

**SCALABLE, CONTROLLABLE, EFFICIENT AND CONVINCING CROWD  
SIMULATION**

by

Mankyu Sung

A dissertation submitted in partial fulfillment of  
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2005

© Copyright by Mankyu Sung 2005

All Rights Reserved

To my family and parents.

# ACKNOWLEDGMENTS

First of all, I would like to appreciate my advisor, professor Michael Gleicher for being such a wonderful advisor. Throughout my research, his patience, care and support helped me get over many hurdles during my graduate years. He always gave me freedom to pursue my own research topic and gave me precious intuitions and guidance that I would not get from any other person. I only hope that I can have a chance to practice what I learned from him to my own students or colleagues. I also would like to thank all my committee members, professor Stephen Chenney, Chuck Dyer, Nicolar Ferrier and Jerry Zhu. Special thanks to professor Chenney for his willingness to discuss about research and help on writing papers.

Second, I appreciate my parents, who supported and granted me when I decided to go overseas for my studying. At the age of 30, it was not easy decision for me to go back to school, but they always believed in me, and that belief motivated me to keep continuing the “not-always-fun” graduate studies.

I would like to thank all UW graphics group members, including Michael(another Michael), Rachel, Tom, Greg, Feng, Yu-Chi, Shaohua, Gudong and old members, Lucas, Alex and Eric for making such a fun environment to work. I strongly believe that our open-lab style is the best environment for graduate students. We can discuss about our own research together, cheer up when we are depressed and have fun together for relax. In addition to our group members, I also thank one of the my co-workers, undergraduate student Aaron, for his hard working of visualization of crowd simulation through a game engine. Although they are not in our lab anymore, I would like to thank former post doctor, Dr. Hyunjoon Shin for allowing me to use his Snap-Together-Motion system and Dr. Lucas Kovar for helping me on writing a paper and preparing conference

presentation. Rachel, Tom and Greg helped a lot on proofreading of this dissertation. I appreciate them again.

Finally, I should mention my wife Sungeun. Without her, I don't think I could go through the graduate study here. I really understand that it was not easy to be a fulltime housewife for a graduate student husband and a full time mom for two children at the same time. But, she has sacrificed her personal life and always been a good mom for children and a wonderful wife for me. Thanks, Sungeun. This dissertation is for you. Last but not least, my two little angels, Hyunju and Hyunji, helped me to overcome hard time by showing me their beautiful smiles. Girls, I love you!

**DISCARD THIS PAGE**

# TABLE OF CONTENTS

	Page
<b>ABSTRACT</b> . . . . .	vii
<b>1 Introduction</b> . . . . .	1
1.1 What Are a Crowd and a Crowd Simulation? . . . . .	1
1.2 Motivation . . . . .	2
1.3 Challenges . . . . .	3
1.4 Complexity . . . . .	3
1.5 Research Goal . . . . .	4
1.6 Existing Methods . . . . .	7
1.7 What Makes a Crowd as a Crowd? . . . . .	9
1.8 Thesis Statement . . . . .	10
1.8.1 The High-Level: Situation-Based Simulation . . . . .	10
1.8.2 The Low-Level: Motion Synthesis . . . . .	12
1.8.3 Demand Satisfaction of Two-Level Simulation Framework . . . . .	15
1.9 Contributions . . . . .	16
1.10 Outline of Thesis . . . . .	16
1.11 Overview . . . . .	17
1.11.1 Situation-Based Simulation with Probability Scheme (Chapter 3) . . . . .	17
1.11.2 Collision Detection Using MOBB Trees (Chapter 4) . . . . .	19
1.11.3 Constrained Motion Synthesis (Chapter 5) . . . . .	20
<b>2 Related Work</b> . . . . .	22
2.1 Animating Human Characters . . . . .	22
2.1.1 Graph-Based Motion Synthesis . . . . .	25
2.2 Crowd Modelling . . . . .	28
2.3 Crowd Rendering . . . . .	31
2.4 Collision Detection . . . . .	32
2.5 Commercial Software . . . . .	33

	Page
<b>3 Situation-Based Simulation with the Probability Scheme</b> . . . . .	35
3.1 Introduction . . . . .	35
3.2 Probability Scheme . . . . .	39
3.2.1 Behavior Functions and Behavior Composition . . . . .	39
3.2.2 Default States and Behaviors . . . . .	42
3.3 Situation and Pluggable Character Architecture . . . . .	44
3.3.1 Virtual Sensors and Memory . . . . .	47
3.3.2 Situation Composition . . . . .	49
3.4 Painting Interface . . . . .	49
3.5 Experiments . . . . .	50
3.5.1 Street Environment . . . . .	51
3.5.2 Theater Environment . . . . .	53
3.5.3 Field Environment . . . . .	55
3.5.4 Validation . . . . .	56
3.6 Discussion . . . . .	59
<b>4 Collision Detection Using MOBB Trees</b> . . . . .	61
4.1 Introduction . . . . .	61
4.2 MOBB Trees . . . . .	64
4.2.1 Fitting MOBB Trees . . . . .	65
4.2.2 Intersection Testing . . . . .	67
4.3 Unrestricted MOBB Trees . . . . .	68
4.4 Validation . . . . .	69
4.5 Discussion . . . . .	72
<b>5 Constrained Motion Synthesis</b> . . . . .	79
5.1 Introduction . . . . .	79
5.2 Synthesizing Motion . . . . .	81
5.2.1 Overview . . . . .	81
5.2.2 Creating the Seed Motions . . . . .	85
5.2.3 Adjusting and Merging the Seed Motions . . . . .	87
5.3 Validation . . . . .	90
5.4 Discussion . . . . .	98
<b>6 Validation</b> . . . . .	101
6.1 A Virtual City . . . . .	101



## Appendix

	Page
6.2 Results . . . . .	102
6.3 Discussion . . . . .	110
<b>7 Conclusion . . . . .</b>	<b>112</b>
7.1 Summary . . . . .	112
7.2 Applications . . . . .	114
7.3 Limitations and Future Work . . . . .	116
7.3.1 Limits of the two-level crowd simulation framework . . . . .	116
7.3.2 Expressiveness . . . . .	117
7.3.3 Limits to Motion Data . . . . .	118
7.3.4 Off-line Constrained Motion Synthesis . . . . .	119
7.3.5 Character Rendering . . . . .	119
7.3.6 Intuitive User Interface . . . . .	120
<b>LIST OF REFERENCES . . . . .</b>	<b>121</b>
<b>APPENDIX Behavior Functions . . . . .</b>	<b>129</b>

# ABSTRACT

Crowd simulation is a difficult task — not only because we require extra computational time for simulating a lot of characters, but also because the crowd behaviors are highly complex and it is hard for them to maintain convincingness. We establish four specific demands that good-quality crowd simulation should satisfy, which are scalability, controllability, efficiency and convincingness, and propose a novel two-level crowd simulation framework that satisfies these demands at the same time.

At the high-level, we adopt a distributed crowd control mechanism called a *situation* in which environmental-specific or social relationship-specific information is automatically added to characters when they are in the situations. At the low-level, and in relation to the applications, the *probability scheme* or the *constrained motion synthesis* is called to synthesize motions for each individual character. The probability scheme, which computes the probabilities of all available actions and selects the next action through random sampling, facilitates simulation of the short-term aggregate behaviors. On the other hand, the constrained motion synthesis, which synthesizes motions that meet constraints, is useful when the behaviors need long-term planning. In addition, we address the issue of fast collision detection between motions and, therein, propose the *MOBB* (*Motion-Oriented Bounding Box*) tree representation of motions, which simplifies a motion into a simple bounding box in a spatio-temporal domain. Given the two bounding MOBB trees, the intersection between two bounding boxes is tested hierarchically from the top node to the leaf node,

and this outcome minimizes the number of tests, which makes collision test fast. To validate the satisfaction of the four demands, we perform a series of experiments.

# Chapter 1

## Introduction

This thesis proposes a novel method for simulating 3D virtual crowds that is scalable, efficient, controllable and convincing. As a starting point for our dissertation about the proposed method, this chapter defines the research problems and clarifies the thesis statement.

First, section 1.1 defines *crowds* and *crowd simulation*. Next, section 1.2 explores the motivation underlying this research by presenting potential applications of general crowd simulation. Section 1.3 describes the challenges of crowd simulation. Section 1.4 defines the environmental complexity and the density of crowds, which are important criteria to test the performance of simulation. Section 1.5 presents specific research goals of this thesis. Section 1.6 overviews existing methods and categorizes them. Section 1.7 lists the typical features of crowds that we exploit in our simulation. Section 1.8 presents our research statement and briefly introduces the proposed methods. Section 1.9 summarizes the contributions of this dissertation. Section 1.10 gives the outline of this dissertation. Section 1.11 briefly introduces the subsequent technical chapters.

### 1.1 What Are a Crowd and a Crowd Simulation?

This section defines the term *crowd* and explains what *crowd simulation* means in relation to computer graphics.

In general, a *crowd* is “a large group of individuals in the same physical environment sharing a common goal who may act in a different way rather than when they are alone” [TKOR05]. Crowds are ubiquitous in real life. Hence, according to the paper of Ulicney and Thalmann [UT01], crowds have become an important research area for many scientists since the nineteenth century. In particular, from the perspective of computer graphics research, *crowd simulation* refers to an artificial creation of virtual crowds that mimic the behavior of real crowds. More specifically, our simulation focuses on the *motion generation* of individual characters. The *behavior* of a character, in this context, is then defined as change of motion choice that the character makes over time. Therefore, *crowd simulation* is defined as the collection of every individual’s behaviors over time.

## 1.2 Motivation

This section explores the motivation underlying our research by enumerating important crowd simulation applications.

Because we can see crowds anywhere at anytime, crowd simulation is vital in many virtual environment applications that concern education, training, and entertainment. For example, a movie might use a special effect to create a battle scene where many characters are fighting each other. A virtual environment, such as a city for training fire fighters, will look more convincing if there exists a large crowd populating the environment, just as in real cities. In architecture, crowd is also an important factor for designing a building or for urban planning. Computer games, war-strategy games in particular, require many characters that interact with each other and that exhibit high-level behaviors, such as grouping or separating. In light of these specific applications and the impressive demand for graphics content, there is every reason to believe that demand for simulated crowds will grow in the future.

## 1.3 Challenges

Although crowd simulation can be applied to a wide range of applications, there are several challenges that make high-quality crowd animation a difficult task to carry out. First, crowd behaviors are complex because so many factors affect them all the time. Second, it is not easy to maintain the real-time performance of a large simulated crowd. Third, crowd behavior must be visually and semantically plausible during simulation, even when we simulate a large number of characters. Fourth, the collision between characters should be checked fast and efficiently. Finally, we would also like to control the actions of the crowd, but we don't want to control every character individually. Coming up with control mechanisms that reduce the number of control parameters for simulating crowds is hard.

## 1.4 Complexity

Crowd simulation heavily depends on the complexity of the environment and the density of the crowd. The density of a crowd and the complexity of an environment are defined as follows:

- **Density of a crowd:** The density of a crowd in an environment can be computed as the ratio of the summation of each character's size against the total walkable area of the environment. The density of a crowd is proportional to the number of characters in a given fixed environment.
- **Environmental complexity:** Environmental complexity depends on how many local behaviors are needed in the environment. A local behavior is defined as a behavior that is needed only for specific areas of the environment or a behavior that only works for some specific individuals of the crowd. Therefore, the number of local behaviors depends on how many stuff there is in the environment and also depend on how many social relationships the

crowds have. For example, consider a street environment; if there are a lot of objects in the environment, such as crosswalks, newspaper venders, vending machines, stores, benches, buildings and parks, then the number of local behaviors also increases because people show different behaviors when they meet different objects. Therefore, environmental complexity is proportional to the number of objects in the environment. For testing the performance of a crowd simulation, we define the *physical* complexity of an environment more specifically as a ratio of the area of all obstacles against the total area of the environment. Thus, as the total area of all of the obstacles in an environment increases, the physical complexity of the environment also increases.

## 1.5 Research Goal

In this research, we define “*high-quality*” crowd simulation as simulation that meets four specific demands. These demands are *scalability*, *efficiency*, *convincingness*, and *controllability*. The hypothesis of this thesis is that satisfying these four demands can result in high-quality crowd simulation. **The research goal of this thesis is to propose a method that meets these four demands at the same time.**

It is critical that this thesis should define these demands with greater precision:

- **Scalability:** Scalability is the major demand of this research. It is the ability of a crowd simulation to continue to function well when the environment becomes more and more complicated.

The satisfaction of this demand makes it possible that an increasingly complex environment, where many complex behaviors are needed, can be created without a corresponding increase

in the complexity of the characters. This demand requires that the information that a character stores in his or her own data storage should gradually increase or decrease depending on the complexity of the current situation.

Because we do not want to store all information in the character architecture, this demand makes characters to be as generic as possible. This allows the crowd to adapt themselves to various kinds of environments automatically without requiring each character's architecture to undergo changes. For instance, introducing a generic crowd from a shopping mall environment into a theater environment should not necessitate changes in a character's architecture. The more specific demands require the following:

- **Scalable character memory:** The amount of memory a character uses does not increase proportionally to the complexity of an environment.
- **Scalable authoring:** A simulation can be authored in a scalable way if (1) the global simulation divides into a small set of local simulations, (2) the design of each local simulation can be done using a simple user-interface, and (3) simple copy-and-paste techniques allow each local simulation to be reused for other purposes.
- **Efficiency:** Efficiency is defined by how fast an algorithm can be performed given a fixed amount of computing power. In particular, we require a crowd simulation that satisfies the following three efficiency demands.
  - **Fast motion synthesis:** Given initial and target constraints, such as a pose, position, orientation and time duration, an algorithm must synthesize the motions that meet those constraints in real time or without any noticeable time delay.
  - **Fast collision detection:** The detection and the avoidance of both inter-character collision and collision between characters and virtual objects should be efficient. In particular,



the inter-character collision testing speed should be faster than simple frame-by-frame method.

- **Fast simulation performance:** Overall simulation performance should increase linearly as we increase the number of characters.

The speed of a crowd simulation algorithm is affected by the density of the crowd and the physical complexity of the environment. For validating the efficiency demands, experiments should explicitly specify the density of the crowd and the physical complexity of the environment.

- **Convincingness:** Convincingness is defined as the ability to cause viewers to believe specific aspects of a crowd. Our demand for convincingness means that a crowd should behave convincingly during simulation in the following two specific areas:

- **Visually convincing behavior:** Visually convincing behavior refers to the realistic motions of crowds. In particular, visually convincing behavior should guarantee that there are no significant visual artifacts, such as discontinuities, foot-skating, inter-character collisions, or collisions between characters and environmental obstacles.

- **Semantically convincing behavior:** Semantically convincing behavior is behavior that strikes one as reasonable for any given time. Reasonable behavior depends on the animator's intention for crowd control. That is, if an animator wants to see a particular crowd behavior at a particular place, the crowd should exhibit that behavior when they are at that place. For example, if an animator wants to see characters crossing a street when on a crosswalk, they should exhibit such behavior when they are on the crosswalk.

- **Controllability:** Controllability is the animator’s ability to specify crowd behaviors by setting constraints. The constraints in our approach include high-level constraints, called scenarios, for controlling the overall crowd without considering specific individuals, and low-level constraints, such as position, orientation, time duration and pose, for specific individuals.

A high-level scenario constraint specifies the order of a crowd’s behaviors. For example, a theater-themed scenario should specify that people must buy a ticket at a ticket booth, enter the lobby, hang around in the lobby before the movie begins, and finally enter the theater and watch the movie, in that order. One requirement for this high-level scenario constraint is that the scenario should account for the variety in a crowd, meaning that not all people should act in the same way.

Low-level character constraints require a fast, accurate motion-synthesizing technique, meaning synthesized motions satisfy the constraints exactly and the synthesizing time is fast.

Controllability is closely related to the user-interface problem because the former needs both authoring tools to specify behaviors and special data structures to set constraints automatically for each individual.

## 1.6 Existing Methods

In recent years, several researchers have proposed different methods related to crowd simulation. These approaches can be categorized into *crowd modelling*, which proposes computational models for crowds [FTT99, WCP\*, AC01, Rey87, FMS00, MT01, HM95, HFV00, BC97, GLM99, BMdB03], *smart environments*, where the integration of environment-related information into a character’s knowledge structure gave rise to proper character behaviors [FBT99, TLCC01,

FMS98], and *collision detection*, which made the fast identification of collisions between characters possible [KKS03, RKL\*04].

Most of these approaches were application specific, focusing on different aspects of the crowd. As a result, they employ different techniques ranging from those that treat individuals as a flow to those that represent each individual as being controlled by rules based on either physical laws or behavioral models [UT01].

Application-specific approaches may meet some particular demands but fail to satisfy our four demands simultaneously. This failure stems from the following reasons:

- **Lack of generality:** Most previous approaches have focused on specific environments. Therefore, the simulated crowds were not applicable to a general environment. For example, Helbing et al.'s crowd-modelling technique was applicable only to a panic or emergency situation [HM95, HFV00]. To be useful in other environments, the crowds would have to go through a complicated re-design process.
- **Lack of behaviors:** The previous approaches ignored not only the high degree of complicatedness characterizing crowd behaviors but also how the environment affects crowd behaviors. As a result, only a small set of simple behaviors can be simulated in these approaches [Rey87, MT01].
- **Lack of convincingness:** Previous approaches did not account for the details of each character's motions. Even though some researchers have adopted 3D characters and animation, the resulting motion quality is insufficiently convincing because the motions are created from simple motion-synthesis techniques, for instance, key-framed animation, a small number of motion captured examples or image-based character animation; all of these techniques have limitations in convincingness [MT01, UCT04, TLC02a, SMH05].

## 1.7 What Makes a Crowd as a Crowd?

In developing an efficient method for crowd simulation, the first step is to observe real crowds and then identify their behavioral features so that simulations can exploit these features. The most typical features are the following:

- **Anonymity:** When we see a big crowd, it is hard to distinguish one particular individual from the others. That is, aggregate behavior of the crowd is more important than what each individual does is. Owing to this anonymity, viewers are attentive to the overall statistics of a crowd: the overall direction in which crowds are moving or the number of people waiting for a bus. Hence, the motions of an individual matter only in their short-term contribution to a crowd's behavior.
- **Locational and social-relation effects on crowd behaviors (Locality):** Crowd behaviors are complicated in that some behaviors are global and unchanging whereas other behaviors are local and temporal. For instance, the *walking-straight* behavior is a global behavior because it can be shown at any time in crowds. But the *walking-together* behavior is a local behavior because it is needed only when an individual is walking with other individuals. Similarly, *walking-by-checking-the-traffic-signal* is necessary only when individuals want to cross the street. Most of the time, crowd behaviors are mixed with both global behaviors and local behaviors. The problem of which behaviors subsume other behaviors is dependent on an individual's location and social status relative to other individuals. For example, when the crowd is located at a crosswalk, the *walking-by-checking-the-traffic-signal* behavior is more prominent than the *walking-straight* behavior. From this observation, we determined that the crowd needs only a few local behaviors at any given time and that its behaviors depend on an individual's location and social status relative to other individuals.

- **Variety:** Various events occur in an environment; moreover, people have different responses to the same event. For example, in the event that people meet a crosswalk, some people might use the crosswalk while the others might avoid the crosswalk. When people visit a museum, some of them show an interest in a painting by standing in front of it and by looking at it for a long time, but still others just pass by the same painting without even looking at it.

## 1.8 Thesis Statement

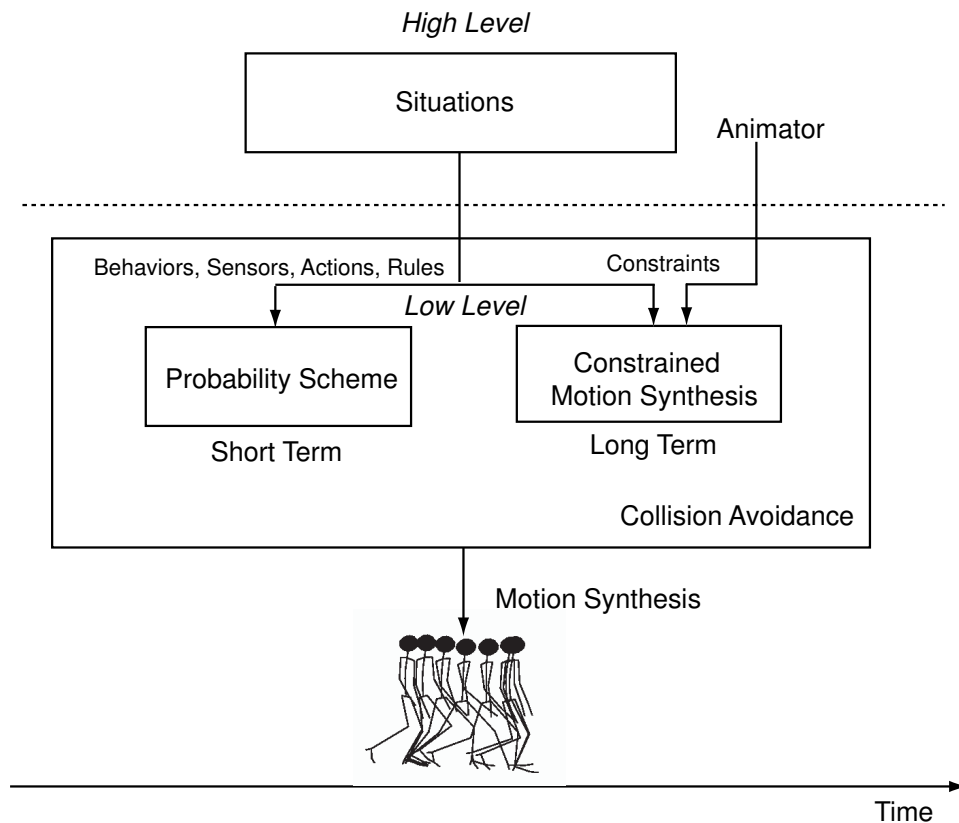
This research presents a novel two-level crowd simulation approach that meets four demands at the same time. By using this method, animators can create convincing yet controllable behaviors for characters without requiring character complexity to increase drastically when the complexity of the environment does. **The thesis of this dissertation is that convincing and controllable crowd behavior can be simulated through a scalable and efficient two-level crowd simulation approach.**

The two-level simulation method consists of a high-level and a low-level part. The basic intuition underlying the two-level crowd simulation is, itself, bi-fold; the high-level part provides crowds with all the required information about the current status of their environment (including information about other characters), and then the low-level part synthesizes motions for each individual by using that information.

The overview of the two-level simulation approach is described in Figure 1.1. The next section briefly describes the two-level crowd simulation framework.

### 1.8.1 The High-Level: Situation-Based Simulation

At the high level, we adopt a distributed control mechanism, called a *situation*, which gives crowds specific details about how to react at any given moment. The main inspiration for the situation-based approach is the anonymity feature of crowds (Section 1.7). When we look at a



**Figure 1.1:** At the high-level, situations provide all the necessary information to characters for simulating local behaviors. Then, at the low-level, depending on the applications, the crowd simulation generates motions for individuals by using one of these two approaches either the probability scheme, which computes probabilities over all available actions and composes them to select one next action, or the constrained motion synthesis, which synthesizes motions that meet constraints.

crowd, we care only about *what* is happening with them, not about *who* is doing the what. Hence, the behavior of the crowd should be driven by situations – what is happening – and not by individuals. To specify what is happening in a particular situation, we use the locality feature of crowds (Section 1.7); crowd behaviors depend on location and social relationship. Specifically, we ensure that the situation contains information such as (1) local behavior functions that implement the conceptual notion of behaviors by computing probabilities of all actions, (2) local actions, which are motion clips that are only needed for the situation, (3) sensors for catching events happening in the

environment, (4)rules for determining any subsumption between behaviors and (5)constraints like particular target positions, orientations, time duration or poses. When a character enters a situation, to make the character exhibit proper behaviors, the situation extends the character by adding relevant information. Once he or she exits the situation, the situation removes that information from the character. In this manner, characters carry information only about situations they are currently under. This feature enables us to simplify the character architecture and to render it adoptable in a wide variety of environments.

The environment can include cases where several situations influence the crowd simultaneously. For this case, it is possible to compose multiple situations that emulate a complex situation.

To specify a particular situation in an environment, we adopt a simple *painting interface*. By drawing the situation on the environment directly, animators can effectively specify both regions on which a situation takes effect and a particular group of characters that have a social relationship.

In addition, animators can connect situations as a graph structure to represent an order of situations that a crowd follows. In this graph, a node represents a situation and a directed edge represents an order between two situations. By traversing the graph, we can specify a scenario for controlling crowd flow. For example, we can connect two situations with an edge, and when a character enters one situation, then we can get a random position of the other situation and set the position as a target position for the character. By using a specific behavior function which is provided from the situation, we can make the character move to the target position.

