constraints in near-unactuated coordinates’, *ACM Trans. Graph.* **27**(5), 1–5.

Yeh, H., Curtis, S., Patil, S., van den Berg, J., Manocha, D. & Lin, M. (2008), Composite agents, *in* ‘SCA ’08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation’, ACM.

Yersin, B., Maïm, J., Pettré, J. & Thalmann, D. (2009), Crowd patches: populating large-scale virtual environments for real-time applications, *in* ‘I3D ’09:

Proceedings of the 2009 symposium on Interactive 3D graphics and games', ACM, New York, NY, USA, pp. 207–214.

Yin, K., Loken, K. & van de Panne, M. (2007), 'Simbicon: Simple biped locomotion control', *ACM Trans. Graph.* **26**(3), Article 105.

Zhao, L. & Safonova, A. (2008), Achieving good connectivity in motion graphs, *in* 'Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation'.

Zordan, V. B. & Hodgins, J. K. (2002), Motion capture-driven simulations that hit and react, *in* 'SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation', ACM, New York, NY, USA, pp. 89–96.

Zordan, V. B., Majkowska, A., Chiu, B. & Fast, M. (2005), 'Dynamic response for motion capture animation', *ACM Trans. Graph.* **24**(3), 697–701.

Zordan, V., Macchietto, A., Medin, J., Soriano, M., Wu, C.-C., Metoyer, R. & Rose, R. (2007), Anticipation from example, *in* 'VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology', ACM, New York, NY, USA, pp. 81–84.