## 1.8.2 The Low-Level: Motion Synthesis

Given the information provided from high-level situations, each character synthesizes motions at the low-level. In this motion-synthesizing procedure, the low-level part adopts the graph-based motion-synthesizing algorithm [GSKJ03] where each node of a graph is a common pose and edges of the graph represent short motion clips that connect two nodes. Basically, the graph-based motion

synthesis creates motions by selecting one edge from the current node and playing a motion clip that corresponds to the selected edge. Therefore, the main job of the low-level part is to select the current node of a motion graph for a character and to make a selection for the next action (edge). For action selection, we use two different mechanisms called the *probability scheme* and the *constrained motion synthesis*. Depending on the application, our algorithm chooses between the probability scheme and the constrained motion synthesis to synthesize motions. The main ideas underlying our use of the two action-selection mechanisms are as follows:

- **Probability scheme:** Because of the anonymity feature of crowds (section 1.7), we cannot predict certain aspects of an individual's movement. Also, because of the variety feature (section 1.7), even when we know him or her well, his(her) motions are hard to be anticipated. This lack of predictive power motivates the probability scheme. Because we are unsure of an individual's movement, we model the movement probabilistically. The probability scheme computes the probabilities of all available actions by accounting for information from current situations, and selects one action through random sampling. The probability scheme can take into account of the various features of crowds (section 1.7). That is, since we select the next action through random sampling, not all people select exactly the same actions.

Identifying which action's probability is higher than the other actions' probabilities is determined by what behavior needs to be shown for the character at that time. In section 1.7, we note that, with regard to crowds, individuals' short-term behaviors are more evident than individuals' long-term planning. Therefore, the probability scheme is good for simulating short-term behaviors where each character does not have specific target constraints, such as a particular position or orientation. A crosswalk where many people are either waiting for a traffic signal or crossing the street exemplifies a scenario where the probability scheme is



useful. In this scenario, it does not matter which character arrives at a particular spot on the other side of the road so long as someone can actually use the crosswalk to cross the street.

- **Constrained motion synthesis:** If a character needs to arrive at an exact position or if he or she wants to sit down on a chair, the probability scheme fails to generate motions because it needs *longer planning on motions*. Such planning should find a combination of motions that satisfies the constraints. The constrained motion synthesis, which is another motion-synthesizing technique, is useful for this case. Given constraints (position, orientation, pose and time duration) provided from the current situation or from the animator, the constrained motion synthesis performs random searches on a motion graph to find a path (a series of edges on the graph) close to the solution, and then continuously adjusts the path so that the path satisfies the constraints exactly. Thus, the constrained motion synthesis is good for long-term behaviors corresponding to a series of action choices that should be figured out beforehand.

One important component of this two-level crowd simulation is collision avoidance. The collision avoidance is very important because the collision, itself, is a very noticeable artifact that affects the visual convincingness. Our two-level crowd simulation framework proposes a collision data structure called the MOBB (Motion Oriented Bounding Box) tree. The MOBB is a space-time variant of OBB (Oriented Bounding Box) trees for fast collision checking between two motion data, where the whole motion is divided hierarchically and each node of the tree contains a time interval as well as the geometrical extent of characters' motion (bounding box) during the time interval. Because the child nodes are checked only when the parent node causes a collision, we can reduce the number of collision tests significantly. The collision testing occurs during the low-level motion synthesis process. If an action causes any collision between characters, the constrained

motion synthesis does not choose the action, or the probability scheme sets a zero probability on the action.

### **1.8.3 Demand Satisfaction of Two-Level Simulation Framework**

The high-level situation-based simulation technique satisfies the scalability demand because characters are under only a few situations at the same time and their complexity depends only on how many situations they are in at that time. Once a character is not under a certain situation anymore, all information related to that situation is automatically removed from the character's structure, minimizing the information stored in character memory (scalable character memory demand). Because a situation encapsulates all required information for the control of particular local behaviors as a single unified structure, animators can easily create new situations by simply modifying existing situations and can create a big environment in a scalable way; we simply add more situations to the environment as we need them (scalable authoring). And by using the situations, we can display semantic convincingness as well; people will exhibit reasonable behaviors when they are in a particular position. A situation graph, which specifies the order of situations, improves the controllability demand because we can control crowd flow by traversing the situation graph.

When animating characters, the use of both motion-capture data and a graph-based motion-synthesis algorithm guarantees the high-quality of motion, contributing to visual convincingness. Use of the probability scheme at the low-level draws various responses from characters because the action-selection is determined in a stochastic way. This stochastic action selection improves the convincingness demand because it makes possible a situation in which not all the characters exhibit the same action-selection pattern over time. The constrained motion synthesis for long term behaviors is useful for the controllability demand; the animator can locate characters anywhere he or she wants in the environment through the algorithm.

A fast random search method on the motion graph for finding a path that meets constraints and fast collision detection through an MOBB tree improves the efficiency requirement.

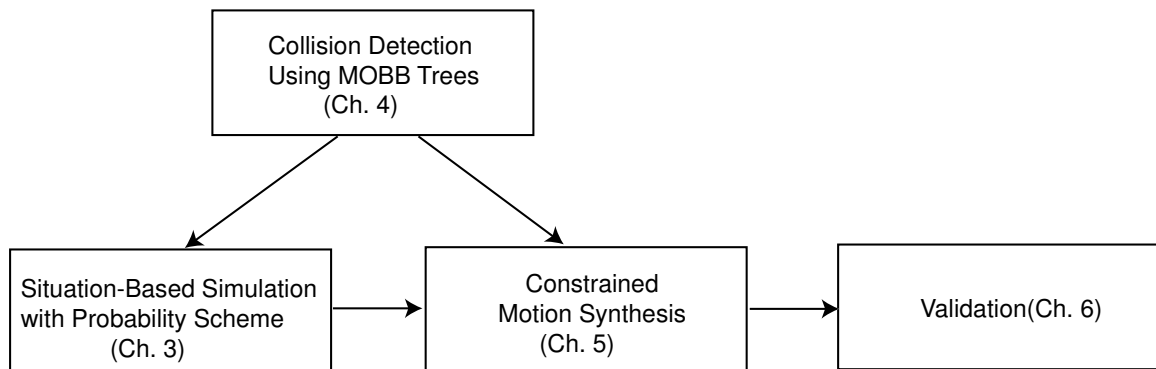
## 1.9 Contributions

In this section, we summarize the technical contributions that we make throughout this dissertation.

1. **Scalable, controllable and convincing framework of crowd simulation (Chapter 3).** We introduce a two-level crowd-simulation framework. At the high level, situations control the local behaviors of crowds by adding information to crowds when they are under these situations. At the low level, depending on the application, either the probability scheme or the constrained motion synthesis is used for the synthesizing of motions. By performing a series of experiments, we have validated that this framework has advantages in scalability, efficiency, controllability, and convincingness.
2. **Fast collision detection between motions (Chapter 4).** We propose a novel MOBB tree representation of motions for fast collision detection between two motion data.
3. **Fast and accurate goal-directed motion synthesis for crowds (Chapter 5).** We present a novel, fast, and accurate motion-synthesis algorithm. By performing a random search on the structured-motion graph and by continuously adjusting motion clips, this algorithm creates a series of motions that meet constraints fast and accurately.

## 1.10 Outline of Thesis

The remainder of this dissertation is composed of six main parts. Chapter 2 explains the related works. Chapter 3 represents the situation-based approach and the probability scheme. Chapter 4



**Figure 1.2:** The logical relationship among chapters 3-6.

describes the fast collision detection using MOBB tree. Although the collision detection using the MOBB tree is not directly related to the content of other chapters, it is an essential component in our simulation and may be read independently of the other two main parts. The constrained motion synthesis for long term behaviors given a set of constraints is considered in Chapter 5. Chapter 6 validates whether or not our method can meet the four demands we have established. Chapter 7 discusses about the limitations of our approach and future work.

Before reviewing the details of the three main parts, we will present a brief overview of the methods that will be developed in subsequent chapters (Section 1.11).

## 1.11 Overview

We summarize the following technical chapters (3-6) in this section.

### 1.11.1 Situation-Based Simulation with Probability Scheme (Chapter 3)

The situation-based approach with a probability scheme corresponds to the high-level part and to one of the two low-level motion synthesis (Figure 1.1). The goal of the situation-based

approach is to provide a scalable yet authorable way to synthesize complex behaviors while maintaining convincingness. Exploiting the feature that concerns locational and social-relation effect on crowd behaviors (Section 1.7), we allow each situation to have a particular region in the environment that this situation affects. When a character enters the region, the situations extend the character by adding information to the character. The information includes local actions, behavior functions that implement the local behaviors, virtual sensors that catch events, and rules for determining subsumption between behavior functions. The purpose of the extension is to make crowds show appropriate local behaviors and, simultaneously, to account for crowds' various features over events. The local behaviors, which should be controllable and variable, are derived from the motion selection mechanism called *probability scheme*. In this scheme, behavior functions, part of the information given by situations, compute the probabilities of all available actions considering their own criteria for judging actions. For instance, a *walking-straight* behavior function checks all available actions and returns high probabilities on the actions that move the character straight, and then returns low probabilities on the other actions. Most of the time, the crowd behaviors are affected by many different factors, therefore, in general multiple number of behavior functions are needed at any given time. To accommodate when multi-behavior functions are needed, we compose the probability distributions, which are derived from several behavior functions, in order to create the final action selection method. In this way, we combine simple behaviors and transform them into a more complex aggregate behavior. Once we complete the behavior composition, we apply random sampling to the final distribution so as to select the final action.

Because a character might be under several situations at the same time, we allow multiple situations to be combined into one by conducting a union operation. Basically, the union operation over two situations adds up all components such as actions, behavior functions and sensors.

To set a particular situation into an environment, we adopt a painting interface in which animators can specify a region or a group of people in much the same way as an artist paints a picture on a canvas.

### **1.11.2 Collision Detection Using MOBB Trees (Chapter 4)**

Animating multiple characters with motion data requires a fast collision testing among them. In our two-level crowd-simulation framework, the low-level motion-synthesis part creates motions by selecting an edge (motion) from discrete choices of a motion graph. In this process, the algorithm should ensure that the selected motion does not cause any overlap with other characters' motions (Figure 1.1). Therefore, we need only to check whether or not two given motions preserve a minimum distance between the two corresponding characters. A simple method is to check frames one by one and, therein, to compute the distance between two motions. Obviously, this approach is time consuming, especially when we need to check many characters.

We instead use a hierarchical MOBB tree data structure for fast collision testing. Each node of the MOBB tree contains a spatio-temporal simplification of motion data. In particular, the geometrical simplification is represented as a bounding box that covers entire poses at the given time interval. More specifically, the root node of the tree has simplification information over the entire motion data, and the two children nodes of the root node have the bounding box over the first half frame of the motion and the second half frame of the motion, respectively. This subdivision continues until the number of frames of the node reaches the predefined minimum number. The construction of the MOBB is completed at the preprocessing step. At run time, given two motions that need collision checking, the MOBB trees are retrieved from the two motions. Then, overlapping between the two bounding boxes which are associated with each node of the trees is tested from the top to the bottom node. Because the overlap test goes down to the children

nodes only when the parents nodes are overlapped, we can reduce the collision checking time significantly.

### 1.11.3 Constrained Motion Synthesis (Chapter 5)

Constrained motion synthesis is one of the two low-level action selection mechanisms that works for long-term behaviors (Figure 1.1). The constrained motion synthesis presents a highly efficient algorithm that synthesizes realistic goal-directed motions for a large number of characters. This technique focuses on the important problem of character navigation and introduces an algorithm that creates collision-free constraint-satisfying motions. The constraints include time duration, position, orientation, and particular body pose. Given initial and target constraints for a character, we propose a two-level motion synthesis algorithm (it is different from the two-level crowd simulation framework in section 1.8) where the high-level provides a rough path to the character by performing global path planning algorithm and the low-level motion synthesis then generates motions that meet the constraints exactly by using the high-level's rough path. Basically the low-level motion synthesis starts by creating two initial paths called the forward and the backward path that satisfy only the initial and the target constraint respectively. For measuring the distance between them, we come up with a cost function,  $C$ , which indicates the minimum distance between the forward path and the backward path. If the cost is lower than the user threshold value  $\epsilon$ , then the algorithm merges the two paths into one path by connecting the two paths with distributing the distance evenly to the both paths. If the cost is bigger than the user threshold value  $\epsilon$ , then the algorithm randomly perturbs the initial two paths until the cost function returns a value below the threshold value.

Because we allow the motion clips from the motion graph to be continuously adjusted to alter a character's position, orientation, and speed, we can satisfy the constraints exactly. Also, this allowance yields a shorter search time because a raw sequence of clips needs only sufficiently

reduced constraint deviation, rather than minimized one. Our algorithm also works in highly complicated environments where there are a lot of obstacles. The main reason is that the rough path obtained from global path planning at the high level provides a collision free path which is used for initial paths at the low level.



## Chapter 2

### Related Work

This chapter reviews researches related to crowd simulation. We first present a general overview of human character animation techniques in section 2.1. In particular, we illustrate the graph-based motion synthesis algorithm more in detail in section 2.1.1 because we used it as our low-level motion synthesis. And then, section 2.2 lists up a number of crowd modelling researches. Section 2.3 reviews the crowd rendering techniques. Section 2.4 overviews the general collision detection algorithms. Finally, section 2.5 recaps the existing commercial softwares for crowd simulation.

#### 2.1 Animating Human Characters

Because we define the crowd simulation as a collection of individuals' behaviors, we need a way to animate characters. In particular, we need an algorithm for animating human-like characters. In this section, we overview the existing human character animation techniques and figure out what problems they might have when we apply those techniques in crowd simulation. From this section, we assume that a character means a human-like character.

The motion of an individual character can be animated in several different ways. The classical key-frame method, which manually specifies a sequence of character poses over time, has been the most popular way up to now. However, it requires intensive labors, artistic sense, and much time for creating animation. This becomes worse in crowd simulation because a lot of characters need

to be animated. An alternative to key-framing method is the procedural animation where hand-craft algorithms produce particular motions. This allows an artist to create entire motions at once, rather than one pose at a time, and motions can be altered on the fly to allow for online, interactive character animation [Kov04]. For example, Perlin [Per95] and Perlin and Goldberg [PG96] have demonstrated that a variety of motions can be generated with simple and computationally efficient algorithms. However, these algorithms are quite difficult to design and a totally new algorithm should be developed for different kinds of motions. In particular, for a walking motion, a few procedural algorithms have been proposed. Bruderlin et. al. proposed a hybrid method in which goal-directed high level control and dynamics were incorporated to create human walking cycle motions [BC89]. Boulic et. al. presented a human walking dynamic model which is built from bio-mechanical experimental data [BMTT90, BUT04]. These methods could create motions that exactly follow arbitrary trajectories effectively, eliminating the need for additional search mechanism. However, the procedural techniques cannot be applied to crowd simulation problem directly because those techniques are quite difficult to reproduce the subtle personality of human motions.

The physically based approach is one of the other possibilities. In physical simulation, Newton's laws are used to solve a system of ordinary differential equations that can be integrated to obtain joint trajectories given the mass distribution for each body part and the torques generated by each joint [Kov04]. However, finding joint torques that create a particular motion trajectory is a difficult task [Kov04]. Hodgins et. al. were able to create jogging and bicycling motions through proportional-derivative servos on a finite state machine [HWBO95]. By using the similar approach, they also produced the gymnastic motions [WH95, WH00]. Faloutsos et. al. proposed physics-based composable joint controllers for characters, and were able to create the balanced and protective stepping motions [FvdPT01]. One common problem of these approaches is that

their methods could not produce the wide range of motions; only particular motions could be created. This makes it impossible to apply to crowd simulation because crowds need various kinds of motions.

Another way is to record the motions of a live performer and reuse them in animation, which is called *motion capture* [Men00, Stu94]. Although the motion capture approach guarantees the realistic human motion, it lacks control over character's actions because it is no more than fixed signal data. For example, if we need to make a character move around an environment, we need many motions with different speed, jumping motion over obstacles and motions with different turning angles. Capturing all required motions costs too much. Also, it is infeasible to adjust an animation by directly editing motion data [Kov04]. Thus, many researchers have proposed techniques called *motion editing* in which the original motions are adjusted automatically to meet the animator's constraints [BW95, WP95, Gle98, LS99, CK00]. The goal of motion editing is to provide control over motion data while maintaining the fidelity of original motion data. To achieve that goal, they generally build a *conceptual model* to manipulate the motion data and synthesize a new motion from it.

For example, Kovar et. al. [KGP02], Lee et. al. [LCR\*02a], and Arikan et. al. [AF02a] proposed a *motion graph* model respectively. The motion graph was built from the original motion corpus by identifying similar poses of motions and connecting them with an edge which represented a transition. Once the motion graph has been built, a predefined searching algorithm traversed the graph to generate motions. Basically, the graph-based motion synthesis creates a new motion by re-ordering the frames of motion data. More details about the graph-based motion synthesis will be followed in section 2.1.1.

Rather than reordering the motion data, several researchers have proposed a *generative model*, which is learned from original motions and generates a new motion from the model. For example, Li et. al. presented a statistical LDS (Linear Dynamic System) model [LWS02] and Brand et. al.

used HMM (Hidden Markov Model) [BH00] as the generative model. Those methods overcome the problem of reordering based motion graph in that they can edit at the frame level. However, the generative model techniques are not necessarily good for crowd simulation because the frame-level synthesis is slow for a large crowd.

Another alternative to motion generation model is the motion blending [GR96, PSS02, PLS03], which (as with procedural synthesis) allows continuous control over character trajectories. However, blending-based models are more restricted since individual clips must be blendable, and need additional computational overhead because each generated character pose must be formed by combining example poses [SKG05]. Therefore, we cannot use those techniques for crowd simulation because high speed motion synthesis is highly required. Hybrid graph/blending methods [RCB98, KPS03, PSS04] share similar concerns.

### 2.1.1 Graph-Based Motion Synthesis

In crowd simulation, fast and realistic motion synthesis is required. Our two-level crowd simulation framework adopts the graph-based motion synthesis algorithm for the low-level motion synthesis. In this section, we overview the general graph-based motion synthesis algorithms.

Graph-based motion synthesis has been used widely in game industry as the name of *move tree*[MBC01] for more than decades [Kov04]. Historically, these motion graphs have been made in a manual way in the sense that a user explicitly specifies which motion clips should be connected [RCB98, Kov04]. In physically based simulation, joint controllers were also linked together as a graph structure manually [FvdPT01]. In videos, by grouping similar frames, Schödl et. al. generated user-controllable new video clips from the original video clips [SSSE00, SE02].

To overcome this manual work, several researchers have proposed the automatically constructed graph structure where nodes were similar poses and edges represent transitions between them. Kovar et. al. [KGP02] proposed a pure automatic way for constructing a graph structure

from the motion capture data and a searching method on the graph. For automatic construction of the motion graph, the algorithm identified the similar poses from the original motion data and then synthesized transition motions at these points [Kov04]. Similarity was determined through a novel point-based distance metric [Kov04]. Once the graph was built, rather than having a user directly control the graph, they introduced a search algorithm for finding a sequence of edges that meet a user-supplied constraints [Kov04]. To demonstrate this approach, they applied this framework to the problem of creating motions that traverse arbitrary paths [Kov04]. As an extension for highly interactive application such as a game, they extended the generic motion graph to the structured motion graph, called *snap-together-motion*, where connection between nodes could be predetermined by user and similar motions were gathered around a small number of hub nodes. At running time, the motions were just concatenated each other to synthesize a long series of motions without any further processing [GSKJ03]. We adopt the structured motion graph for our low-level motion synthesis; primary reason is that its applicability in interactive system. The Snap-Together Motion (STM) preprocesses a corpus of motion capture examples into a set of short clips that can be concatenated to make continuous streams of motion. The result process is a simple graph structure that facilitates efficient planning of character motions [SKG05]. A user-guided process selects common character poses and the system automatically synthesizes multi-way transitions that connect through these poses. In this manner well-connected graphs can be constructed to suit a crowd simulation, allowing for fast synthesizing motion without the effort of manually specifying all transitions at run time [GSKJ03].

Lee et. al. [LCR\*02a] and Arikan and Forsyth [AF02a] introduced the similar original motion graph algorithm respectively. Both algorithms also automatically identified the transition locations based upon a distance metric and then used a search algorithm on the resulting graph to find motions that satisfy the user-defined constraints. Differences among these three approaches are 1)

they used a different distance metric for finding similar poses and 2) they targeted on different applications; Kovar et. al. considered constraints on the path followed by characters while Arikian et. al. focused on more generic constraints such as position, orientation and poses [Kov04]. Lee et. al. presented an interface where a user could select which motion should happen at that place [Kov04]. One significant problem of these approaches is that because these methods create new motions by strictly attaching existing clips, constraints on the generated motions are not exactly satisfied, and finding a solution that minimizes deviation from the constraints tends to be a time consuming job [SKG05]. Our constrained motion synthesis, on the other hand, takes account the motion adjustment in searching process, which speeds up the searching time and makes motions satisfy the constraint exactly [SKG05]. Similar motion adjustment on the regular grid space was introduced by Reitsma and Pollard [RP04], but, their goal was to come up with an evaluation function for a particular motion graph so that they could test whether the given motion graph had the ability to perform navigation task (e.g., directing a character to a particular position and orientation).

As an extension, Lai et. al. [LCF05] proposed a group motion graph technique. In this technique, they first constructed a graph structure from seamless group animations by identifying similar configurations (nodes) and connecting them as edges (short animation). By traversing the graph with satisfying constraints, they were able to synthesize a new group motion such as path-following and region constrained of a flock. Their approach is good for animation for flock, but does not necessarily suit for crowd simulation because the crowd need complex behaviors such as "go crossing the street only when the traffic sign says walk".

## 2.2 Crowd Modelling

In this section, we overview current crowd modelling researches. In particular, we focus on whether those approaches satisfy the four demands we set and how different they are from our high-level situation-based simulation.

For generating human-like behaviors, many researchers have adopted the artificial intelligent (AI) techniques, in particular, intelligent character techniques in recent years [FTT99, WCP\*, AC01]. Basically, their goal was to provide autonomy to each character so that he or she could decide his or her behaviors automatically. In order to do that, they proposed the character architecture that is comprised of a knowledge representation, algorithms that learn new knowledge, and modules that plan actions based on the knowledge. However, because the level of knowledge for a character depends on the level of autonomy, the deeper knowledge a character needs, the more complicated character architecture is required, which has a problem in scalability.

Rule-based schemes, such as Reynolds' *boids* models [Rey87], are fast enough to be used for large numbers of characters, but they are not controllable. Controlling crowds for complex environments is extremely hard. For example, controlling a flock to follow a specific trajectory is impossible because their simple rules do not consider the global shape of flocks. A simple solution is to add more rules to the flock. But, determining weight values among rules that control which rule has bigger influence than other rules is another hard problem. Similarly, Shao et. al. [ST05] proposed several reactive behaviors based on specific rules for controlling pedestrians. The reactive behaviors include the safety-turning, crowd direction control and collision avoidance behavior. These behaviors are processed sequentially with specific order. But, the rules for behaviors are hard to generalize to be used in other environments other than pedestrian.

Hierarchical schemes have been proposed to address scalability [FMS00, MT01]. In particular, Musse et. al. endow crowds with different levels of autonomy for hierarchical crowd behaviors [MT01]. Depending on the level of autonomy, they employ different behavior generation techniques ranging from script-based behaviors to innate or pure reactive behaviors. However, all behaviors that they can simulate were relatively simple such as splitting, wandering, repulsing and attracting behaviors.

At the other extreme, physics-inspired approaches, such as social force models [HM95, HFV00] or particle systems [BC97, GLM99, BMdB03], can create realistic crowd flow but are only applicable to situations such as emergency evacuations in which crowd behaviors are limited and interactions with the environment are minimal. Main reason for these limitations is that complicated human motions cannot be explained with a simple physical law. Therefore, although the algorithm might mimic the realistic flow of crowd when a particular event happens, convincingly detailed motions of individuals cannot be simulated.

In order to reduce the complexity of controlling crowds yet retain detailed behaviors, several systems [FBT99, TLCC01, FMS98] have attached information to the environment to guide the characters within it. Simple examples include driving cars or pedestrian simulators that embed lane or pathway information in the model. Our situation-based approach also embeds information, such as planned paths, into the environment, but, moreover, we include the *behaviors* and even *sensors* to interpret that information.

The most commercially successful system of this type is the computer game, *The Sims* [FW01], in which objects advertise services, such as “satisfy hunger”, and define the procedure that runs when the character responds to the service. *The Sims* highlights the authoring advantage of a rich environment. Part of the game’s appeal is the ability to add new objects easily and have characters respond to them. *The Sims* adds behaviors by enforcing a specific, linear *plan* to the interacting characters. If the desired object-specific behavior is not amenable to a simple plan, the approach



breaks down. For instance, a “mingle with a crowd” behavior could not be added even though such interaction must be part of the character’s innate behavior which makes the character complicated. In contrast, our situation-based approach adds “behaviors” that interact on equal terms with the existing behaviors and has all the power of rule-based state machines, which allows for much simpler underlying characters and more complex extension behaviors.

Similar to *The Sims*, Kallmann and Thalmann [KT98] describe *smart objects*; objects that provide a plan for their use. The character, upon approaching an object, is told to execute a specific sequence of steps. For instance, an elevator informs a character to push the button, wait, and then enter when the door opens. This approach is similar to our situation-based approach in that the characters are provided all necessary information from the environment directly. However, in our approach, the situation is a more general concept that includes non-physical effects such as friendship and membership on a group among characters. Moreover, rather than indicating a specific behavior to a character at a particular time, our situations propose a composable behavior that can be combined with others. One result of this is that characters in our system do not always respond to the same object in the same way, just as real people behave.

Robotics algorithms have been applied to animate multiple characters. Bayazit et. al. [BLA02] use a global road map to set collision-free paths for multiple characters, and similar methods have been introduced [Feu00]. Kamphuis et. al. [KO04] proposed a coherent path planning algorithm for multiple character. Basically, in their algorithm, they first do a path planning for a single character, and then they extend the path as a corridor for making characters move together.

Tsi-Yen Li et. al. proposed the Leader-Follower model [LJC01], the main goal of which is to generate collision-free paths for characters. Since their focus is on path planning for multiple characters, complex behaviors such as “finding an empty seat and sitting down, then watching a movie” cannot be simulated. Because the situations contain all information for complicated behaviors and that information is plugged into crowds when they need it, our approach is able

to simulate complicated behaviors. In our situation-based approach with a probability scheme, collision avoidance is achieved through a behavior function that penalizes states which may cause collision by giving low probability to them.

## 2.3 Crowd Rendering

Although our research does not focus on the visual rendering of crowds, the rendering is one of the important parts because it is the most time consuming job in crowd simulation [AW04]. This section overviews the current character rendering techniques.

The rendering of an articulated body skeleton is usually described as *skinning*. Basically, the skinning can be done in two ways, which are geometry-based rendering and image-based rendering. In the geometry-based rendering, animators typically manipulate an underlying hierarchical skeleton. Then, the character mesh geometry must be attached to the underlying skeleton so that as the skeleton deforms, the mesh also deforms appropriately. The underlying mesh geometry is computed by transforming each vertex by a weighted combination of the joints' local coordinate frames [MG03]. Although the geometry-based rendering is able to show the character poses in detail, it is quite computationally expensive for a large number of crowds. As an extension, James et. al. [JT05] proposed a general setting of skinning deformable mesh animations, where skinning was approximated from the mesh (not from skeleton) through a clustering of orientations of mesh triangles and robust least squares methods to determine bone transformations. Incorporating current programmable graphics hardware, they could generate skinning animation of more than 1,500 characters.

Tecchia et. al. [TC00, TLC02b], on the other hand, describe a real-time, image-based alternative to motion-captured data for crowds. In this technique, the character's imposters are computed at preprocessing and rendered at the run time depending on the camera orientation and distance

from the characters. Because it is a purely image-based rendering, it can render a lot of characters at the same time. However, the realism of image-based approaches is limited by the amount of imagery that must be stored in order to handle arbitrary, close-up viewing conditions. Ulicny et. al. [UCT04] store complete meshes in OpenGL display lists and then carefully sort them, taking cache coherency into account while rendering. This results in a method which has little or no processing on the CPU because precomputed meshes are stored on the graphics card. Recently, Dobbyn et. al [DHOO05] propose a metric for switching between geometry-based and image-based rendering for large crowds. The geometry-based rendering is used for detailed animation of characters who are located close to the camera while the image-based rendering is used for rough rendering for characters located relatively far away from the camera.

## 2.4 Collision Detection

Collision detection between a lot of characters is an important problem because the collisions affect the the visual convincingness significantly. However, the dynamic feature of moving characters makes the collision detection problem hard. In this section, we overview the current collision detection algorithms between moving characters and compare them with our proposed method.

Our collision detection algorithm is a novel application of OBB trees [GLM96] to swept volume intersection in space-time. OBB trees have been applied to continuous collision detection in the past [RKC02, RKLM04], but existing techniques use bounding boxes fitted to static geometry and account for motion in the intersection test. This restricts their application to simple parameterized motions (such as linear translation [Ebe01]) and makes them unsuitable for motion capture data.

The majority of literature in the graphics community is targeted at simulation algorithms where the future motion of the body is either ballistic or bounded but unknown (see Mirtich [Mir96] for

rigid body examples). Kim et al. [KKS03] describe a crowd-specific solution that uses parabolic horns as space-time bounds (spheres of changing radii swept along parabolic paths), while kinetic data structures [BEG\*04] assume motions along rational curves. Our problem differs in that we know the precise trajectory but in a sampled form, and we have a generate-and-test strategy, rather than the continuous search for the next collision.

The robotics community has dealt with similar problems in the guide of intersection-free robot motion planning. Several solutions have been proposed based on 4D space-time swept volumes (see Abdel-Malek et. al. [AMBJ02] for a survey). Recent advances include work aimed at complex models [KVLM03], but the most similar approach to our MOBB tree is due to Foisy and Hayward [FH93]. They use a hierarchy of convex swept volumes, each volume specified by a set of bounding planes. Our method is simpler than theirs due to the use of OBB trees and targeted specifically at motion in the ground plane.

MOBB trees detect collisions between the cylindrical bounding volumes of skeletal motion, not the skeleton itself. Redone et. al. [RKL\*04] provide a solution for close-in skeletal motion that also exploits hierarchies of volumes but assumes small time intervals between tests. Our approach can be viewed as a broad phase test that complements their work.

## 2.5 Commercial Software

To meet a lot of demands of crowd simulation in recent years, several commercial softwares have been developed as the form of plug-in type software or stand-alone system. For instance, *SoftImage/Behavior*, *AI-Implant*, *Character Studio 4.0* and *Massive* [Koe] have been used in movies or advertisement for crowd scenes. In the *SoftImage/Behaviors*, the crowd behaviors are incorporated with a complete Integrated Development Environment (IDE) that is equipped with visual state-graph editing for character and an embedded debugger for easy testing and tweaking.

The Behaviors are then created by using Piccolo, a high-performance, compiled Java Script-style language[Sof]. The *AI-Implant* provides the intelligent navigation of crowds in which the entire environment is composed of a lot of cells and dynamic searching is performed on cells for finding path. The software also has visual authoring and debugging tools for brain creation of individual [AI-]. The *Massive* software is a character-based crowd simulation system. In this system, a character has an artificial vision, hearing and touch-sensor processes that allow him (or her) to respond to the environment naturally [mas]. To get the variety of crowd reaction over environment, the software used a fuzzy logic algorithm.

In general, all current crowd softwares need a complicated character architecture because each individual needs an automatic way to decide his or her behaviors. Our approach is different from their approaches in that we try to transfer all information to environment as much as possible. This approach allows us to make the character architecture simple and make crowd adopted in various environment easily.

## Chapter 3

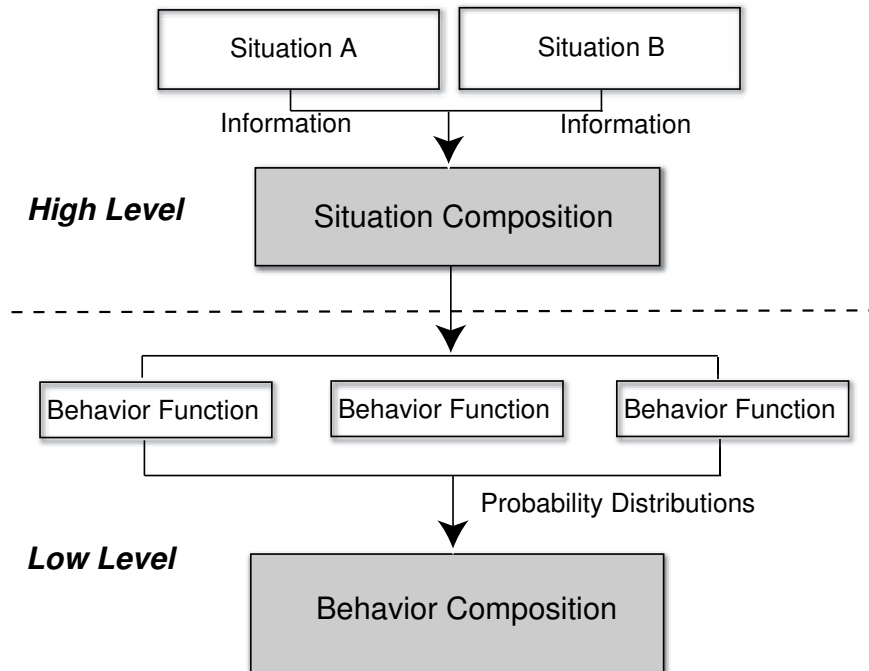
# Situation-Based Simulation with the Probability Scheme

In this chapter, we present the situation-based simulation with probability scheme. The situation-based simulation corresponds to the high-level part of two-level crowd simulation framework (Figure 1.1) and the probability scheme is one of the two low-level motion syntheses that is responsible for short term aggregate behaviors.

### 3.1 Introduction

The situation-based simulation with probability scheme adopts a *situation* that is a distributed control mechanism that gives each character in a crowd specific details about how to react at any given moment based on his or her local environment at the high level. And then, at the low level, we use a probability scheme which computes probabilities of all actions to be the next action and then samples to move the simulation forward. For maintaining the visual convincingness, we use the *Snap-Together-Motion* [GSKJ02], which guarantees that there is no artifact as motions are concatenated together over time. The overall structure of this approach is shown in Figure 3.1.

The main inspiration for the situation-based approach with probability scheme is the anonymity and the locality feature offered by crowds. When we look at a crowd, we care only about *what* is happening, not *who* is doing it. This has two implications. First, the actions of the crowd should



**Figure 3.1:** An overview of our situation based simulation architecture. At the high level, single or multiple situations are composed to provide information to characters. Then, at the low level, each behavior function which is obtained from situations, computes the probabilities of all actions and those probabilities are composed together to select one final action.

be driven by situations – what is happening – and not by individuals. The “what is happening” part is usually decided by the location of crowd. This motivates our situation-based strategy. Secondly, viewers of a crowd cannot individually identify and track characters. Rather, viewers are attentive to overall statistics of the crowd: the direction it’s moving, the apparent agitation of its members, the number of people waiting for the bus, etc. Hence, the actions of an individual matter only in his (her) short-term contribution to the crowd’s behavior, and not in their long-term planning. This motivates our probabilistic approach to simulation.

The probability scheme exploits the observation that when individuals are anonymous, their specific actions may appear somewhat random. Consider a man crossing a busy city street at a particular instant. There are many actions he may choose: he might continue walking across the

street; he might turn around and walk back the way he came; he might glance to the side to check if a car is disobeying the traffic signals. If we knew the individual, his or her choice might be clear: we might know that a particular person is prone to remembering that he or she forgot something in his or her office while crossing the street. For an individual we know little about (a member of a crowd), we cannot say for certain. We thus model the action choice probabilistically; a person crossing the street is more likely to keep crossing, but there is some chance that he or she may turn around and walk back. Therefore, the state of character is decided by what action the character takes for each time step.

When the next state is required, our probabilistic action selection method considers the choices that the character has for his or her next state, creates a probability distribution as to which choice is likely to be taken, then samples from this distribution to determine the course of action. The challenge, then, becomes determining the probability distributions such that the sequence of action choices leads not just to plausible behavior of the individual, but also the desired behavior of the entire crowd.

Complex behavior (and a correspondingly complex distribution function) is often made from a number of simpler pieces. For example, a person walking in a city will be avoiding others, trying to move towards a goal, trying to obey traffic laws, and so on. We define *behavior functions* that describe the actions for each of these simple behaviors through probability distributions. We compose distributions to create the final action selection method. In this way we combine simple functions into more complex aggregate behaviors. Behavior functions are discussed in detail in Section 3.2. A behavior function may depend on many things. For example, it may depend on the location of the individual (the middle of the street is not a good place to stop), what the character is able to “sense” (wait until the signal says “walk” before crossing), or even an aggregate controller that attempts to regulate the crowd (if there are too many people on one side of the street, it is more likely for an individual to cross so that things are better balanced).



Our situation-based approach determines which behavior functions are currently influencing a character, and hence determines the overall behavior of each character. When a character enters a situation, such as crossing the street, our system extends the character to enable appropriate behavior. Primarily, it adds temporary behavior functions to the character that are composed with his or her existing ones, allowing him or her to choose his or her actions more wisely. Situations may also add actions to the character - an individual need not know how to look both ways unless he or she is crossing the street. In fact, even the character's ability to sense his or her environment can be adapted to the situation. Only in the street crossing situation does the character need to know how to sense the status of the crossing signal. The situation also might add constraints to the characters so that they can show a series of motions that meet the constraints exactly. For example, some characters might need to sit down on a chair exactly or some of them might need to arrive at the particular positions at the same time. In this case, the situation provides the constraints such as chair positions and target positions to characters when they enter the situation. Those constraints are then input to the constrained motion synthesis (Chapter 5) to synthesize motion for the characters. When the character leaves the situation, all of the situation-specific extensions are removed.

The situation-based approach puts an emphasis on the environment design because overall crowd movement depends on the situations present in the space and where they are located. For specifying situations, we adopt a *painting interface*. It allows the user to specify a particular situation by drawing it on the environment directly. Situations can be easily composed by overlaying several in one area.

The remainder of this chapter is organized as follows. Section 3.2 explains the probability scheme. Section 3.3 describes the situation-based approach with pluggable character architecture and section 3.4 describes our painting user interfaces. Section 3.5 shows experiments we performed with our proposed architecture. Finally, Section 3.6 concludes with a brief summary and a discussion of our approach.

## 3.2 Probability Scheme

As described in Section 3.1, the behavior of a character comes from his or her choices about which action to take. At each time step of the simulation, the character has a state  $s = \{t, \mathbf{p}, \theta, a, \mathbf{s}^-\}$ , where  $t$  is the time,  $\mathbf{p}$  is a 2D position vector,  $\theta$  is an orientation,  $a$  is an action, and  $\mathbf{s}^-$  is a list of previous states. The position and the orientation indicate the spatial disposition of the character. The action is directly linked to a motion clip and determines which clip should be played for the current frame. The past states are used by the behavior mechanism to give some correlation in the character's behavior over time. Without some previous state information it is difficult to enforce behaviors like "walking in a straight line." The aim of the probabilistic state selection mechanism is to choose a next state given the current state.

The link between actions and motions means that we have a finite set of possible choices for the next action – it has to be one of the available motion clips. From the graph-based motion synthesis perspective, these available motion clips are identified by the edges on the graph which are linked from the current node. Each potential next state has an associated probability representing the chance of being selected. Our behaviors modify these probabilities on the fly, as well as the set of potential states. Note that this is conceptually similar to probabilistic finite state machines typically used for character AI [WP01], but in our approach both the state graph and the transition matrix are modified at run time. Our state transitions also represent lower level behaviors compared with traditional finite state machines.

### 3.2.1 Behavior Functions and Behavior Composition

When examining the movement of a crowd in real life, we easily see that there are many different factors that influence the behavior of a character. For example, people might change their route depending on whether or not there is a person or an object nearby. Or, if they have

a target place that they are moving toward, they usually go as straight as possible. To simulate crowd behavior realistically, we need a way to take account of these various kinds of factors and synthesize a complex behavior that reflects them. More specifically, given the possible next states, we need a way to judge these states from the point of view of different factors and then compose transition probabilities to reflect all factors.

Each influence on the character is encoded in a *behavior* and special functions called *behavior functions* perform the task of transforming a behavior into a set of transition probabilities on states. For example, the “overlap behavior function” takes care of avoiding other characters. The “Don’t turn behavior function” checks if a state transition causes too much change in orientation. The behavior functions judge the potential state transitions with their own rules independently and return the probabilities.

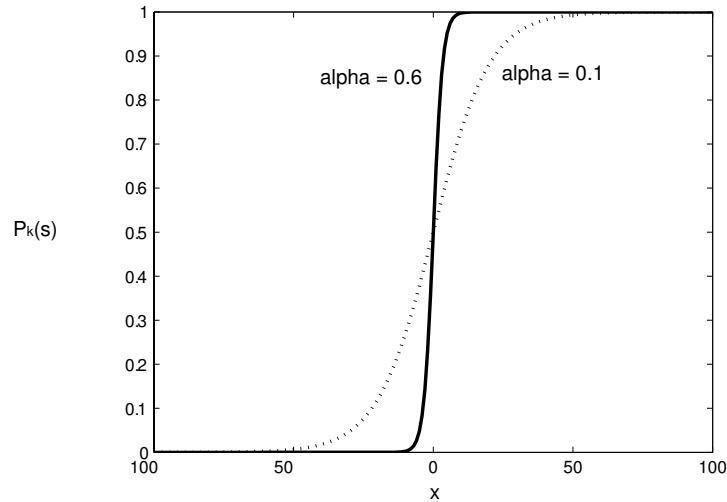
Suppose the set of possible next states is  $\mathbf{S} = \{s_1, s_2, \dots, s_m\}$ , where  $s_i$  is a particular state. Note there are  $m$  states. A behavior function,  $k$ , evaluates all states in  $S$  and calculates their probabilities. A prototype behavior function is shown below.

```

Behavior function  $k(\text{States } S[], \text{Prob } P[])$ 
{
  For state  $s$  in set  $S$  do:
     $x = \text{evaluate}(s)$ 
     $P_k(s) = \text{sigmoid}(x, \alpha)$ 
}

```

The *evaluate* routine inside a behavior function is a general function that can use any information available to it, such as the state of various features of the environment or the past state of the character or distance from a position in the environment. The evaluation function characterizes a



**Figure 3.2:** Sigmoid function

conceptual notion of “behavior” that the behavior function is trying to model. Considering this information, the *evaluate* function returns any real value. Then, the value is normalized to the range  $[0, 1)$  using a *sigmoid* function:

$$P_k(s) = \frac{1}{1 + e^{-\alpha x}} \quad (3.1)$$

Note that the collection of these probabilities for all available actions cannot be represented as a probability distribution because it may not be summed up to 1. Natural interpretation of this probability is the *weight* value for given perspective of behavior function. That is, the probability represents how likely the given action is to be next action.

We do this non-normalized probability computation to make composition of behaviors more stable. The constant  $\alpha$  determines the slope of the curve and is chosen differently for each behavior function. The values we use range from 1.0 to 10.0. A typical graph of sigmoid function is shown in Figure 3.2.

The primary reason for using behavior functions is that we are able to compose several component behaviors to synthesize more complex behaviors that embody all the influences of the component behaviors. Composition of behaviors is simply the multiplication of the probabilities the behavior functions produce (see Figure 3.3). That is,

$$P'(s) = \prod_{k=1}^n P_k(s)$$

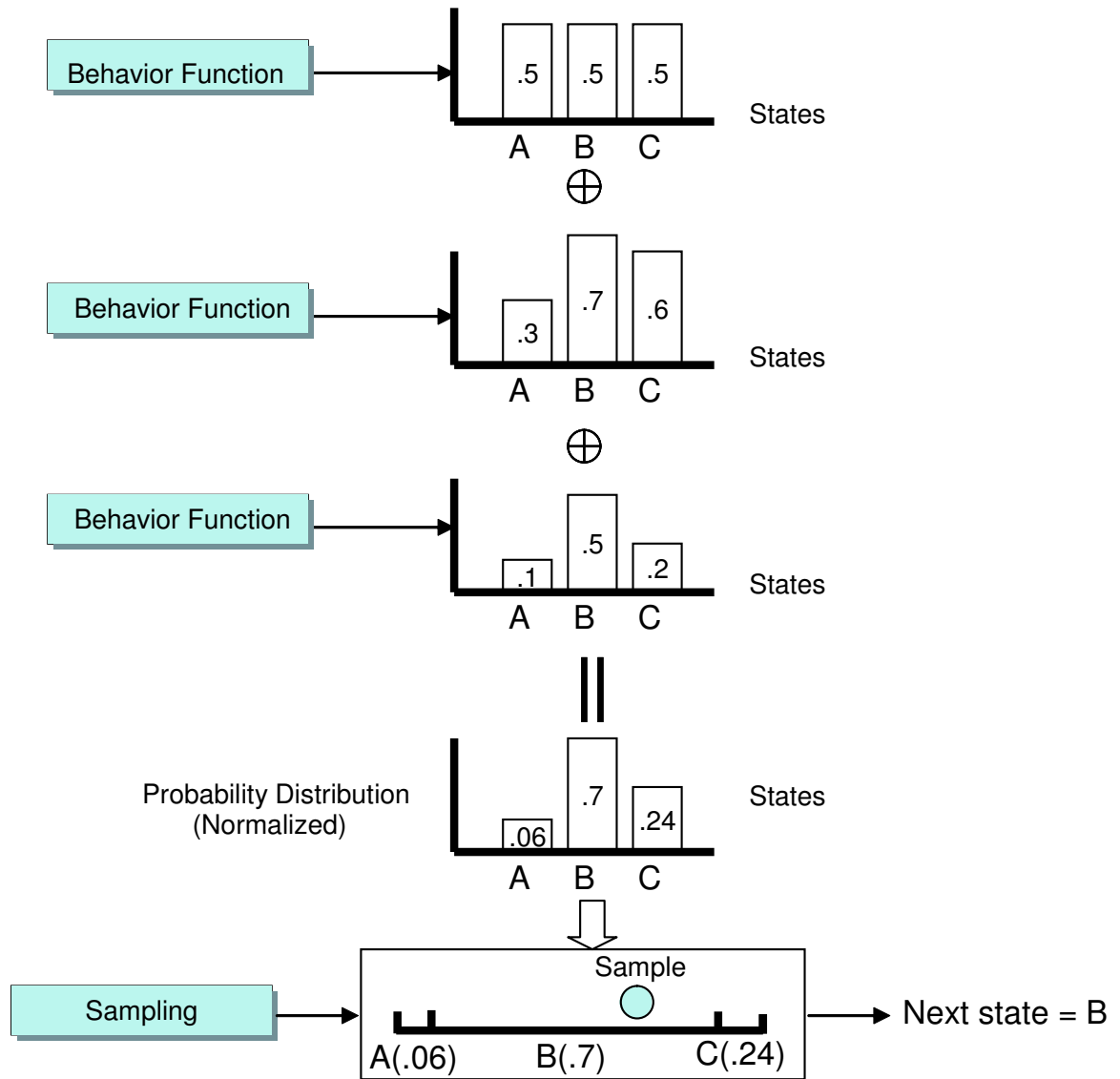
where  $n$  is the number of composed behavior functions,  $s$  is a possible next state, the  $P_k(s)$  is a probability distribution from  $k^{th}$  behavior function and  $P'(s)$  is an unnormalized probability distribution function. We use multiplication because it allows one behavior to veto a state transition (by setting its probability to 0). Finally, re-normalization should be performed on the final composed probability distribution to ease sampling. For each state  $s_i$ :

$$P(s_i) = \frac{P'(s_i)}{\sum_{j=1}^m P'(s_j)}$$

Once we have a final probability distribution, the simulation algorithm chooses the particular state transition to perform by sampling according to  $P(s)$ . The sampling allows the character to choose not only states with high probability but also those with low probability, even though it's not frequent. This gives the crowd an element of randomness.

### 3.2.2 Default States and Behaviors

If a character is not in any specific situation, we would still like him or her to exhibit certain default actions. This relieves the user from having to specify a situation for every point in the world. The default state transitions in our system are the identity transition (which does not change the character's state), five walking actions with different turn angles and six turning in place actions with different angles. Each transition takes a character from one position and orientation in the environment to another, and has an associated motion clip.



**Figure 3.3:** The behavior functions compute the probability of input states independently. Then, the probability distributions computed from behavior functions are composed. Finally, the next state is selected through sampling on the composed probability distribution.

Several different default behavior functions are combined to produce a sequence of states appropriate for a character wandering through an environment. More detail about the behavior functions can be found in Appendix.

- ***ImageLookup Behavior***: This behavior gets a bitmap describing the obstacles in the environment, and gives zero probability to states that cause the character to enter a place where some objects are located. Otherwise, it gives high probability.
- ***TargetFind Behavior***: If a character has a goal position, this behavior gives high probability to states that make the character move toward the position. Otherwise, it gives low probability. Basically, this behavior is for moving characters from one place to another through path planning. We use the Probabilistic Road Map (PRM) method [KL94a, OS94] in which many way-points are distributed randomly in the environment and linked together with Dijkstra's all-pairs shortest path algorithm including visibility. Since we compute all-pairs paths between two way points in a preprocessing step, at run-time we can find any path in real time.
- ***Collision Behavior***: This behavior gives zero probability to states that cause collision with another character. Otherwise, it gives high probability. The collision detection algorithm used in our system is a simple adaptive spatial subdivision.

These default behaviors are always composed unless a character is in a situation that specifically ignores them (see Section 3.3.2).

### 3.3 Situation and Pluggable Character Architecture

Our basic assumption is that an environment consists of a set of different situations and only a few characters are under the same situation at the same time. A situation is distinguished from

other situations by what typical behaviors the crowd can show. That is, the situation in our system controls the behaviors of a local group of characters. This provides distributed control over the crowd because we can give situation-specific behaviors to characters only when they need them. In addition, our approach provides composability among multiple situations so that characters can respond to the complex scenario when several different situations are combined.

The local crowd behaviors need several local data. The local data include local actions or they might need sensors to catch the events happening in the environment. Our situations contain that information and add it to characters dynamically. This allows character to be as generic as possible and to be adopted in wide range of environments because the local crowd behaviors depend only on what situations the crowds are in. We call it *pluggable character architecture*. By interacting between situations and the pluggable character architecture, we can control the crowd behaviors efficiently.

A situation can be any circumstance that has typical local behaviors. From the observation of real people, we easily see that one of the most important factors that affect character behaviors is its spatial location. For instance, a behavior for getting on or off a bus can be seen only at the bus stop. Therefore, we can set the “bus stop” situation around the bus stop. The relationship between people is another important factor in determining behavior. Friends usually walk together when they move. Therefore, “walking-together” is a typical behavior that can be seen among people with friendship. It means that “friendship” can be another situation. We categorize all situations into two different kinds.

- **Spatial situation:** The situation has a region that it affects in the environment (e.g., bus stop, ticket booth, and ATM machine). This situation cannot be moved after definition (which can be at run-time) but can be deleted at run-time.

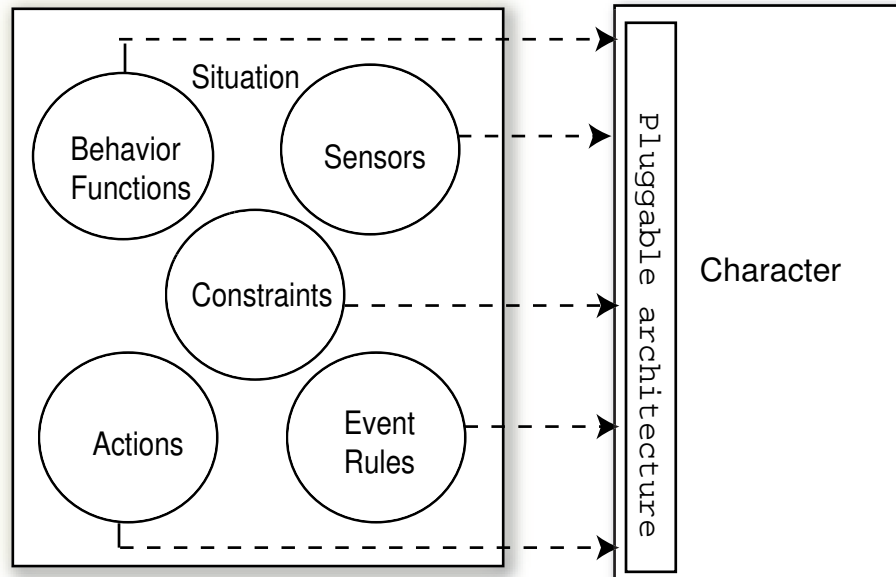


- **Non-spatial situation:** This situation (e.g., friendship, or group) does not have any region in the environment because it is attached to the crowd directly, not to the environment. Therefore, this situation can only be set at run time and characters are explicitly added or removed from the situation by the animator.

The structure of a situation is shown in Figure 3.4. The behavior functions implement behaviors specific to the situation, the sensors provide sensing capability for characters to catch events, local actions that create local states are used only for the local behaviors, and the event rule is a way to relate events to specific behaviors, the constraints are particular target positions, orientation, poses, and time duration for a particular individual. The constraints are used for the constrained motion synthesis which will be described in Chapter 5. The constraints and behavior functions are mutually exclusive. That is, if a situation plugs constraints to a character, the algorithm calls the constrained motion synthesis for the character, which makes the behavior function not necessary.

When a character is under a situation, all these components are added to his or her representation at the same time. For a spatial situation, the components are added when the character first enters the situation's region of influence. For non-spatial situations, the character is affected when the user assigns the situation to him or her. The components are removed from the character when he or she leaves the situation's region or the situation is otherwise removed. This dynamic addition and removal of behaviors enables us to achieve scalable characters.

The first thing a situation does to a character under its influence is to extend its states by adding new transitions to the character's current state transition graph. We refer to this as *extending* the graph. Because the situations add local actions, each action creates a new state that the character can be under. This increases the number of states available for selection by the probabilistic selection mechanism. An example of an extended graph is shown in Figure 3.5. In essence, each situation has its own small state transition graph that is grafted onto the character's existing graph by connecting new transition edges. Each situation knows where in the character's default graph its



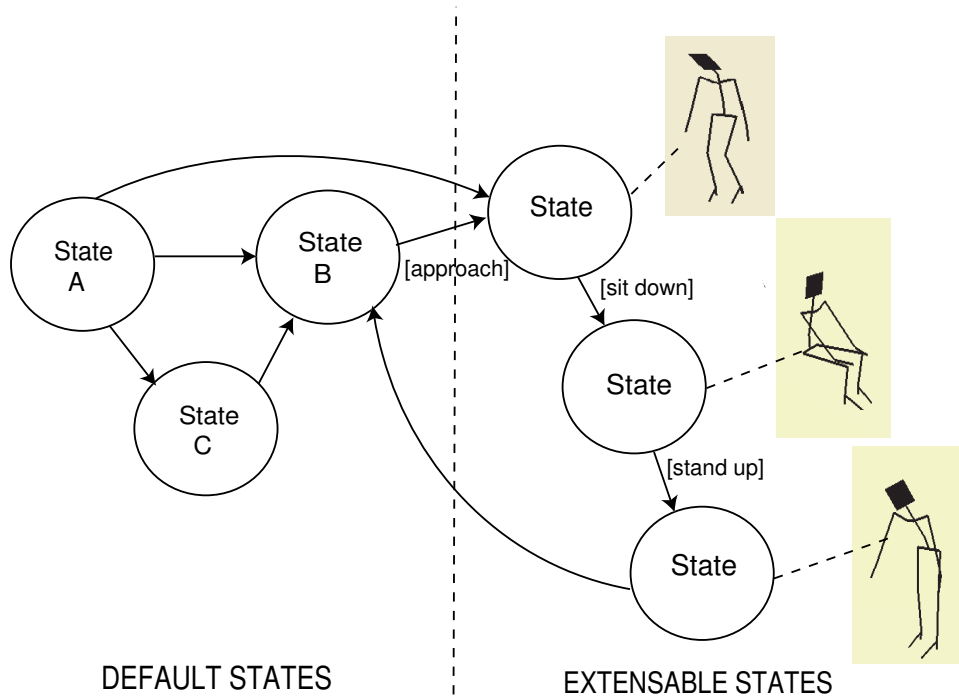
**Figure 3.4:** When a character is under a situation, he or she gets all the information from the situation directly. This includes actions, behavior functions, sensors and event rules.

own small state graph should be linked to. There can be multiple transitions between the existing graph and the new portion.

### 3.3.1 Virtual Sensors and Memory

The events control the action of local behaviors in a situation. That is, depending on what happens in the environment, different events are sent and these change the character's behaviors. The events are captured by virtual sensors which are attached to the character by a situation. The captured events are stored in a list in character's virtual memory structure. When a situation attaches a sensor, the sensor creates an entry in this list and updates it whenever the character uses the sensor. The contents of virtual memory are wiped out when he or she is no longer in the situation.

In our system, we use four different sensors to catch four different events.



**Figure 3.5:** The states are organized as a graph structure and the graph is extended by adding a new sub-graph when the character is in some situation.

- **Empty sensor:** This sensor checks for the presence of any character in a particular position in the environment.
- **Proximity sensor:** This sensor maintains the distance from a character to a particular position in the environment.
- **Signal sensor:** This sensor checks whether or not a signal in the environment is on.
- **Agent sensor:** This sensor checks a particular character’s motion and behavior including position and orientation.

Given events captured from sensors, the event rules inform the character which behavior functions to evaluate and compose, and can provide arguments to behavior functions that depend on the sensor. For example, suppose a character is in a “crosswalk situation”, then the situation attaches a

signal sensor to the character to catch the traffic sign. The associated behavior rule checks the sensor and applies a behavior suitable either for waiting (if the light says don't walk) or for crossing the street. These dynamic behaviors are composed with others to determine the character's actions. In our system, the event rules are encoded as Python script files and loaded automatically when simulation begins.

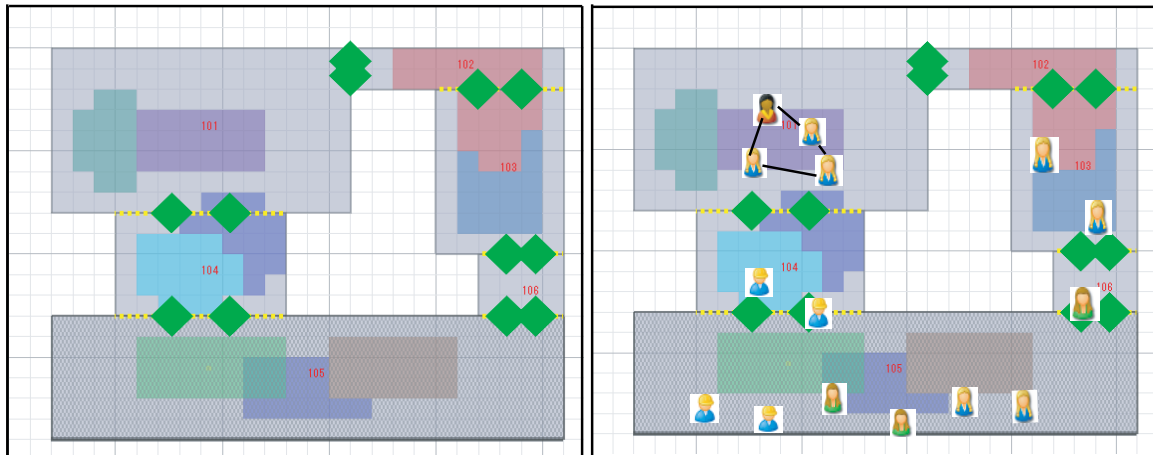
### 3.3.2 Situation Composition

In general, characters are under several situations at the same time. This is especially true of non-spatial situations such as group membership that must be composed with behaviors for fixed areas like a bus stop. In our system, the composition of two situations is similar to taking their set union; any state belonging to either is extended onto the graph, and any behavior function required by either is added, as are any sensors.

Some situations must prevent certain behaviors from occurring. For example, in the bench situation, to make a character sit down at a specific position, the "don't turn" default behavior should be ignored in composing the behaviors. This is achieved through event rules that specifically delete the undesirable behavior from the composition.

## 3.4 Painting Interface

The situation-based approach puts an emphasis on environment design because the crowd's behavior depends on where a particular situation is located (for the case of spatial situation) and what situations the characters are in. For specifying a particular situation on the environment, we adopt a *painting interface*. This allows us to specify situations easily by drawing their regions on the environment directly like drawing a picture on the canvas. Painting is particularly useful for spatial situations. For non-spatial situations, we use standard techniques to select the characters to whom the situation should apply.



**Figure 3.6:** **Left:** Spatial situations can be easily set by drawing directly on the environment. Situation composition can be specified by overlaying regions. **Right:** Non-spatial situation can be set on the crowd by grouping participants.

The painting interface is based on a layered structure where each layer represents a region for a situation and is saved as a bitmap file. The layered structure makes situation composition easy because it is done by overlapping several layers. Figure 3.6 shows the painting interface for spatial and non-spatial situations.

## 3.5 Experiments

We have performed experiments on a PC with a 1.3GHz Athlon processor and 1GB of memory. At the design stage of a crowd environment, a physical environment and situations that will be fixed for the simulation are defined with the painting interface. At run-time, non-spatial situations and run-time spatial situations are specified. The actual environment file is a Python script including links for situation files which are also encoded as Python scripts. Once all the situations and the environment are set, crowds are created either manually by the user or automatically by the program, and then simulated.

To verify our approach, we tested our system on three different environments, which are the street environment, the theater environment, and the field environment. To clarify the location of situations within an environment, we put an identifying situation number in parenthesis and use the numbers in the associated figures. The detailed explanation about the behavior function and situation can be found in appendix.

### 3.5.1 Street Environment

We tested our approach on a typical street environment where a lot of characters are walking on the street. In this environment, we made two crosswalks and two sidewalks on a city street (Figure 3.7 and Figure 3.8). One crosswalk has a traffic sign and the other one does not. For the crosswalk with a traffic sign, we composed two situations: the “crossing street” situation (1) and the “traffic sign” situation (2). For the unsigned crosswalk, we just used the unsigned “crossing street” situation (4). Also, in the middle of street away from the two crosswalks, we put an “in-a-hurry” situation (3).

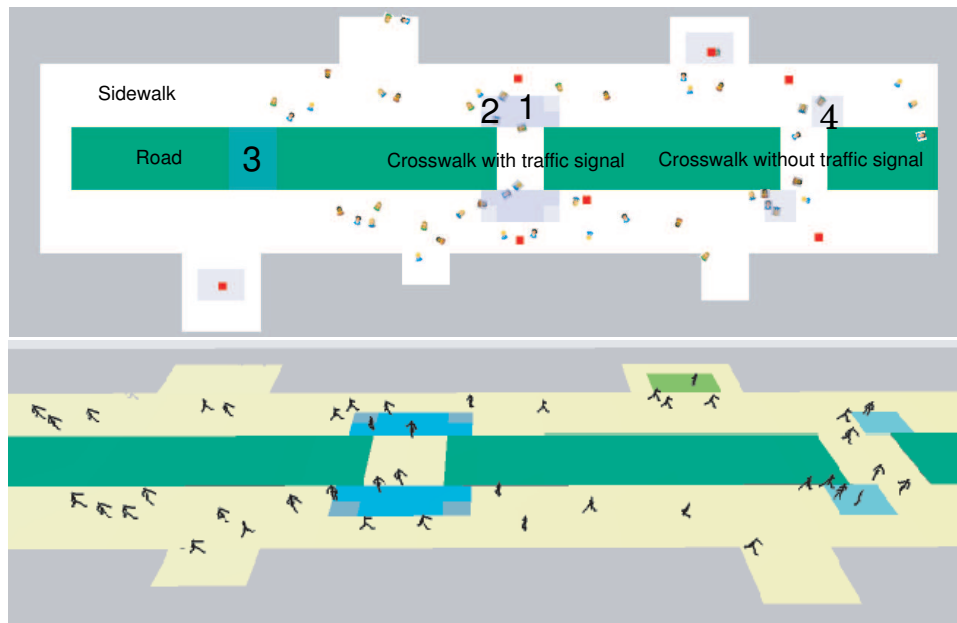
The more detailed specification of situations are follows:

- *Crossing street* situation (Goal : Make crowd crossing street)
  1. **Actions:** no local actions.
  2. **Behavior functions:** *target finding* behavior function(Refer to the appendix for detail).
  3. **Sensors:** no sensors.
  4. **Event rules:** no event rules.
- *Traffic sign* situation (Goal : Make crowd crossing street depending on traffic sign)
  1. **Actions:** no local actions.

2. **Behavior functions:** *standing* behavior function and *target finding* behavior function (Refer to the appendix for detail).
  3. **Sensors:** an *event* sensor for capturing traffic sign.
  4. **Event rules:** a rule for calling *standing* or *target finding* behavior function depending the signal captured from the event sensor.
- *in a hurry* situation (Goal : Make crowd run across the street)
    1. **Actions:** running actions
    2. **Behavior functions:** *running* behavior function that gives high probabilities on running actions.
    3. **Sensors:** no sensors.
    4. **Event rules:** a rule for unplugs the running behavior function from behavior composition when the character finishes the crossing.

At the beginning of simulation, people are walking on the sidewalk. But when they meet the crosswalks, they begin to respond to the situations. At the crosswalk with a traffic sign, they first check the traffic sign using a sensor that is provided by the situation, and wait for the traffic signal before crossing. At the crosswalk without a traffic sign, on the other hand, people just cross the street. Meanwhile, if they are in the “in-a-hurry” situation, they cross the street without using crosswalks. The “in-a-hurry” situation adds running transitions to the character’s state transition graphs, allowing the crowd to run rather than walk when crossing in an unmarked area.

Due to the sampling strategy, some people who are trying to cross the street might be turning back to the sidewalk at the middle of crosswalk. To solve this problem, we can adjust the weighting constant  $\alpha$  in equation 3.1 for the crossing behavior function so that once they are crossing the street, they continue to walk until they reach the other side of crosswalk.



**Figure 3.7:** Top: Plan of the street environment (numbers indicate situations). Bottom: 3D view of the street.

### 3.5.2 Theater Environment

In this experiment, we made a complex theater environment in which there are four different kinds of rooms. These rooms are a ticket booth, a lobby, a movie room, and restrooms. In each room, we set several situations and have some of them overlapped and hence composed. Figure 3.9 shows the theater environment. The various regions are:

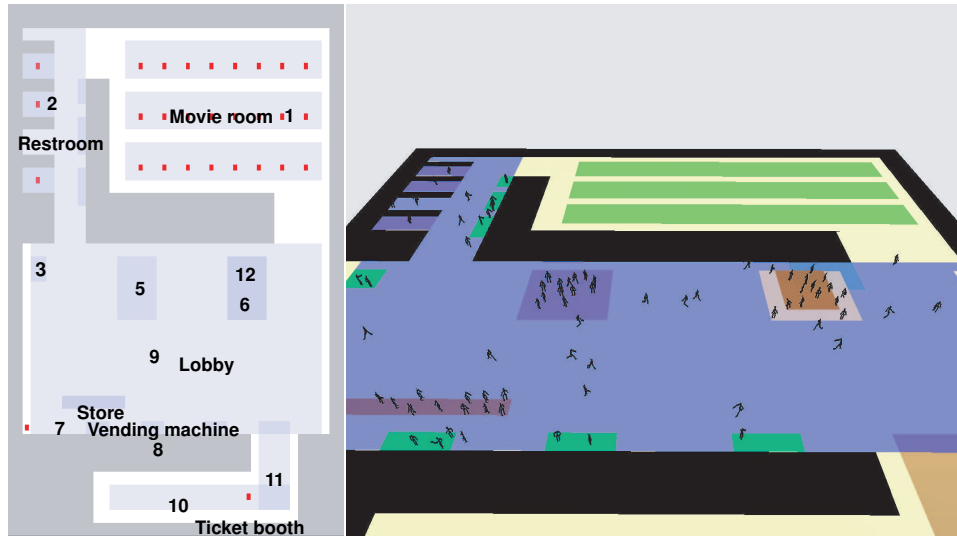
- **Ticket booth:** In the ticket booth, we set a “horizontal queue” situation (10) and a “follow” situation (11). When characters enter the “horizontal queue” situation, they stand in a line and try to buy a ticket, one by one. After that, they are guided by the “follow” situation and enter the lobby of the theater.
- **Lobby:** In the lobby, we set two “gathering” situations (5, 6), a “stay-in” situation (9), another “horizontal queue” situation (7), a “talk” situation (12) and three “vending machine”





**Figure 3.8:** Crowd behavior rendered in a game engine.

situations (8). Among the two gathering situations, one (5) is composed with a “talk” situation (12). For the case of (6), only a “gathering” situation is used. When characters meet the “gathering” situation (6), they gather around for a predefined duration and then spread apart. On the other hand, in the case (5), they gather around for the predefined duration as well, but also sometimes show the talking behavior due to the “talk” situation. At the “vending” machine situation, the crowd stops for a little while, and then moves forward. It is intended to simulate purchasing something from the machine (we were limited by a few available motions). The “stay-in” situation covers the lobby and restrooms and keeps the crowd in those areas before the movie time. This situation adds a signal sensor to all characters in the area. If the signal from the sensor is off, they stay in the lobby or in the restrooms. Otherwise, they move to the movie room through path planning.

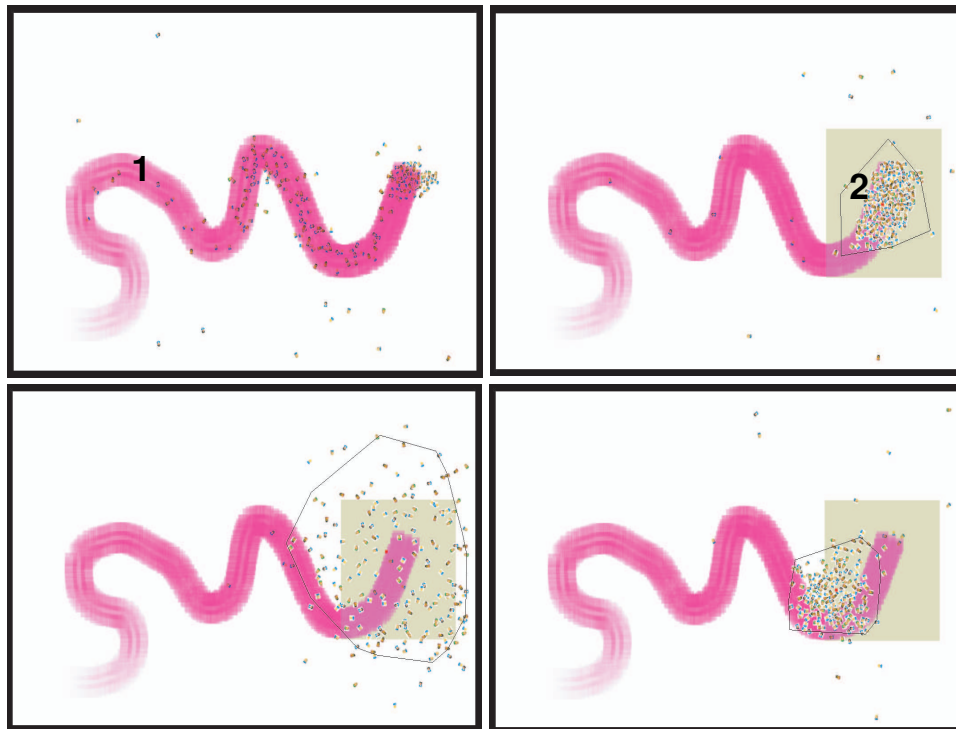


**Figure 3.9:** **Left:** Plan of the theater environment (the number indicates situations). **Right:** 3D view of the theater.

- **Restroom:** In the restroom, we put three “bench” situations (3), and three “exclusive” situations (2). The “exclusive” situations prevent more than one person from getting into the same restroom at the same time.
- **Movie room:** In the movie room, we set three “seat” situations (1). Each situation makes seven characters sit down in the region, one on a seat. When people are in the situation, they are provided empty sensors and proximity sensors. Using these sensors, they know which seats are empty and which seats are occupied. If they find an empty seat, they move there and sit down.

### 3.5.3 Field Environment

In order to show that situations can be set at run-time, we create a simple field environment and put a relatively large crowd of 200 characters in the environment (Figure 3.10). At run-time, a “follow” situation (1) is set by the painting interface. Due to this situation, the crowd flows through



**Figure 3.10: Top Left:** A crowd is reacting to a “follow” situation. **Top Right:** We set a “group” situation on the crowd without adding any other behaviors. **Bottom left:** Due to the group situation, the crowd spreads around (we have disabled the “follow” situation). **Bottom right:** People gather back when we compose a “close” situation with the group situation.

the region and gathers at its end. At that point, we set a “group” situation (2) on the crowd (a non-spatial situation). At the same time we disable the “follow” situation, which results in the crowd spreading around. However, if we compose a *close* behavior function with the “group” situation, they gather back.

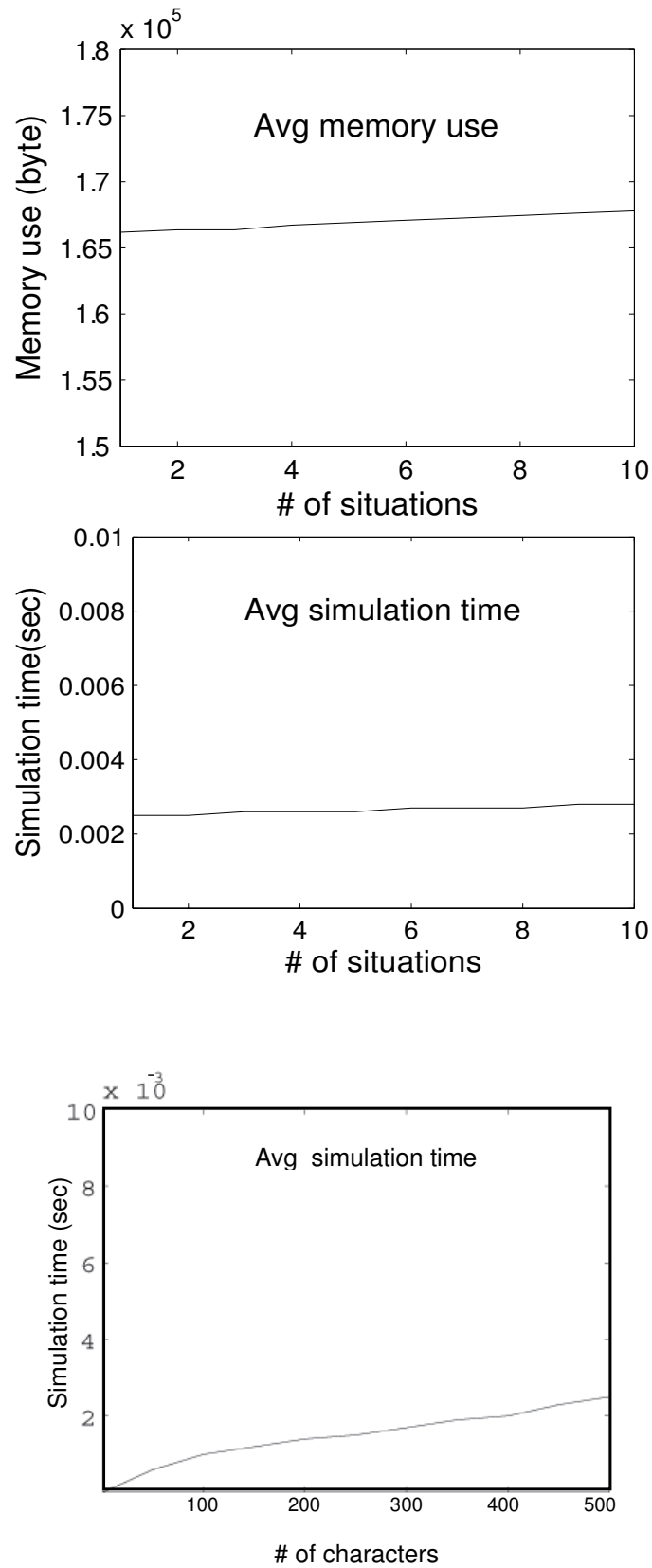
### 3.5.4 Validation

For validating the ability of our approach to meet the scalability demand, we conducted three experiments. First, we measured the average memory use of 500 characters for 2,000 simulation steps as the situational complexity of the environment increased. We built up an empty room

with the painting interface and put an increasing number of randomly selected situations in the environment. As we increased the number of situations, we computed the memory usage at each of 2,000 simulation steps before averaging across steps. To examine the rate of growth of simulation time as a function of situational complexity, we repeated the experiment but this time computed the average amount of time spent on animating the environment (excluding rendering) per frame of motion, or  $\frac{1}{30}$ th of a second. The results are shown in Figure 3.11. The density of crowd was 10.0 and the physical complexity of environment was zero (no obstacles). Note that the complexity grows slowly according to a function of the number of situations, suggesting our approach is successful.

We also examined the cost of simulating increasing the number of characters with a fixed number of situations (10). Again we averaged the time taken to simulate each frame of motion. The density of crowd (10.0) and the complexity of environment (0.0) was same to the second experiment. The results are shown in Figure 3.11. Even for 500 characters we can compute all the behavioral and motion information in around 2.5ms, representing less than 7.5% of the time available for each frame.

Our simulation result confirmed that crowds showed particular local behaviors when they met the situations. This validated the controllability demand. The painting interface was quite useful for setting situation on the environment. As we showed in section 3.4, the painting interface allowed us to set a particular situation directly on the environment by painting situation strokes which were associated with the situation on the environment. The painting strokes were designed at the preprocessing step. For implementation of situation strokes, we used the python script wherein all information about the environment was encoded as a simple script and brought up when we selected the situation stroke. The selected script was added to the main simulation script which described the world and specification of where the situations were. In this way, we could add any number of situations on the environment and control the local behaviors of crowd easily.



**Figure 3.11: Top:** The average memory use of 500 characters for 2,000 simulation steps. **Middle:** The average simulation time of 500 characters for 2,000 simulation steps. **Bottom:** The average simulation time of crowd with the fixed number of situations.

The convincingness demand was validated through the motion quality of individuals and proper behaviors of crowd. In our experiments of the 3.5, we have showed that the character showed no artifact such as pops in their motions and there was no collision between characters during simulation.

## 3.6 Discussion

The situation-based simulation with the probability scheme introduced a new framework for synthesizing virtual crowds in complex environments. At the high level, we use a situation-based approach that provides a scalable mechanism to control the local behaviors of the crowd. At the low level of the framework, we adopt a probability scheme that composes the influence of several behaviors to drive a realistic motion synthesis system. We have demonstrated that our framework can create complex crowd behaviors through the composition of situations and the composition of behaviors while minimizing data stored in each character.

The first advantage of this approach is ease of authoring; it breaks the problem of character design into the design of local activities, rather than one monolithic system. Second, re-use is enhanced; the core behavior and actions can be used in any environment and the situation specific modules can be shared wherever situations are shared. Third, efficiency is improved; at any given moment a character has information only for the situation that he or she is in – not all of the information for the entire environment. Finally, de-centralized control of characters makes overall performance scalable, especially when a large crowd is simulated; each situation takes care of only a small number of characters.

There are several ways in which we could improve our system. We have not experimented with situations that control the density of the crowd or other multi-agent statistics. This could be done with more intelligent situations that acted as simulation entities in their own right by dynamically

adjusting the behaviors they add to characters. From an efficiency standpoint, in our current system we assume that all crowds go through the simulation step at the same time. In order to simulate a massive crowd like 10,000 people, we need to avoid this work in some way, possibly by composing longer pieces of motions that hence require less frequent transitions.

Finally, while our probabilistic composition framework is efficient and works well in the situations we have experimented with, it has limitations as the time scale of actions increases. In other words, the character needs to remember more and more past states in order to piece together long running actions. We would like to explore other mechanisms for combining behaviors.

Our method is a significant advance in scalable behaviors for characters. Architectures such as ours will be essential for controlling the design complexity of virtual characters as the environments they populate continue to evolve.

The range of applications that our approach can simulate heavily depends on motion data. If we don't have available motion clips, we cannot show behaviors that use those motions. Also, our agent architecture does not contain emotional status of agents. From the psychological standpoint, the people's behavior is mostly influenced by the internal status of them. In our approach, we do not take care the behaviors driven by emotional motivation, which is limit the number of behaviors we can simulate.

## Chapter 4

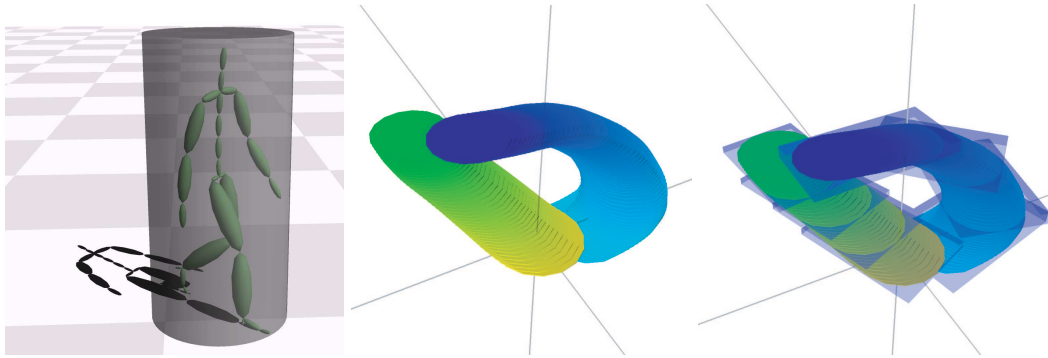
# Collision Detection Using MOBB Trees

In this chapter, we present an efficient collision detection algorithm between characters whose motions are animated with motion capture data. This algorithm uses a novel MOBB tree representation of motions to simplify motions in spatio-temporal domain and test the overlap between them hierarchically. We first explain the background of collision detection in section 4.1. Sections 4.2 and 4.3 precisely define MOBB trees, describe their construction and present algorithms for intersection testing. We close with several experiments and a discussion of future work in section 4.4.

### 4.1 Introduction

Data driven animation based on motion capture is a common way to animate human characters. When multiple characters are animated this way, care must be taken to provide proper interaction between them. A minimal requirement is that they do not interpenetrate. A better goal is that two characters interact appropriately when they are close. In either case, when a system selects captured motion clips for characters, it must check whether these motions will lead to the characters colliding so they can respond appropriately. This chapter presents algorithms for identifying potential collisions between motion clips, and hence animated characters.



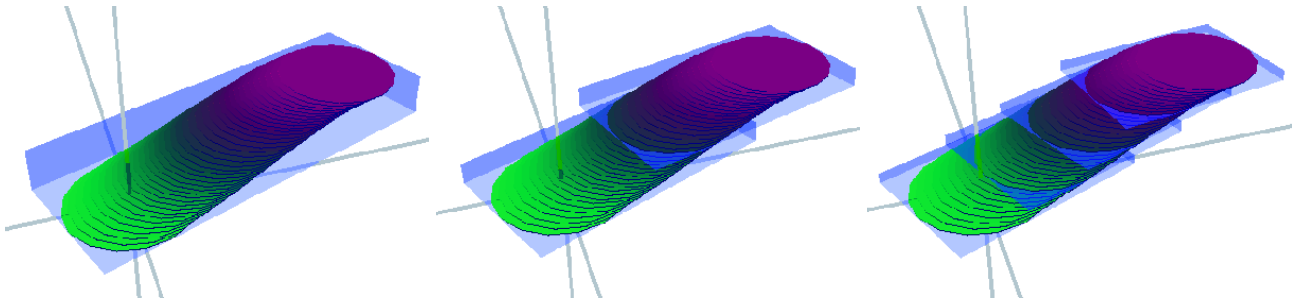


**Figure 4.1:** *On the left, a sample frame of motion data from a clip used in our system. The skeleton is bound by a vertical cylinder centered on its root position with a radius large enough to contain the limbs. In the middle there is a 3D space-time visualization of two motions that do not intersect in time (vertical axis) even though their paths cross in space. Each cylindrical bound is now a thin disk - a circle in space extruded in time. On the right, part of the Motion Oriented Bounding Box (MOBB) for the motions. This hierarchical structure bounds the motion in space-time and enables efficient collision queries.*

When many characters are animated simultaneously, such as in crowd simulation, efficiency is an important concern. Detecting interactions between fully articulated characters is computationally intensive, and may be excessive if the application requires characters to maintain reasonable separations. We therefore focus on the use of a simplified character geometry; a bounding cylinder. In cases where this simplification is unacceptable, the efficient algorithms it affords can be used as a culling step to identify potentially interacting characters.

Even with the simplified geometry, checking for interactions between characters on every frame can be prohibitive if there are many characters. The problem is even worse if we are evaluating potential choices for motions for each character. We therefore exploit the fact that data driven animation often selects an entire clip of captured motion at once. We can, therefore, check entire motions for collisions before they begin.

The motion clip collision problem considers two motions, along with their transformations that place their starting points in space. One motion may have a time offset, for the event that the two do not start simultaneously. A collision must be detected if at any time during the duration of



**Figure 4.2:** *Three levels in an MOBB tree hierarchy. Boxes are split evenly in the time (vertical) dimension going from one level to the next, and then spatial bounds are generated containing all the samples in that time-slice.*

the motions the bounding cylinders for the characters associated with the motions overlap. If the characters are spatially disjoint, or at the same place at different times, there are no collisions.

To perform motion clip collision detection efficiently, we precompute a hierarchical representation of each motion. We refer to our collision data structure as a Motion Oriented Bounding Box (MOBB) tree (Figure 4.1, right). It is a space-time variant of OBB trees [GLM96] targeted at skeletal motion clips, and it can be viewed as a continuous collision detection technique based on hierarchies of swept volumes. The design of MOBB trees is motivated by several motion specific properties: the character’s path is densely point sampled in time and can be arbitrarily shaped (Figure 4.1, center); the time steps are large compared to typical physically based simulation; the aim is to avoid collisions entirely, so we require a yes/no intersection test and have no need for contact points etc.; and characters are moving on the ground plane, so the problem is 2D with time (we are looking for intersections between circles extruded in time – thin disks in space-time). These properties primarily drive the way in which an MOBB tree is constructed, but also influence the intersection testing algorithm.

## 4.2 MOBB Trees

The motion clip collision detection problem considers a pair of motions,  $\mathbf{m}_a(t)$  and  $\mathbf{m}_b(t)$ . A motion is a function that maps times to character poses. Because we simplify character geometry as a cylinder, we consider motion functions as providing three values for any given time: the position of the character in the plane  $(x, y)$  and the radius of the cylinder around the character  $r$ . For simplicity, we center the cylinder around the projection of the character's root joint onto the ground plane. We compute  $r$  from the point on the character whose projection is furthest from this position. When the character is asymmetric, for example when carrying a large sword, the use of the root as center leads to an oversized bounding cylinder. In practice, this has not been a problem. In fact, we often use a fixed value for  $r$ .

Because we are working with data-driven animation, motions are represented by samples. We assume, without loss of generality, that the first sample is at time  $t = 0$ . We also assume that the motions are sampled finely enough that we need not consider interpolation between samples.

A collision detection test is given two motions,  $\mathbf{m}_a$  and  $\mathbf{m}_b$ , a 2D transformation for each,  $\mathbf{T}_a$  and  $\mathbf{T}_b$ , and a time offset for each,  $t_a$  and  $t_b$  measured in a global timeframe. Define  $t_{start}$  as  $\max(t_a, t_b)$  and  $t_{end,a}$  and  $t_{end,b}$  as the last sample in  $\mathbf{m}_a$  and  $\mathbf{m}_b$  respectively. The test should return a positive result (an intersection) if there exist sample times,

$$\begin{aligned} (t_{start} - t_a) &\leq o_a < t_{end,a} \\ (t_{start} - t_b) &\leq o_b < t_{end,b} \end{aligned}$$

such that

$$\|\mathbf{T}_a \mathbf{m}_a(o_a) - \mathbf{T}_b \mathbf{m}_b(o_b)\| < r_a(o_a) + r_b(o_b) \quad (4.1)$$

In practice, the time offsets could be arbitrary real numbers, while the samples are discrete. We therefore interpret  $\mathbf{m}_a(o_a)$  to be the sample from the time closest to but below  $o_a$ . An alternative is

to interpolate  $\mathbf{m}_a$ , but our sample spacing is sufficiently fine with respect to the character's speed and size that this is unnecessary for the purposes of collision avoidance.

In 3D space-time, each sample is a short cylinder, axis aligned with the time dimension, the center of the base at  $(\mathbf{m}(t); t)$ , radius  $r(t)$  and height equal to the sample spacing,  $dt$ . Detecting a collision is equivalent to identifying collisions between these space-time cylinders, appropriately transformed.

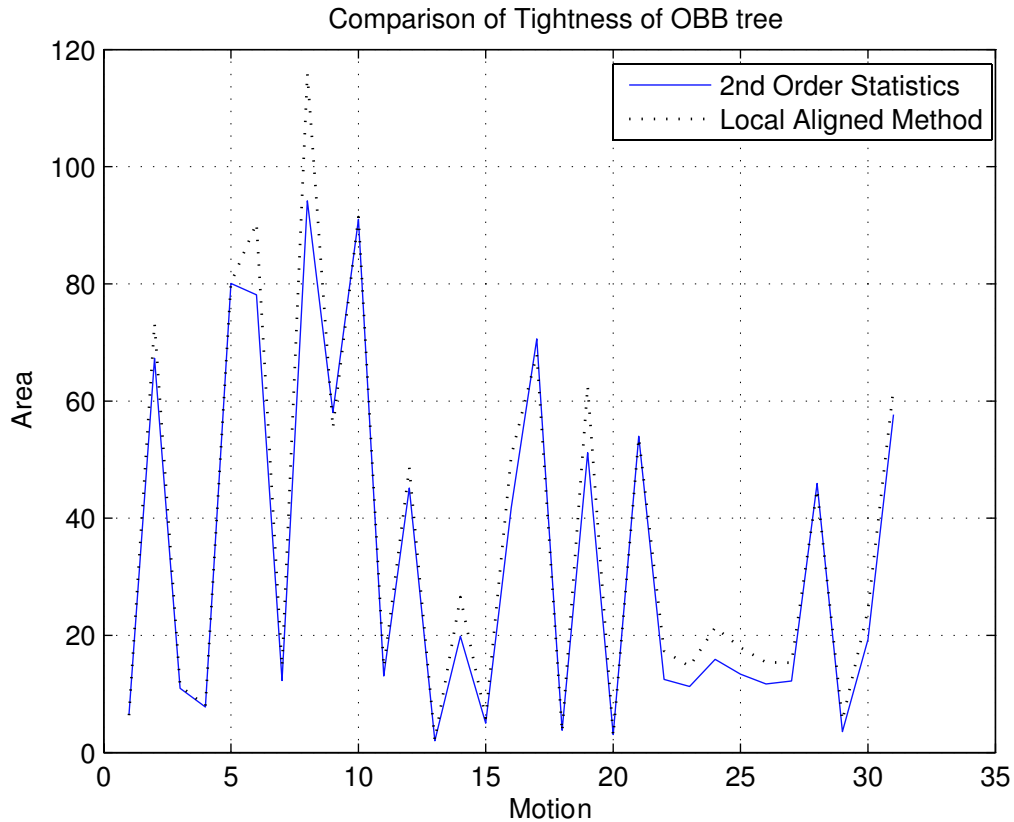
An MOBB tree is a hierarchical bounding volume in 3D space-time. Each node in the hierarchy is an oriented bounding box with one axis parallel to the time domain and the other two axes lying in the  $xy$ -plane. This is equivalent to a 2D spatial OBB extruded in the time domain. Each node bounds a set of samples from  $t_{min}$  to  $t_{max}$ . The children of a node in the tree bound the subsets of samples from  $t_{min}$  to  $\frac{t_{max}-t_{min}}{2}$  and  $\frac{t_{max}-t_{min}}{2}$  to  $t_{max}$ . In other words, the hierarchy is built by subdividing the volume at its midpoint in time.

The following sections describe how we construct a MOBB tree from a motion, and how we intersect two MOBB trees.

### 4.2.1 Fitting MOBB Trees

MOBB trees are built in a manner analogous to standard OBB trees; given a sequence of samples to bound, we must compute the orientation and dimensions of the box, and then recurse on the two child sub-sequences. Note that in each node we store data defining a 2D OBB tree and the time range, which can be thought of as the third dimension of the space-time box. Three levels of an example tree are shown in Figure 4.2.

Given a set of sample points in 2D, the optimal oriented bounding box (OBB) can be computed by computing the second order statistics of the points [GLM96]. In practice, we find it sufficient to approximate the optimal box with one obtained by a simpler method. Figure 4.3 shows the result of comparison between our axis aligned method and original second order statistics method.



**Figure 4.3:** Comparison between second order statistic method and our local aligned method.

Given a set of sample points, we choose the major axis of the OBB,  $\mathbf{a}_0$ , by subtracting the location of the first sample,  $\mathbf{m}(t_{min})$  from that of the last,  $\mathbf{m}(t_{max})$ , and normalizing:

$$\mathbf{a}_0 = \frac{\mathbf{m}(t_{max}) - \mathbf{m}(t_{min})}{\|\mathbf{m}(t_{max}) - \mathbf{m}(t_{min})\|}$$

The minor axis is perpendicular:  $\mathbf{a}_1 = (-a_{0,y}, a_{0,x})$ .

To compute the dimensions of the box, we iterate over the samples and keep track of the maximal extents seen. To compute these, each sample is transformed into the box's local coordinate system and  $r(t)$  is added (subtracted) to get the maximal (minimal) extent in each dimension. If  $\mathbf{d}_{max}$  and  $\mathbf{d}_{min}$  are the largest and the smallest extents found, then the center of the OBB is at

$$\mathbf{c}_{local} = \frac{\mathbf{d}_{max} + \mathbf{d}_{min}}{2}$$

and the dimensions of the box are

$$\mathbf{d} = \frac{\mathbf{d}_{max} - \mathbf{d}_{min}}{2}$$

The center is transformed back into global coordinates and stored, along with  $\mathbf{a}_0$ ,  $\mathbf{a}_1$  and  $\mathbf{d}$ . Each node also stores  $t_{max}$  and  $t_{min}$ .

After computing the bound at one node, we divide the duration of the sample sequence in half. Recursion stops when a fixed number of samples,  $n_{min}$ , remain and the samples are stored in the node (OBBs are still computed and stored for leaf nodes). We experimented with various values for  $n_{min}$ , ranging from 1 to 50. Optimal performance occurred at  $n_{min} = 10$ , which reflects the cost of a 2D OBB intersection relative to computing the distance between two samples. Performance is near best when one OBB test is equivalent to  $\frac{n_{min}}{2}$  sample distance tests.

Standard OBB tree construction algorithms [GLM96] use second order statistics to determine the box axes. We tested this method but found it gave essentially identical results (comparing box area) as our method, which is simpler to implement.

## 4.2.2 Intersection Testing

Intersection testing of two MOBB trees is very similar to testing standard OBB trees. Note that we are seeking only yes/no intersection queries, and hence can exit as soon as an intersection is found. Input to the intersection test is two MOBB nodes,  $A$  and  $B$ , their 2D spatial transformations from world coordinates,  $\mathbf{T}_a$  and  $\mathbf{T}_b$  (rotation and translation) and the time offsets,  $t_a$  and  $t_b$ . An example collision test is presented in Figure 4.4.

We first test that the nodes overlap in time: if  $t_a + A.t_{max} < t_b + t_{min}$  then the boxes do not overlap in time (Figure 4.4(c)) and we can exit with no collision, and the same if  $t_a + A.t_{min} > t_b + t_{max}$ . If temporal overlap is found, we test  $A$ 's and  $B$ 's 2D OBBs using separating axes tests. There are only four axis tests required. If the OBBs do not overlap, there is no collision (Figure 4.4(b) and (d)), otherwise we perform one of two actions (Figure 4.4(a) and (e)): if one of

the boxes is a leaf, we call a procedure to test the leaf against the other tree; otherwise we make recursive calls to compare all the child nodes.

A leaf versus tree node test checks the time interval covered by the leaf against the time intervals covered by the node's children (Figure 4.5). If a child overlaps the leaf, we recurse on the child. Recursion continues until we have two leaves, at which point corresponding samples are found from  $A$  and  $B$  and the distance between them compared to the bounding radii. In other words, we explicitly search for samples with  $o_a$  and  $o_b$  satisfying Equation 4.1.

### 4.3 Unrestricted MOBB Trees

The MOBB trees described thus far always use the time axis as one of the OBB axes in 3D space-time. This restriction can be relaxed, essentially treating the samples as regular 3D geometry and building 3D OBBs to bound them. We refer to trees built in this manner as Unrestricted MOBB trees, or UMOBB trees. UMOBB trees are expected to give tighter space-time bounds (Figure 4.6), and hence require fewer tests to identify non-intersecting cases.

To determine the axes of the 3D OBB in space-time, we determine the first axis,  $\mathbf{a}_0$ , by subtracting the space-time location of the first sample,  $(\mathbf{m}(t_{min}); t_{min})$  from that of the last,  $(\mathbf{m}(t_{max}); t_{max})$  and normalizing (similar to the 2D case, but including the time dimension). We obtain a second axis,  $\mathbf{a}_1$ , mutually orthogonal to  $\mathbf{a}_0$  and the time dimension,  $(0, 0, 1)$ . Finally,  $\mathbf{a}_2 = \mathbf{a}_0 \times \mathbf{a}_1$ .

The dimensions of the box are found by transforming the sample points into the box's coordinate system, projecting the extents of the samples' cylinders onto the axes and hence finding the maximal projection across all samples. Figure 4.7 illustrates the projection. First we compute  $\mathbf{r}$ , which is the vector with length equal to the disk's radius,  $r$ , aligned with  $\mathbf{a}_0$  in the  $xy$ -plane:

$$\mathbf{r} = r \frac{(\mathbf{a}_{0,x}, \mathbf{a}_{0,y}, 0)}{\|(\mathbf{a}_{0,x}, \mathbf{a}_{0,y}, 0)\|}$$

From the figure, we see that

$$\min_0 = -\mathbf{r} \cdot \mathbf{a}_0$$

$$\min_2 = \mathbf{r} \cdot \mathbf{a}_2$$

$$\max_0 = \mathbf{r} \cdot \mathbf{a}_0 + dt \mathbf{a}_{0,t}$$

$$\max_2 = -\mathbf{r} \cdot \mathbf{a}_2 + dt \mathbf{a}_{2,t}$$

The extents in the remaining direction,  $\mathbf{a}_1$ , are at distance  $r$  by construction. Taking the maximum and minimum over all samples gives us the necessary information to compute the box origin and dimensions. The node of an UMOBB tree stores the box properties (origin, axes, dimensions) in addition to  $t_{min}$  and  $t_{max}$ . Keeping the times allows for a fast early reject test when looking for box intersections.

Intersection testing of UMOBB trees is essentially identical to that of MOBB trees, the only difference being the use of 3D OBB tests. A 2D transformation plus a time offset becomes a 3D space-time transformation by applying the rotation about the  $t$ -axis and using the temporal offset as a translation in the  $t$  dimension. Otherwise the algorithm is identical.

## 4.4 Validation

Fast collision detection between two motions is a crucial part for visual convincingness and efficiency demand. To validate these demands, we performed a series of experiments to explore the benefits of MOBB trees under an application workload. Our test environment is a crowd simulator in which characters wander through the world avoiding collisions (Figure 4.8). The density of crowd is 9.5 and the physical complexity of environment is 0.0 (no obstacles). The characters are animated with a Snap-Together Motion [GSKJ02] style motion graph built from 51 motions. The motions have an average length of only 2.1 seconds, or 63 frames. With  $n_{min} = 10$  (the number of samples in a leaf node), the tree is very shallow – only 3 levels on average. This limits to some



Method	Time ( $10^{-6}$ s)	# 2D	# 3D	# sample
MOBB 1 level	4.6	0.45	0	10.3
UMOBB 1 level	4.5	0	0.45	10.0
MOBB	2.3	1.7	0	0.91
UMOBB	2.2	0.16	1.53	0.77
Hybrid 1	2.1	0.40	1.34	0.77
Hybrid 2	2.0	1.5	0.25	0.81

**Table 4.1:** Results of our application-based experiment. See the text for a description of the methods. The table shows average time per intersection query, the average number of 2D and 3D OBB tests per intersection query, and the average number of sample-sample overlap tests. Note that, even though the UMOBB contains no explicit 2D nodes, some 3D boxes end up aligned and hence are treated as 2D. This explains the non-zero count for 2D tests in the UMOBB trees.

extent the benefits we see from a hierarchical method. The radius of the bounding sphere around each character was fixed at  $r = 1.85$  meters.

As a base case for comparison, we used MOBB trees that performed bounding box tests only at the root, referred to as “1 level” trees in Table 4.1. These trees simulate a collision detection method that tests an OBB bound for the entire motion, and then use binary search on time to identify potentially overlapping samples (a hierarchy in time but not space). Our results hence show the performance advantage gained from a hierarchy of bounds compared to an algorithm that uses only a root bound but is otherwise intelligent about avoiding sample tests.

In addition to results for the “1 level”, MOBB and UMOBB trees, we also experimented with two hybrid trees: one used an MOBB node for the root and UMOBB nodes for the rest of the tree; while the other used a UMOBB node for the root and MOBB nodes elsewhere. These are listed as “Hybrid 1” and “Hybrid 2” in Table 4.1.

Method	Time ( $10^{-6}$ s)	# 2D	# 3D	# sample
MOBB 1 level	68.3	0.83	0	306
UMOBB 1 level	68.1	0	0.83	307
MOBB	5.9	27.5	0	4.15
UMOBB	6.8	0.62	21.6	3.10
Hybrid 1	6.8	1.45	20.8	3.10
Hybrid 2	6.0	26.7	0.83	4.15

**Table 4.2:** *Results of our experiment using longer motion clips. The methods are described in the text. The table shows average time per intersection query, the number of 2D and 3D OBB tests performed, and the number of sample-sample overlap tests. We see MOBB trees slightly out-performing the unrestricted tree, reflecting the relatively high cost of 3D OBB tests compared to 2D tests.*

The experiments were performed on a PC running Windows with a 3.0GHz Athlon processor. Each experiment ran for 2 minutes of simulated time. Approximately 80% of all queries returned negative at the root node test, which is a sufficiently large percentage to make the fast but inaccurate. The MOBB trees’ 2D OBB test performs very similarly to the more expensive UMOBB trees’ 3D OBB test at the root node level (the results for “Level 1” testing). However, at nodes deeper in the tree the MOBB boxes improve in fit, and they perform better than UMOBB nodes due to their cheaper cost per test. The hybrid trees confirm this result. Overall then, in this environment it matters little which hierarchical method we use.

The short motions of our target environment limit the performance gains available through a hierarchical method. We see only around a factor of 2 improvement over a non-spatial (but still temporal) hierarchy. Longer motions provide greater performance improvements, so we conducted another experiment using a simulation style workload (in terms of percentage of positive tests) but in an isolated test environment.

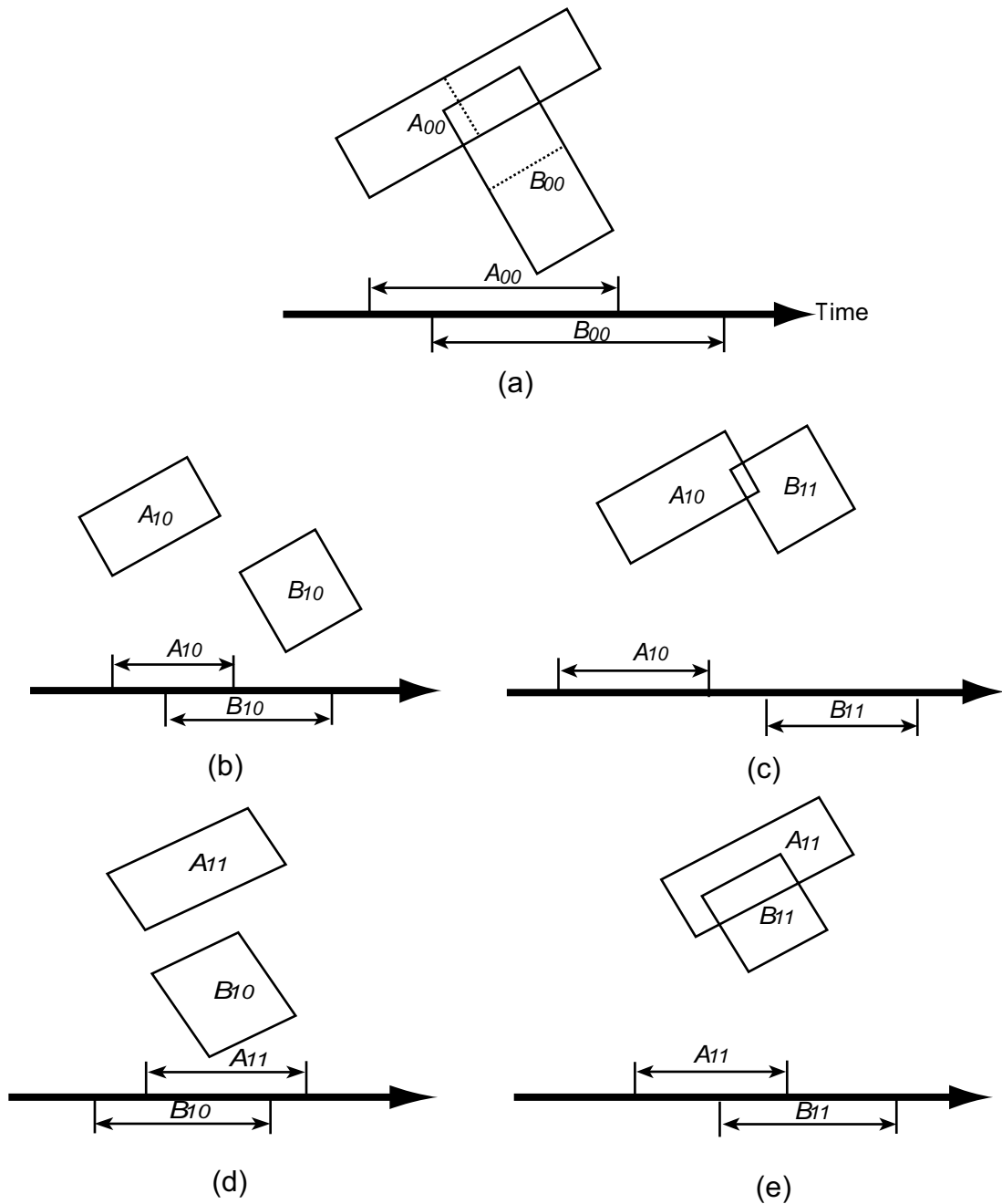
Our second experiment used 14 motions with an average length of 42 seconds. We performed an identical set of 100,000 tests with each style of tree, each test using a random translation, rotation and temporal offset on one of the motions. These tests were done on a 3GHz Pentium 4 PC running Linux. The results are in Table 4.2, and an example motion appears in Figure 4.9. On longer motions, MOBB trees perform best by a small margin, and the almost identical performance of the “Hybrid 2” trees (containing almost all MOBB nodes) supports the conclusion that faster tests with looser MOBB bounds are preferable in this application to the slower UMOBB tests. Regardless of the exact type of bounding tree, we consistently see roughly an order of magnitude improvement. Intuitively, the motions are long enough to allow pairs to frequently start nearby (meaning their root nodes overlap) and move away from each other (meaning that spatial testing is effective when temporal is not). Figure 4.10 shows the result of time cost comparison among MOBB, Frame-by-Frame and SweepOBBTree method (the density of crowd = 3.3 and the physical complexity of environment = 0.0).

## 4.5 Discussion

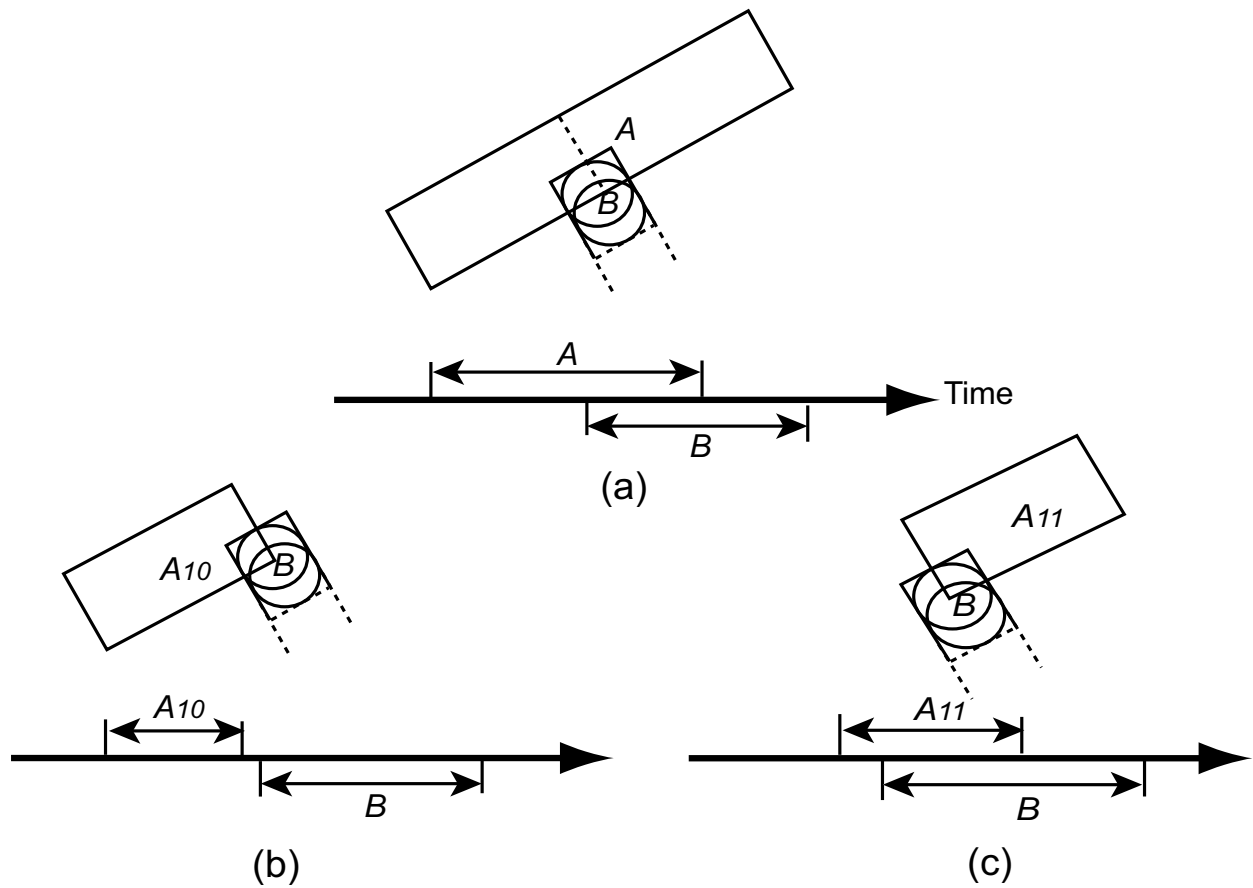
We have presented a novel bounding volume methodology, Motion Oriented Bounding Box trees, for motion capture clips that exploit a spatio-temporal hierarchy. In practice, we found that a restricted form of OBB, with one axis aligned with time, formed the most effective bound for motion data. Experimental tests confirmed that hierarchical bounds are more effective for longer motions – short motions produce trees that are too shallow. Hence, hierarchical bounds are most applicable in planning type applications where long sequences must be tested for intersection, rather than highly reactive environments in which clips are typically short. In the former situation, we saw an order of magnitude improvement in collision detection time, while in the latter case the

fastest approach is likely to be a binary search on time for overlapping samples, followed by direct comparison of sample positions.

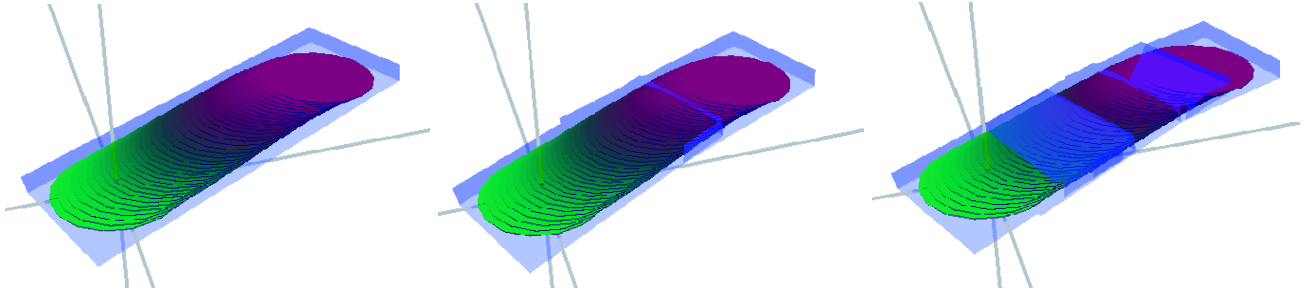
An extension we are exploring is the application of hierarchical bounds to sequences that are temporally combined at run-time, as occurs in motion graphs. Combinations of short clips obviously produce longer ones, and hence suit our technique. It is insufficient to simply test each short segment; better performance should result from combining small trees from the bottom up into larger trees. Advances in this area will result in more realistic simulation of interactive characters, and hence more engaging virtual worlds.



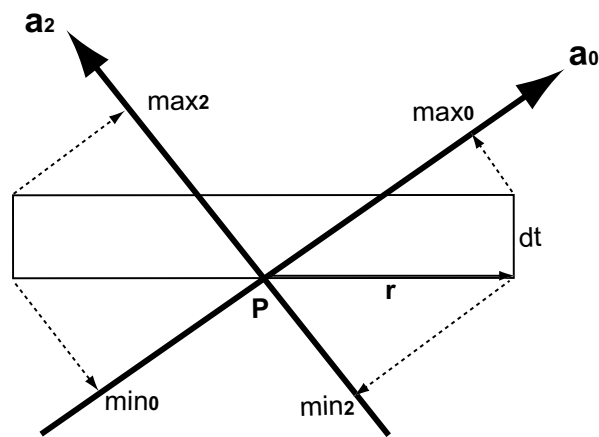
**Figure 4.4:** Testing two internal MOBB tree nodes,  $A_{00}$  and  $B_{00}$ . The relative spatial and temporal arrangement of the nodes is shown in (a). The algorithm first tests for temporal overlap, and then spatial overlap. In this case there is an intersection, so the algorithm recurses with the four combinations of child nodes. At the next level, tests (b) and (d) fail because there is no spatial overlap, test (c) fails because there is no temporal overlap (no spatial test is done), and test (e) leads to further recursion.



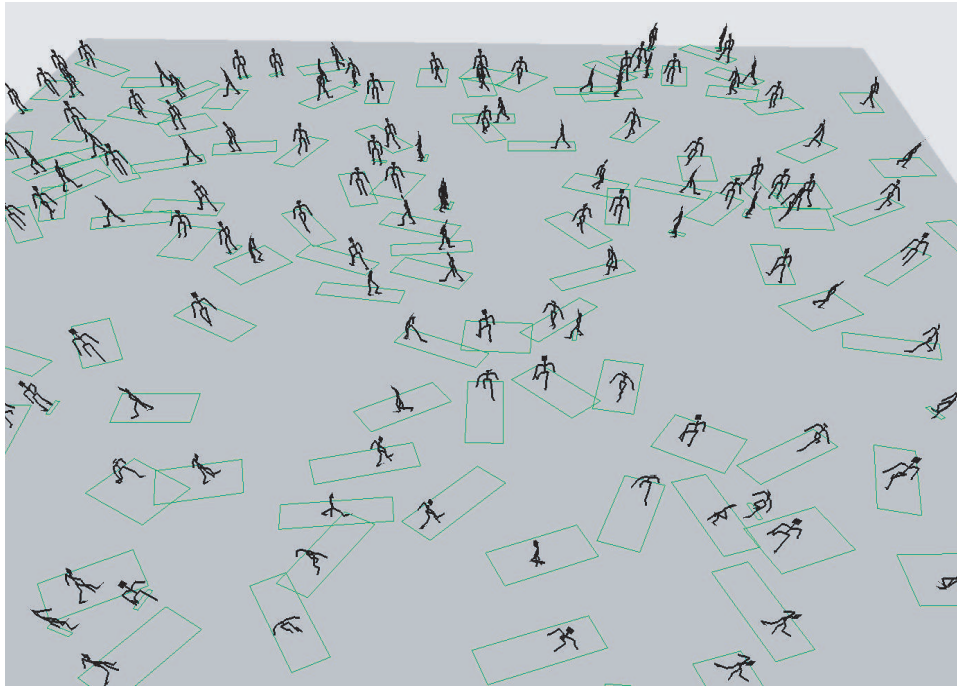
**Figure 4.5:** Testing an internal node against a leaf node. In (a), we have reached a leaf node  $B$  while at a non-root node  $A$  in the other tree. The time range of  $B$  is tested against  $A$ 's children,  $A_{10}$  and  $A_{11}$ . In one case, (b), the time interval does not overlap, so we stop with no intersection found. In the other case, (c), a temporal overlap is found so the algorithm recurses with  $A_{11}$  and  $B$ . Note that we do not perform a spatial test in case (c); experiments found out it gave no advantage.



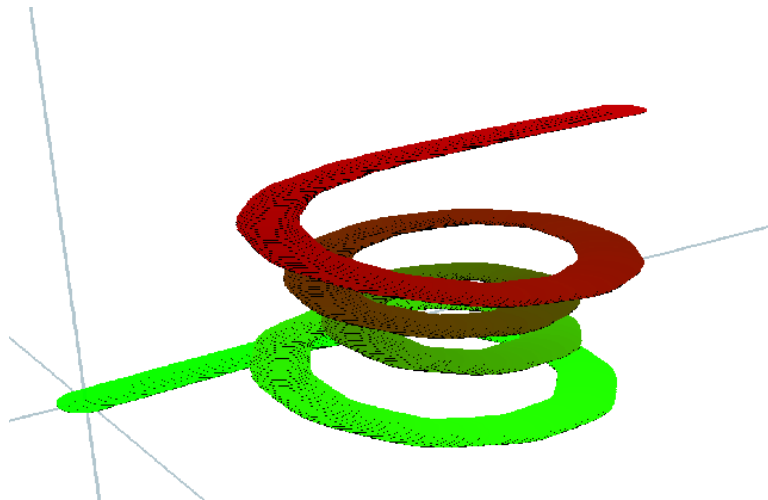
**Figure 4.6:** Three levels in an unrestricted MOBB (UMOBB) tree hierarchy. In this tree, bounding volumes are 3D OBBs in space-time, removing the restriction that one axis aligns with the time dimension. Boxes are still split evenly in the time dimension, but a 3D OBB is fitted to the samples. UMOBB trees have tighter bounds than MOBB trees, but are more expensive to test for intersection.



**Figure 4.7:** Computing the extent of a sample disk for a 3D space-time OBB. The disk has the center of its base located at the sample point,  $\mathbf{p}$ , with radius vector  $\mathbf{r}$  and height  $dt$ . We must find the minimal and maximal projections onto the axes  $\mathbf{a}_0$  and  $\mathbf{a}_2$ . Note the asymmetry in the cylinder's position.

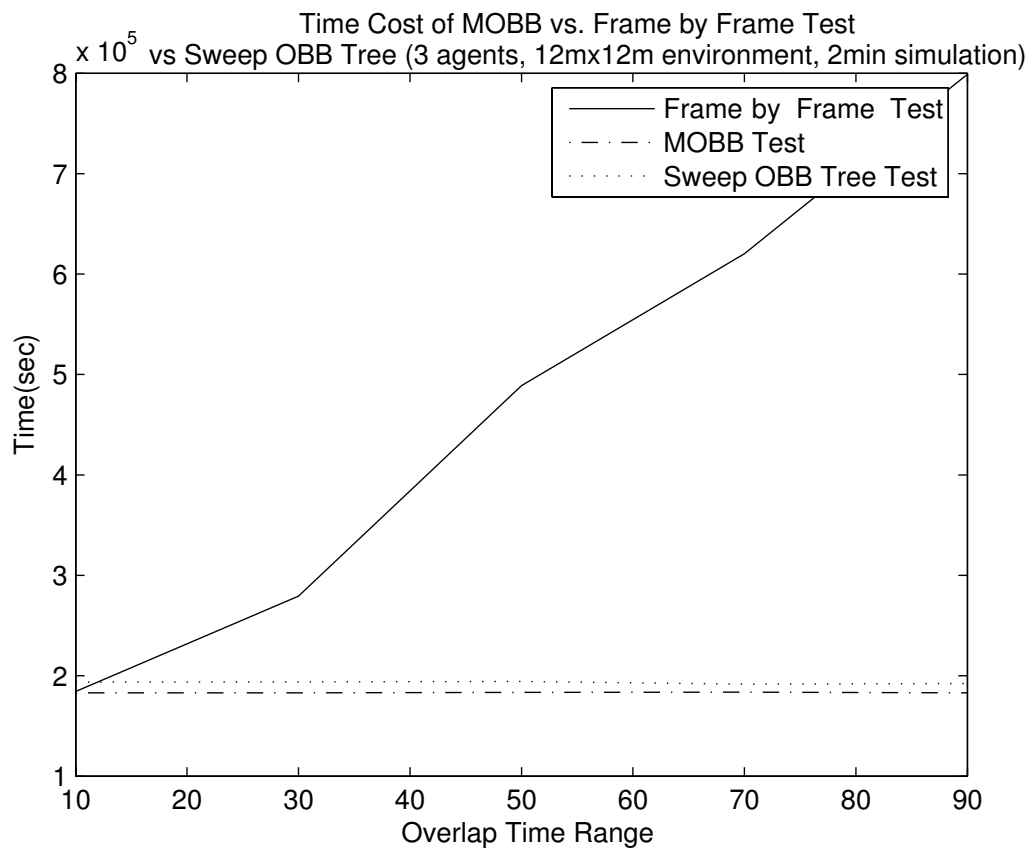


**Figure 4.8:** Snapshot of the crowd simulation used to explore the performance of MOBB trees. The environment is a  $30 \times 40$  meter rectangle containing 100 characters.



**Figure 4.9:** One of the long motions used in our experiments. The motion clip is of someone walking in circles.





**Figure 4.10:** Time cost comparison between MOBB and Frame-by-Frame and SweepOBBTree method.

# Chapter 5

## Constrained Motion Synthesis

This chapter represents the constrained motion syntheses algorithm, which corresponds to one of the two motion syntheses at the low-level parts of two-level crowd simulation framework (Figure 1.1).

### 5.1 Introduction

This chapter presents a highly efficient algorithm for synthesizing realistic goal-directed motion for large numbers of characters. We focus on the important problem of character navigation, and introduce an algorithm for creating motions that are collision-free and that precisely satisfy constraints on duration, position, orientation, and body pose. For example, we might require multiple characters to meet (e.g., face each other) at a specified position and time, or we might require a collection of characters to navigate through a building into a theater and then sit down in an array of seats. Our algorithm is capable of animating entire groups of characters at better than real-time rates, i.e., the motion for the group takes less time to generate than it takes to play. Given a higher-level control mechanism for directing behaviors [HM95, HFV00, FBT99, MT01, BC97, SCG04], our algorithm is well suited as a back end for efficiently generating detailed motion for each individual character that meets specifications on where and when desired actions should occur.

To generate high-quality motion with low computational cost, we represent the space of possible motions with a motion graph generated through the method of Gleicher et al. [GSKJ03]. This motion graph is a directed graph where each edge contains a clip of motion and nodes correspond to character poses that are shared by the end of all incoming clips and the beginning of all outgoing clips (Figure 5.1). By construction, any clip entering a node can be seamlessly connected to any clip leaving that node simply by concatenating the clips. This makes synthesis highly efficient, and it reduces the problem of creating a specific motion to finding an appropriate sequence of edges on the motion graph.

To create motions that avoid collisions and satisfy user-defined constraints, we proceed in two steps. We first use a fast path planner based on probabilistic roadmaps (PRMs) [KL94b] to navigate through complicated environments and produce motions that approximately satisfy the constraints. The result is then refined through a randomized search algorithm that yields a motion which exactly conforms to the constraints. Many existing graph-based synthesis techniques also use search algorithms to generate motions that satisfy constraints [AF02b, AFO03, GSKJ03, HGP04, KGP02, LCR\*02b, LL04]. However, because new motions can only consist of static clips from a fixed set, in general this approach cannot satisfy constraints exactly. For example, if a graph only contains clips where a character turns in 30 degree increments, then that character can never end up facing a direction that is not a multiple of 30 degrees. This can lead to fruitless, time-consuming searches for motions that cannot exist. Rather than limiting synthesis to attaching fixed clips, we modify the search to allow for a continual, gradual adjustment of the character’s position, orientation, and speed in the synthesized motion. This has two important advantages:

1. The search algorithm has greater flexibility to accurately satisfy constraints.
2. Motions can be constructed more quickly because the search need not compute *optimal* motions, but rather motions that are “close enough.”

To provide guarantees on motion quality, we restrict the amount that a synthesized motion can be adjusted. The adjustment tolerance provides a natural mechanism for striking a balance between efficient synthesis and guarantees on motion quality. However, in practice, adjustments that are small enough to be difficult to discern are sufficient to make constraint satisfaction reliable and efficient.

The remainder of this chapter begins with describing our synthesis algorithm in detail in Section 5.2 and then presents results in Section 5.3, and concludes with a brief discussion in Section 5.4.

## 5.2 Synthesizing Motion

### 5.2.1 Overview

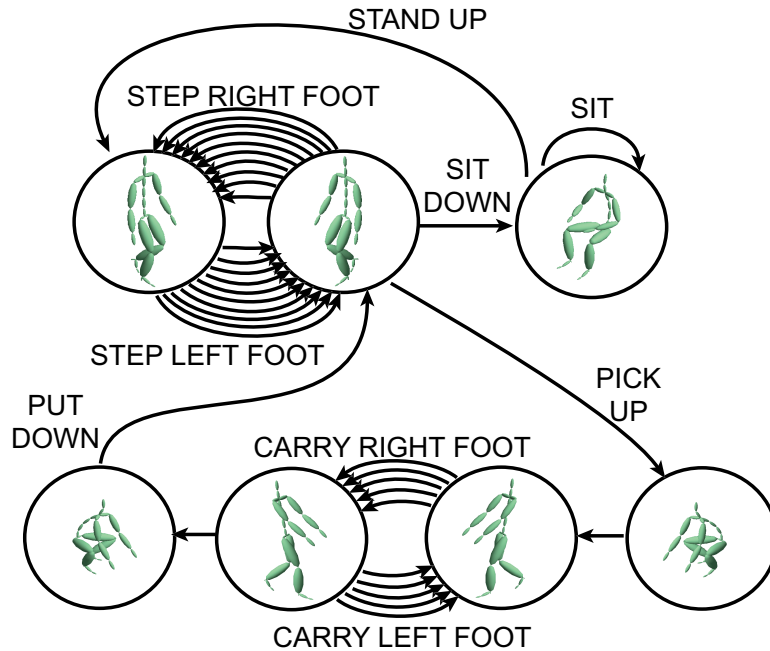
Given a group of characters and a set of constraints on each character's configuration, our goal is to synthesize motions for the individual characters such that all constraints are satisfied and no collision occurs. Each constraint can specify a character pose, a position  $\mathbf{p}$  and orientation  $\theta$  for this pose, and a time interval  $[t^a, t^b]$  in which this configuration must be obtained (possibly  $t^a = t^b$ ). The time constraint can either be absolute (e.g., the character must arrive at a spot 3 seconds from now) or relative to another character's motion (e.g., the character must arrive at a spot within 1 sec before or after another character). Not all of these components need to be specified — for example, one might require a character to move to a particular spot without specifying a pose, orientation, or time interval.

To avoid inter-character collisions, individual characters are processed sequentially, with characters whose motions have already been planned treated as moving obstacles. This limits the size of the search space and as a result is considerably more efficient and scalable than processing all characters simultaneously. While in principle sequential processing may result in artificially

unsatisfiable constraints, typically there are many possible motions that satisfy a given set of constraints, and we have found that in practice sequential processing does not prevent sophisticated group animations from being generated. The specific processing order is arbitrary except insofar as is necessary to satisfy timing relationships — for example, if character A must arrive somewhere 1s after character B, then B is processed first. We assume that the timing constraints have no circular dependencies, so a feasible ordering always exists.

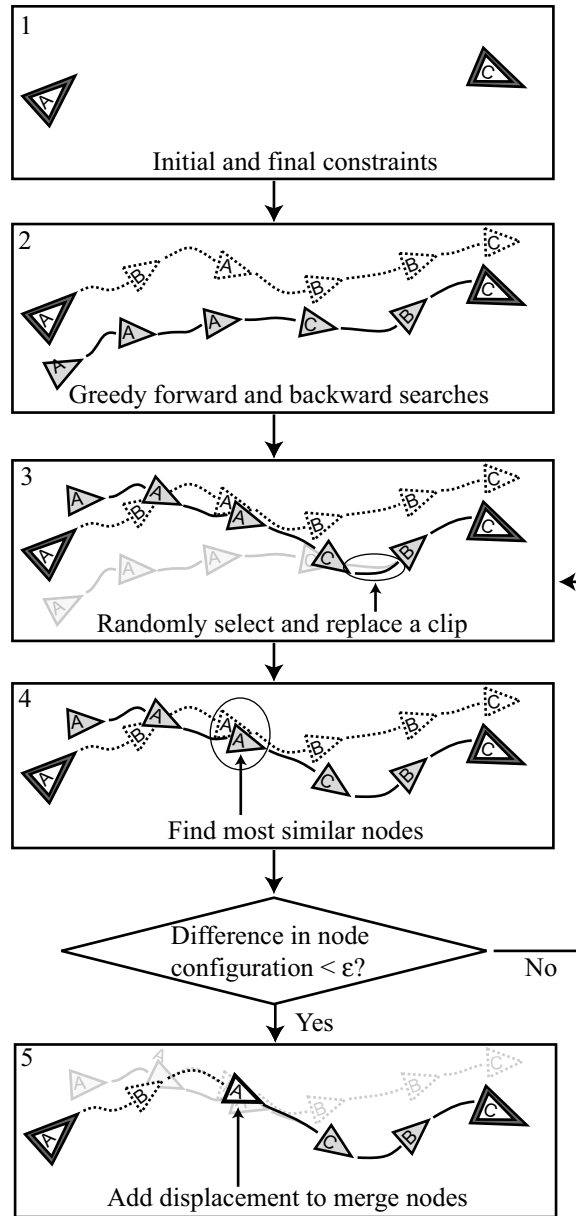
We represent the actions available to a character with a motion graph (Figure 5.1). Each edge corresponds to a clip of motion, and any sequence of connected edges yields a seamless motion composed of the corresponding sequence of clips. Every pose that the character can attain is contained within the motion graph, and so we require each constraint pose to correspond with the pose at some node  $\mathcal{N}$ . Traditional synthesis methods based on motion graphs [AF02b, KGP02, LCR\*02b] are inherently discrete in that they search for a sequence of available clips that meets user-defined criteria. As noted in Section 5.1, this precludes constraints on continuous properties (such as position, orientation, and duration) from being exactly satisfied, and finding a clip sequence with minimal deviation can require an expensive search. We instead allow the clips from the motion graph to be continuously adjusted to alter a character’s position, orientation, and speed. This strategy allows the constraints defined above to be exactly satisfied, and it yields shorter search times since a raw sequence of clips need only sufficiently reduce constraint deviation, rather than minimize it.

To produce a motion that satisfies a sequence of configuration constraints, it is sufficient to consider the problem of constructing segments of motion that start in a specified configuration  $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$  and end in a specified configuration  $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$  within a given time interval  $[t^a, t^b]$ . The full constraint sequence can then be satisfied by iteratively generating motion that travels from the current configuration to the next configuration. To satisfy the constraints at an iteration, we use a fast approximate planner to construct motions that navigate through the environment and

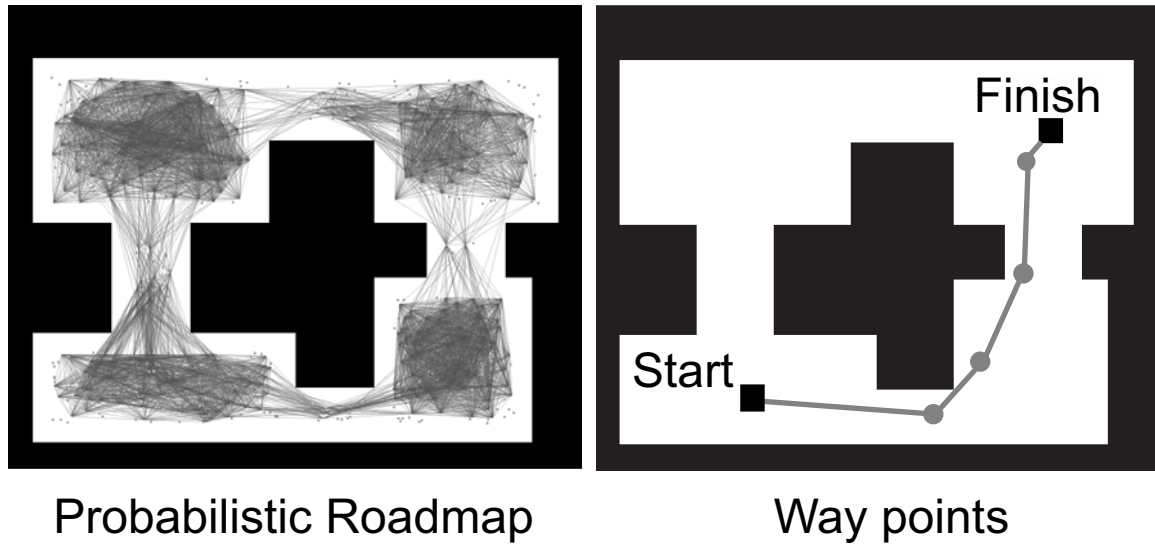


**Figure 5.1:** An example motion graph. Edges are motion clips and nodes indicate frames where clips have the same joint orientations and velocities.

then refine the result to produce a motion that exactly satisfies the constraints. More specifically, the algorithm first constructs two “seed” motions: a *forward* motion  $M_f$  that starts at  $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$  and ends near (but in general not exactly at)  $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$ , and a *backward* motion  $M_b$  that ends precisely in  $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$  and starts approximately at  $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$ . A randomized search procedure then adjusts these motions such that 1) they can be connected with adjustments to position, orientation, and timing that are below a user-defined threshold and 2) the resulting motion satisfies all constraints. This process is illustrated in Figure 5.2. The remainder of this section provides details on how  $M_f$  and  $M_b$  are created (Section 5.2.2) and then how they are adjusted and connected (Section 5.2.3).



**Figure 5.2:** An individual character's motion is generated by computing motions that satisfy, respectively, constraints on the initial and final configuration (steps 1 and 2) and then iteratively adjusting these motions so they can be seamlessly connected while satisfying all constraints (steps 3 – 5).



**Figure 5.3:** **Left:** An example probabilistic roadmap. **Right:** A series of way points that connect a pair of positions.

## 5.2.2 Creating the Seed Motions

For simplicity, we only discuss the construction of  $M_f$ ;  $M_b$  is handled identically, except time flows in reverse. As in previous work [PLS03], we start by building a probabilistic roadmap (Figure 5.3) to aid in navigating through the environment. Nodes in the roadmap are created by randomly sampling positions on the ground that are outside of obstacles, and edges are created between nodes that are within a threshold distance of each other and are mutually visible (i.e., the connecting line segment does not intersect any obstacles). Given desired starting and ending configurations  $(\mathcal{P}_s, \theta_s, \mathbf{p}_s)$  and  $(\mathcal{P}_e, \theta_e, \mathbf{p}_e)$ , nodes are added to the road map at  $\mathbf{p}_s$  and  $\mathbf{p}_e$  and edges are added to all visible neighbors. A shortest path from  $\mathbf{p}_s$  to  $\mathbf{p}_e$  is then found with Dijkstra's algorithm, resulting in a sequence of way points  $w_1, \dots, w_n$  (Figure 5.3).

Once a sequence of way points is determined, a fast greedy planner guides the character through successive way points such that it travels from the initial configuration toward the final configuration. This planner continues until a local error measure does not decrease, and hence while it is



guaranteed to terminate, in general it will not produce optimal motion. This is acceptable because only a rough initial motion is needed for the randomized search, and a fast approximate planner is preferable to a slower planner that produces optimal (but still necessarily inexact) results.

The planner proceeds as follows. First, assume that the current target way point is  $w_i$ , with  $i < n$ . The planner iteratively selects the edge in the motion graph that brings the character closest to  $w_i$  without incurring collisions with the environment or higher-priority characters. This edge is required to be within the subgraph corresponding to the character's current locomotion state — for example, in Figure 5.1, a character who has just picked up a box would be required to use one of the “carry” edges. This subgraph is determined from annotations associated with the originally data, which may be added semi-automatically [AFO03] as a preprocess. Collisions are detected by placing minimum bounding cylinders around each character. Because the set of all possible character poses is encoded in the motion graph, these bounding cylinders can be precomputed for greater efficiency. If all possible edges increase the distance to  $w_i$ , then  $w_i$  is set to the current way point and the planner attempts instead to reach  $w_{i+1}$ .

When travelling to the final way point  $w_n = p_e$ , the planner must also account for possible constraints on target orientation  $\theta_e$  and ending pose  $\mathcal{P}_e$ . The latter constraint reduces to requiring the motion to terminate in the node  $N_{\mathcal{P}_e}$  that corresponds to  $\mathcal{P}_e$ . A similar greedy algorithm is used, except 1) the next edge is selected so as to minimize a weighted sum of the position and orientation errors, rather than just position errors, and 2) whenever the current motion terminates at  $N_{\mathcal{P}_e}$ , it is stored in a temporary variable  $M_{best}$ . Whenever the error stops decreasing, the current  $M_{best}$  is returned. If no value for  $M_{best}$  exists (because no motion terminating in  $N_{\mathcal{P}_e}$  has been encountered), then instead Dijkstra's algorithm is used to find the path in the motion graph to  $N_{\mathcal{P}_e}$  containing the fewest edges. If multiple shortest paths exist, then ties are broken by greedily selecting edges that minimize position/orientation error. Finally, if the orientation is unconstrained,

then only position error is considered, and if the final pose is unconstrained, then the planner simply adds edges until the position/orientation error stops decreasing.

Note that the planner does not consider time constraints. These constraints are addressed in the next stage of the synthesis process.

### 5.2.3 Adjusting and Merging the Seed Motions

The seed motions  $M_f$  and  $M_b$  satisfy, respectively, the constraints on the character's initial and final configuration. Our goal is to merge them into a single motion that satisfies both of these configuration constraints and, if specified, has a duration within the desired interval  $[t^a, t^b]$ . We start by finding frames in  $M_f$  and  $M_b$  where the character is in the same pose and where the position and orientation of this pose are similar. Displacement maps are then added so the position and orientation are identical. Finally, the adjusted motions are spliced together at these frames to form a seamless new motion that begins and ends in the desired configurations, and the speed of this motion is altered to conform to the time constraints. While this algorithm guarantees that the constraints are satisfied, the result may look unrealistic if overly large changes are made. Our strategy is to use a randomized search algorithm to perturb  $M_f$  and  $M_b$  such that they are sufficiently similar at a pair frames and of sufficient duration that the necessary adjustments are below a user-controllable tolerance. Figure 5.2 graphically depicts this process. The remainder of this section explains our algorithm in greater detail.

Let  $M_f$  and  $M_b$  be composed, respectively, of  $n_f$  and  $n_b$  clips from the motion graph. The  $i^{th}$  clip of  $M_f$  is represented by a tuple  $\{t_{f,i}, \mathcal{I}_{f,i}, \mathbf{p}_{f,i}, \theta_{f,i}\}$  containing the clip's duration  $t_{f,i}$ , the index  $\mathcal{I}_{f,i}$  of its starting node in the motion graph, and the position  $\mathbf{p}_{f,i}$  and orientation  $\theta_{f,i}$  of the pose associated with this node. The  $j^{th}$  clip of  $M_b$  is represented similarly. We consider joining  $M_f$  and  $M_b$  at any point where they share a node from the motion graph. Let the subsection of a motion  $M$  consisting of the  $r_1^{th}$  clip through the  $r_2^{th}$  clip be  $M[r_1, r_2]$ , and consider the motion

formed by concatenating  $\mathbf{M}_b[j, n_b]$  onto the end of  $\mathbf{M}_f[1, i]$ . We define the cost  $C_{\mathbf{M}_f, \mathbf{M}_b}(i, j)$  of creating this motion as a sum of position, orientation, and time errors, normalized by the motion's duration:

$$C_{\mathbf{M}_f, \mathbf{M}_b}(i, j) = \frac{1}{N(i, j)} (E_{\mathbf{p}}(i, j) + \alpha_1 E_{\theta}(i, j) + \alpha_2 E_t(i, j)), \quad (5.1)$$

where

- $N(i, j)$  is the total duration of  $\mathbf{M}_f[1, i]$  and  $\mathbf{M}_b[j, n_b]$ , computed as  $\sum_{k=1}^i t_{f,k} + \sum_{k=j}^{n_b} t_{b,k}$ .
- $E_{\mathbf{p}}(i, j)$  is the distance between the end of  $\mathbf{M}_f[1, i]$  and the start of  $\mathbf{M}_b[j, n_b]$ , computed as  $\|\mathbf{p}_{f,i+1} - \mathbf{p}_{b,j}\|$ .
- $E_{\theta}(i, j)$  is the orientation difference, computed as  $|\theta_{f,i+1} - \theta_{b,j}|$ , with numerical values assigned to  $\theta_{f,i+1}$  and  $\theta_{b,j}$  such that this error is no greater than  $180^\circ$ .
- $E_t(i, j)$  is the time error. If the constraint time interval is  $[t^a, t^b]$ , then  $E_t(i, j) = 0$  if  $N(i, j) \in [t^a, t^b]$  and otherwise  $E_t(i, j) = \min(|N(i, j) - t^a|, |N(i, j) - t^b|)$ .
- $\alpha_1$  and  $\alpha_2$  are scaling factors to relate the different error measures. In our implementation,  $1\text{cm} \approx 1^\circ \approx \frac{1}{30}\text{s}$ .

The cost  $C'(\mathbf{M}_f, \mathbf{M}_b)$  of connecting  $\mathbf{M}_f$  and  $\mathbf{M}_b$  is defined as the minimum value of  $C_{\mathbf{M}_f, \mathbf{M}_b}(i, j)$  over all indices where the terminating node of  $\mathbf{M}_f[1, i]$  is the same as the starting node of  $\mathbf{M}_b[j, n_b]$ .

$$C'(\mathbf{M}_f, \mathbf{M}_b) = \min_{i,j:\mathcal{I}_{f,i+1}=\mathcal{I}_{b,j}} C_{\mathbf{M}_f, \mathbf{M}_b}(i, j) \quad (5.2)$$

If a pose constraint exists on the start and/or end of the desired motion (as opposed to, for example, just a position constraint), then  $\mathbf{M}_f$  and  $\mathbf{M}_b$  will always share at least one node. Otherwise, it is possible for them to share no nodes, in which case  $C'(\mathbf{M}_f, \mathbf{M}_b) = \infty$ .

Intuitively,  $C'(\mathbf{M}_f, \mathbf{M}_b)$  represents the amount of per-frame adjustment needed to seamlessly splice  $\mathbf{M}_f$  and  $\mathbf{M}_b$  and satisfy the time constraints. To preserve the realism of the original motion

data, we require this to be below a user-defined threshold  $\epsilon$  (optionally, one might also limit the total accumulated adjustment, but in our experience it is sufficient to consider the per-frame adjustment  $C'(\mathbf{M}_f, \mathbf{M}_b)$ ; see Section 5.3). If  $C'(\mathbf{M}_f, \mathbf{M}_b) > \epsilon$ , then  $\mathbf{M}_f$  and  $\mathbf{M}_b$  are perturbed through a randomized search algorithm to produce a new pair of motions with a below-threshold cost. The search proceeds by generating  $n_m$  perturbed motion pairs, checking as each pair is generated whether the individual motions are collision-free and whether the splicing cost (Equation 5.2) is below  $\epsilon$ . If so, the search returns this motion pair. Otherwise, the  $k_m$  lowest-cost pairs are retained and the search continues. If the search fails to find a below-threshold motion pair after a user-defined maximum number of iterations  $N_{max}$ , then a warning is issued and the current lowest cost pair is returned.

During the search, perturbed motion pairs are generated as follows. First, one of the  $k_m$  motion pairs from the previous search iteration is selected at random, unless it is the first iteration, in which case only the original pair  $(\mathbf{M}_f, \mathbf{M}_b)$  is available. Next, one of the two motions in this pair is selected. A clip  $\mathbf{C}$  is chosen at random from this motion and replaced with a new clip  $\mathbf{C}'$ . In order to ensure that  $\mathbf{C}'$  joins seamlessly with the rest of the motion, it is required to originate and terminate at the same nodes in the motion graph as  $\mathbf{C}$  (this also means that the perturbed motion begins and ends in the same poses, so pose constraints automatically remain satisfied). Because the motion graph is constructed so as to have many more edges than nodes, in general there will be many possible replacement clips. Let the change in the character’s position and orientation in  $\mathbf{C}$  be  $\delta\mathbf{p}_\mathbf{C}$  and  $\delta\theta_\mathbf{C}$ , and let  $\mathbf{C}$ ’s duration be  $\delta t_\mathbf{C}$ . The actual replacement clip  $\mathbf{C}'$  is chosen with a probability inversely proportional to its “distance” from  $\mathbf{C}$ , in terms of the relative change in position, orientation, and time:

$$\|\Delta\mathbf{p}_\mathbf{C} - \Delta\mathbf{p}_{\mathbf{C}'}\| + \alpha_1|\Delta\theta_\mathbf{C} - \Delta\theta_{\mathbf{C}'}| + \alpha_2|\Delta t_\mathbf{C} - \Delta t_{\mathbf{C}'}|$$

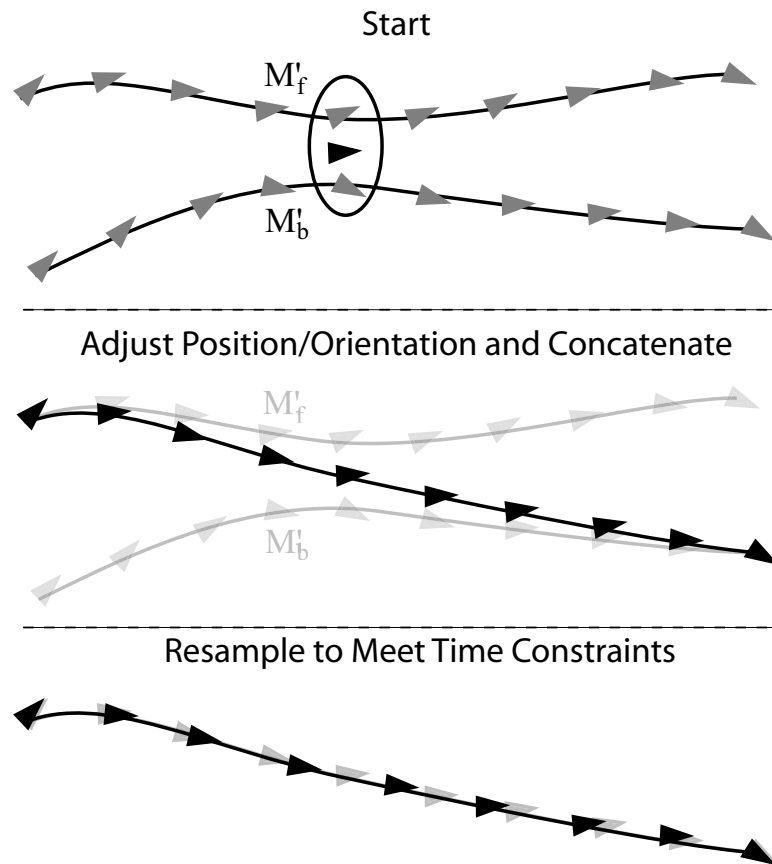
Once suitably perturbed forward and backward motions  $\mathbf{M}'_f$  and  $\mathbf{M}'_b$  are found, they are joined at the nodes calculated in Equation 5.2; see Figure 5.4. Let  $\mathbf{M}'_f[1, i]$  and  $\mathbf{M}'_b[j, n'_b]$  be the motion segments that will be joined, and let the difference in the final position and orientation of  $\mathbf{M}'_f[1, i]$  and the initial position and orientation of  $\mathbf{M}'_b[j, n'_b]$  be, respectively,  $\delta\mathbf{p}$  and  $\delta\theta$ . Also, let  $N_f$  be the duration of  $\mathbf{M}'_f[1, i]$ ,  $N_b$  be the duration of  $\mathbf{M}'_b[j, n'_b]$ , and  $\delta t$  be the smallest amount that must be added to  $N_f + N_b$  such that  $(N_f + N_b + \delta t)$  is inside the constraint time interval  $[t^a, t^b]$ . Lastly, define  $\lambda_1 = \frac{N_f}{N_f + N_b}$  and  $\lambda_2 = \frac{N_b}{N_f + N_b}$ . The final motion is formed as follows:

1. The  $k^{th}$  frame of  $\mathbf{M}'_f[1, i]$  has its position and orientation adjusted by  $\frac{k-1}{N_f-1}\lambda_1\delta\mathbf{p}$  and  $\frac{k-1}{N_f-1}\lambda_1\delta\theta$ .
2. The  $k^{th}$  frame of  $\mathbf{M}'_b[j, n'_b]$  has its position and orientation adjusted by  $-\frac{k-1}{N_b-1}\lambda_2\delta\mathbf{p}$  and  $-\frac{k-1}{N_b-1}\lambda_2\delta\theta$ .
3. The adjusted  $\mathbf{M}'_f[1, i]$  and  $\mathbf{M}'_b[j, n'_b]$  are concatenated.
4. The result is resampled so its duration is  $(N_f + N_b + \delta t)$ .

By construction, the result is continuous and satisfies all constraints. A final test for collisions is made on this new motion and, if the test fails, then the motion is discarded and the search algorithm continues from where it left off.

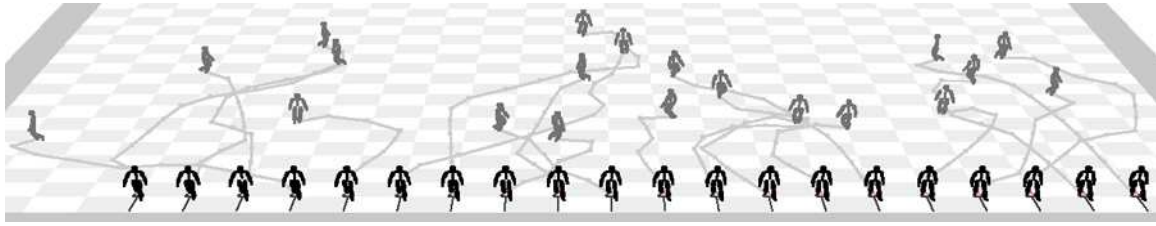
## 5.3 Validation

Given constraints for individual, the motion synthesis that generates motions satisfying the constraints exactly is the most important feature for controllability. This allows us to set various constraints for character at any time as long as the constraints are legal (i.g, if the position constraint is not in the obstacles). Fast synthesizing motions validates the efficiency demands as well because it allows to synthesize a lot of characters simultaneously.

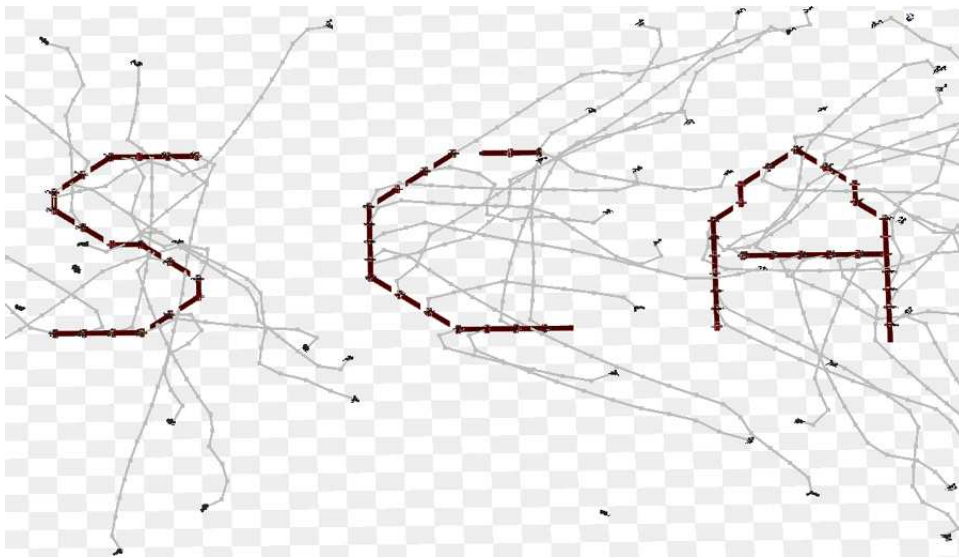


**Figure 5.4:**  $M'_f$  and  $M'_b$  are joined at the nodes calculated in Equation 5.2 by adding displacements that compensate for differences in position and orientation, and the result is then resampled to meet the time constraints.

For validating of satisfying demands, we tested our algorithm using the motion graph shown in Figure 5.1, which allowed characters to walk in various directions; sit down into and get up from a chair; and pick up a box, carry it in various directions, and put it down. Altogether, the motion graph contained 50s of motion data sampled at 30Hz. In our experiments, we set  $\epsilon = 2cm/s = 2^\circ/s = \frac{1}{15}s/s$ , and the parameters for the search algorithm (see Section 5.2.3) were  $n_m = 200$ ,  $k_m = 30$ , and  $N_{max} = 1000$ ; for these values the search algorithm was always able to find a motion pair that was within the adjustment tolerance and avoided collisions. We generated several animations that required a large number of characters to perform certain tasks at specified



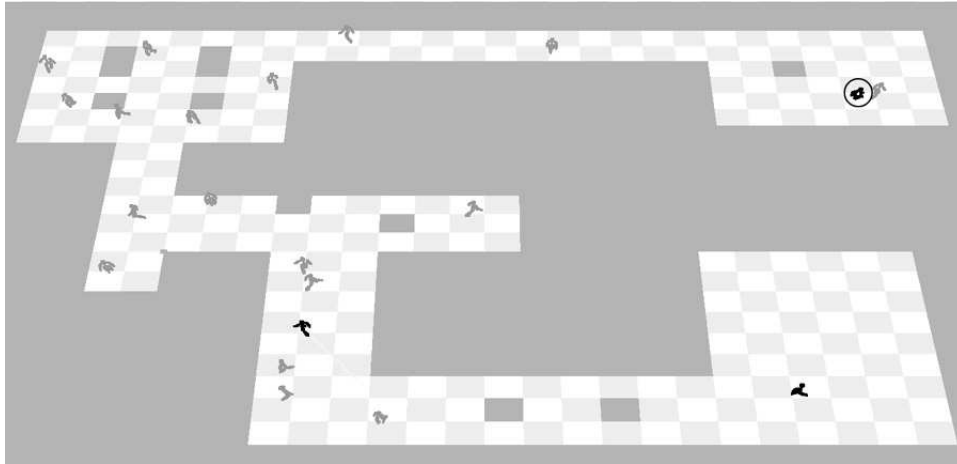
**Figure 5.5: Experiment 1:** 20 characters (grey) are required to arrive simultaneously in specified configurations (black).



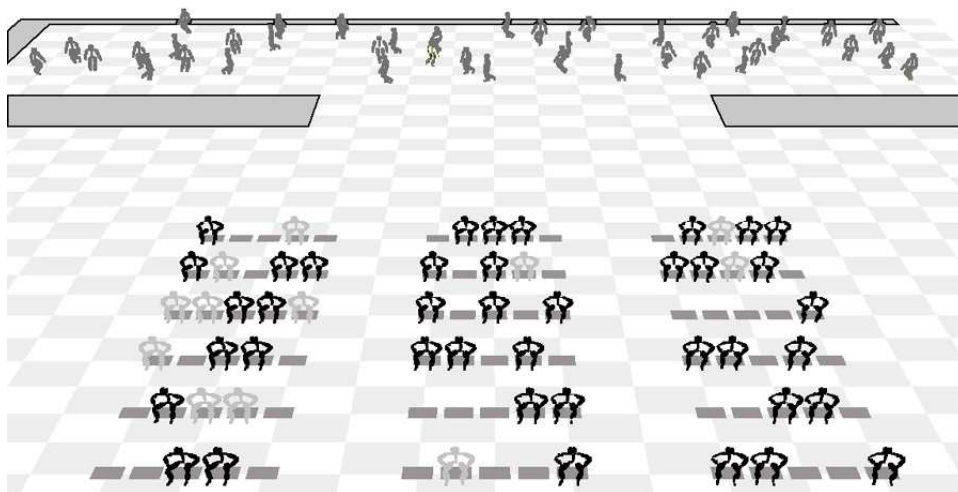
**Figure 5.6: Experiment 2:** 70 characters are required to arrive at a set of positions in which they spell out SCA.

positions, locations, and times; see Figures 5.5 through 5.9, Table 5.1 and Table 5.2 for a summary of results.

In order to test our method on varying numbers of characters and densities of characters, for Experiment 6 we created a contrived scenario where a number of characters are placed on a uniform grid and are given target positions on a translation of this grid. The target for each character is chosen such that each character's path is of approximately the same length, but that the characters must interact in order to meet their goals, as illustrated in Figure 5.10. This experiment allowed us to test examples with large numbers of characters. When the spacing between characters is



**Figure 5.7: Experiment 3:** The character in black must start at the lower right, navigate through a set of rooms filled with obstacles and other characters, and lift a box in the room at the upper right. The location of the box is circled.



**Figure 5.8: Experiment 4:** 40 characters start in a theater lobby (top) and take their seats (bottom). Dark sitting characters represent target poses, and light sitting characters are characters that are already sitting down (e.g., obstacles).

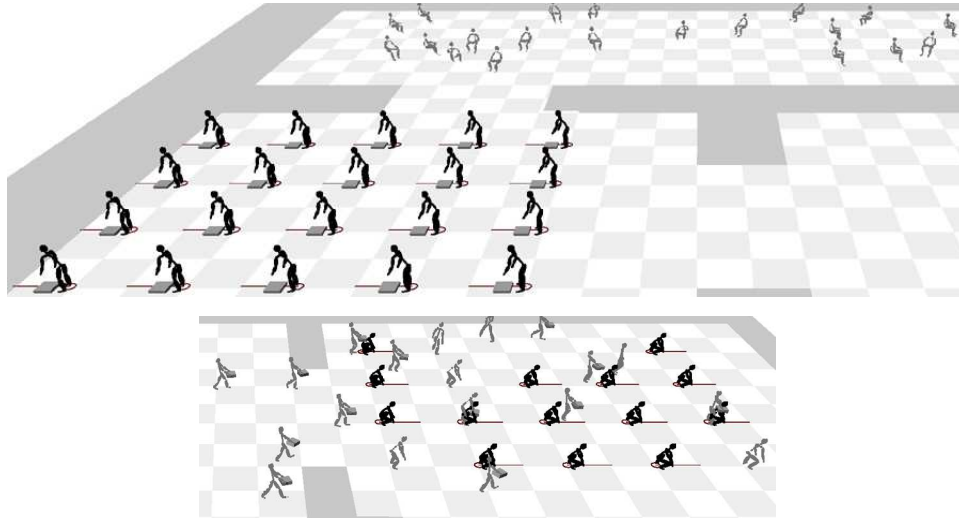


Example	# Characters	Duration	Synthesis Time	
			Average	Total
1	20	14.8s	0.21s	4.2s
2	70	21.3s	0.18s	12.6s
3	1	83.3s	0.01s	0.01s
4	40	30.2s	0.35s	14.0s
5	20	23.6s	0.15s	3.0s
6a	300	25.3s	0.034s	0.86s
6b	500	25.6s	0.035s	0.90s

**Table 5.1:** Information relating to the animations shown from Figure 5.5 to Figure 5.9. The second column states the total number of characters that were animated, the third column states the average duration of each character’s motion, the fourth column states the average amount of time needed to synthesize the motion for a character, and the final column states the total amount of time needed to synthesize all motions for all characters. All experiments were performed on a PC with a 3.0Ghz processor and 1GB main memory. For Experiment 6 (described below) we report results for representative trials with (6a) 300 characters and (6b) 500 characters.

Example	Density of Crowds	Physical complexity of Environment
1	5.6	0
2	5.3	5.3
3	5.6	0
4	10.5	27.2
5	8.2	8.2
6a	3.4	3.4
6b	5.6	5.6

**Table 5.2:** The density of crowds and the physical complexity of environments of experiment 1-6.

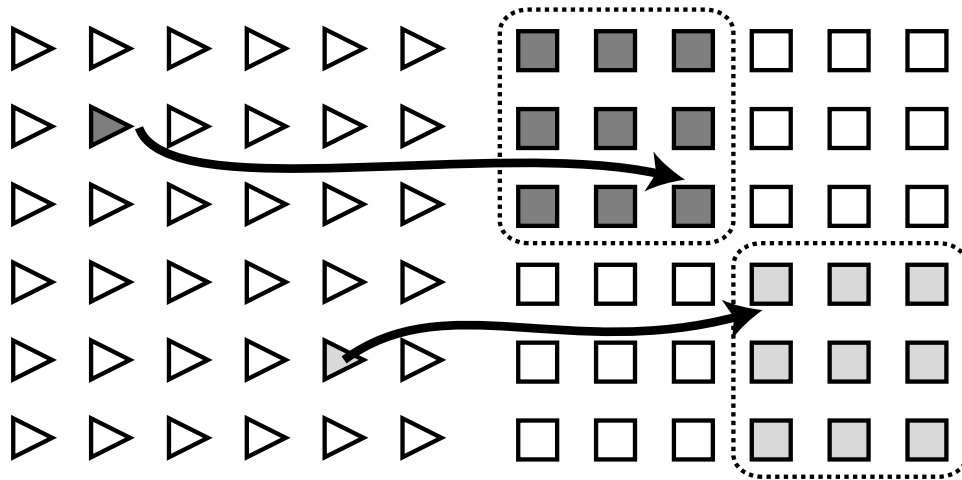


**Figure 5.9: Experiment 5:** 20 characters, initially sitting, must pick up a designated box in the second room, place it in a designated position, and then return to their original seat. The top image shows initial positions (grey) and constraints poses (black). The lower image shows characters moving towards placement goals, with constraint poses again shown in black.

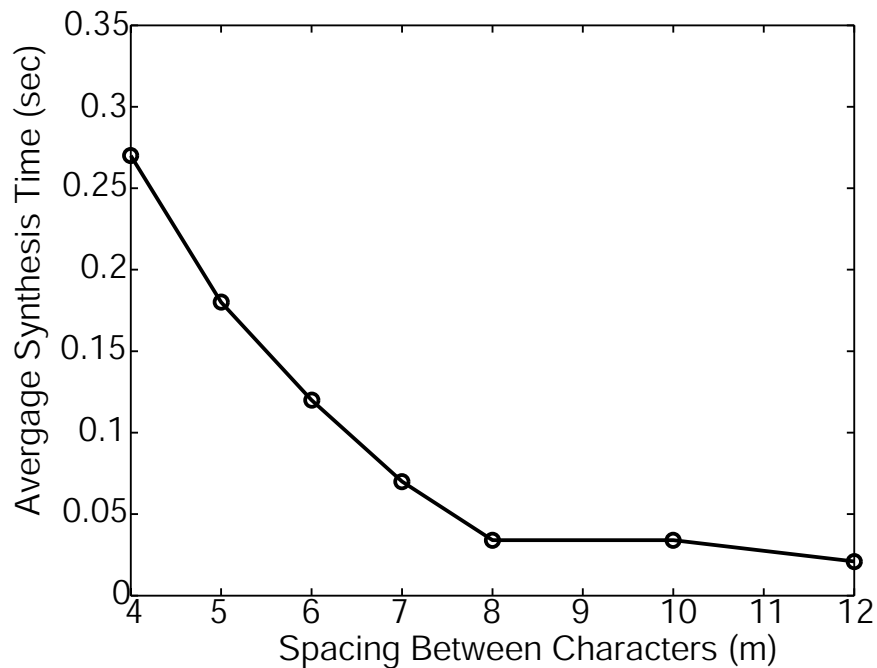
sufficiently large that collision avoidance does not unduly restrict movement, our algorithm can synthesize the motions for hundreds of characters faster than real time.

The speed of our algorithm depends upon the complexity of the scenario that is to be animated. In particular, higher-density groups of characters (or, equivalently, more cluttered environments) require additional time in order to avoid collision, both because collision detection is more time consuming and because the search will typically explore a greater number of paths before finding one that avoids collisions. To illustrate this, for Experiment 6 we created a series of scenarios wherein motion was synthesized for a group of 300 characters with different inter-characters spacings. Figure 5.11 shows the average amount of time needed to plan the motion for an individual character as a function of the grid spacing. At smaller grid spacings, the average synthesis time increases because more paths are rejected during the random search due to collisions.

Different values of  $\epsilon$  provide different tradeoffs between motion quality and synthesis speed — larger values permit greater deviation from the raw graph-generated motions, but also allow the



**Figure 5.10:** The scenario tested in Experiment 6. Characters begin in grid formation (left) and are each assigned a target on a translated version of this grid (right). Targets are chosen to keep each character's path approximately the same length. Specifically, if a character begins at position  $(i, j)$  on the original grid, then it will be assigned a randomly perturbed position on the target grid. For example, the character indicated by the dark grey triangle is assigned one of the target positions indicated by a dark grey square.

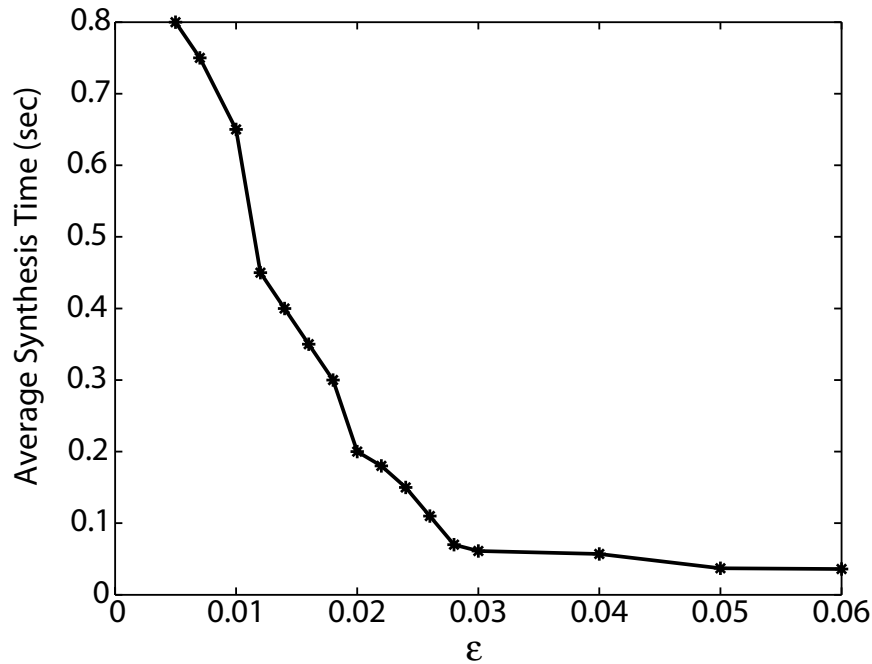


**Figure 5.11:** The average time per character needed to generate motion for 300 characters in the scenario of Experiment 6, as a function of the spacing between characters.

search to terminate more quickly because a larger range of motion pairs will be able to satisfy the constraints. Figure 5.12 shows the time needed to synthesize the motion for all 70 characters in scenario 2 at different values of  $\epsilon$ ; the shape of this graph was similar in the other scenarios. At very small tolerances, relatively few motions can be made to satisfy the constraints, but at higher tolerances a wider variety exists and the search hence can terminate sooner. However, eventually the speed benefit of increasing the tolerance decreases, because sufficient flexibility already exists to quickly find motions that satisfy the constraints, and the synthesis time becomes dominated by other concerns like collision detection.

For collision detection between two paths, we compared the MOBB tree and simple frame-by-frame technique in a simple environment. The size of environment was 30m x 40m, and we randomly distributed characters in the environment and made each character has a random initial and target constraint. As we increased the number of characters, we checked the average collision detection time among characters. The result is shown in Figure 5.13.

We limit the amount of per-frame adjustment to  $\epsilon$  based on the intuition that longer motions can tolerate larger overall adjustments if they are worked in gradually. However, if the total accumulated adjustment exceeds a certain amount, this approach breaks down. For example, a 180° change in orientation will yield unrealistic results now matter how gradually it is introduced, because eventually the character will face opposite the direction of travel. This can trivially be avoided by placing a second limit  $\epsilon'$  on the total allowable adjustment. However, in our experiments we found this to be unnecessary, because the amount that motions needed to be altered to meet the constraints was effectively independent of the duration of the motions.

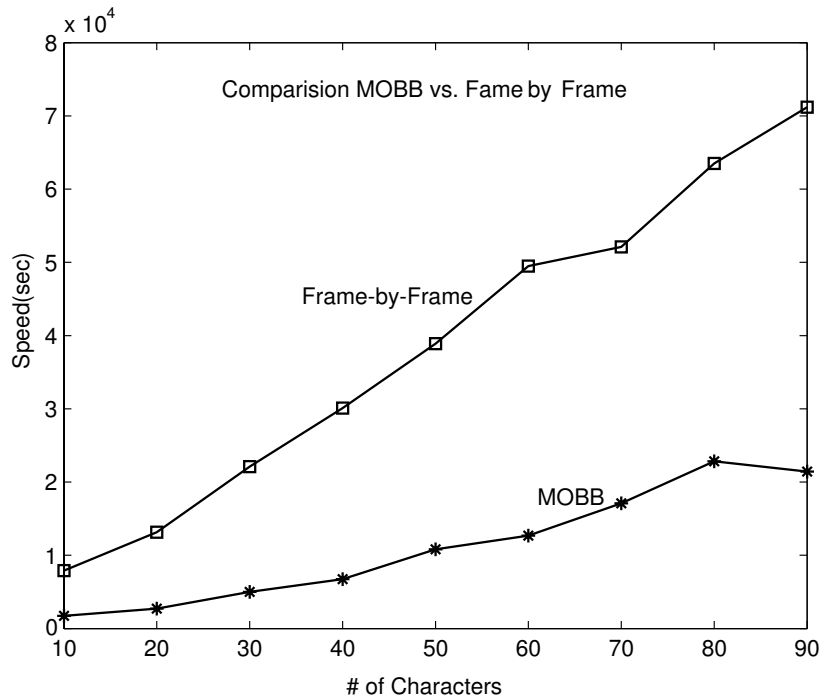


**Figure 5.12:** The average time per character needed to generate motion for 70 characters in a typical scenario, as a function of the allowable amount of motion adjustment  $\epsilon$ .

## 5.4 Discussion

This chapter has presented an efficient algorithm for generating realistic goal-directed motion for large numbers of characters. Our algorithm creates collision-free motion that precisely satisfies constraints on pose, position, orientation, and duration. A fast PRM-based path planner is used to construct rough motions that navigate through complex environments, and this is then refined through a randomized search over a motion graph that explicitly incorporates continual, gradual adjustments to a character’s position, orientation, and speed. The flexibility provided by these adjustments allows us both to precisely satisfy constraints and to reduce the time needed to construct desired motions.

We have focused on the problem of character navigation, where the main technical challenge is to guide a character through an environment such that it ends up in a particular configuration.



**Figure 5.13:** Comparison between MOBB trees and frame-by-frame technique in time cost for collision detection.

For stationary tasks, such as manipulating objects or gesticulating during conversation, we assume that a corresponding clip or known sequence of clips already exists. Also, to be able to reliably satisfy constraints without destroying motion quality, our algorithm relies on constraints being sufficiently sparse that the connecting motions can be meaningfully adjusted. For example, if the character were constrained to be in a specific pose every second, then it is likely that satisfying all of these constraints would require a very large value for  $\epsilon$ , which would in turn probably result in unrealistic motion.

Our motion graph search algorithm does not generate optimal motion in the sense of minimizing deviation from the constraints. Instead, it finds motions that can be made to exactly satisfy constraints by being adjusted within a user-specified tolerance, and all motions within this tolerance are considered equally suitable. In principle, this involves trading off motion quality for

computational speed, but in practice even a modest tolerance can significantly reduce searching times without appreciably altering motion quality. Our search algorithm also assumes that the constraints fully specify what a motion should do, and that any motion that satisfies the constraints is acceptable. This can sometimes lead to artifacts where characters, despite having smooth motion, exhibit unusual behavior. For example, a character might not take the most direct path towards a route, and indeed will intentionally wander if the time constraints require a lengthy motion. This is a common limitation that is shared by existing graph-based synthesis algorithms which rely on constrained minimization [AF02b, KGP02, LCR\*02b] — if the objective function and constraints does not completely describe the desired properties of a motion, then undesirable behavior is inevitable. For scenes with many characters, however, we have found this to be less of a problem because a viewer’s attention typically is not limited to a single individual, and hence longer-term behavioral oddities are less noticeable.

The constraints used by our synthesis algorithm are produced by an external process, and it is the responsibility of this process to ensure that these constraints are achievable. For example, it should not require two characters to be at the exact same place at the same instant of time.

Our current methods are offline and hence not directly applicable to applications like games. However, for more interactive assessment of the results the search can be broken into stages, first generating motion that satisfies the initial configuration/time constraint of each character, then the next constraint for each character, and so on. A more general extension to online applications is left for future work. Also, a straightforward extension of this work is to employ smooth, continual motion adjustments for collision avoidance, in addition to constraint satisfaction.

# Chapter 6

## Validation

This chapter presents an experiment that combines all the proposed techniques and validates the demand satisfaction of the simulation results. In particular, our goal of this experiment is to validate whether our methods can satisfy the four demands *at the same time*.

### 6.1 A Virtual City

To validate the demand satisfaction of our two-level crowd simulation framework, we have tested the framework on a particular target environment. The target environment is a *virtual city* that consists of small blocks. Each block has four crosswalks, one bench, and one get-together area. By copying and pasting this block on the environment, we could create a city as large as we wanted. At the high-level of the framework, situations operate crosswalks, benches and get-together areas. At the low-level, both the crosswalk situations and the get-together situations use the probability scheme while the bench situations use the constrained-motion synthesis.

Figure 6.1 shows a target city environment and one block of the city with a detailed lay-out of situation displacement. The size of each block is approximately 50m x 50m.

The behaviors that we would like to simulate for each situation are as follows:

- **Crosswalk situation:** People are crossing the street according to traffic signs. For this goal, (1) the *waiting* and the *crossing* behavior function, (2) a *signal* sensor for checking traffic

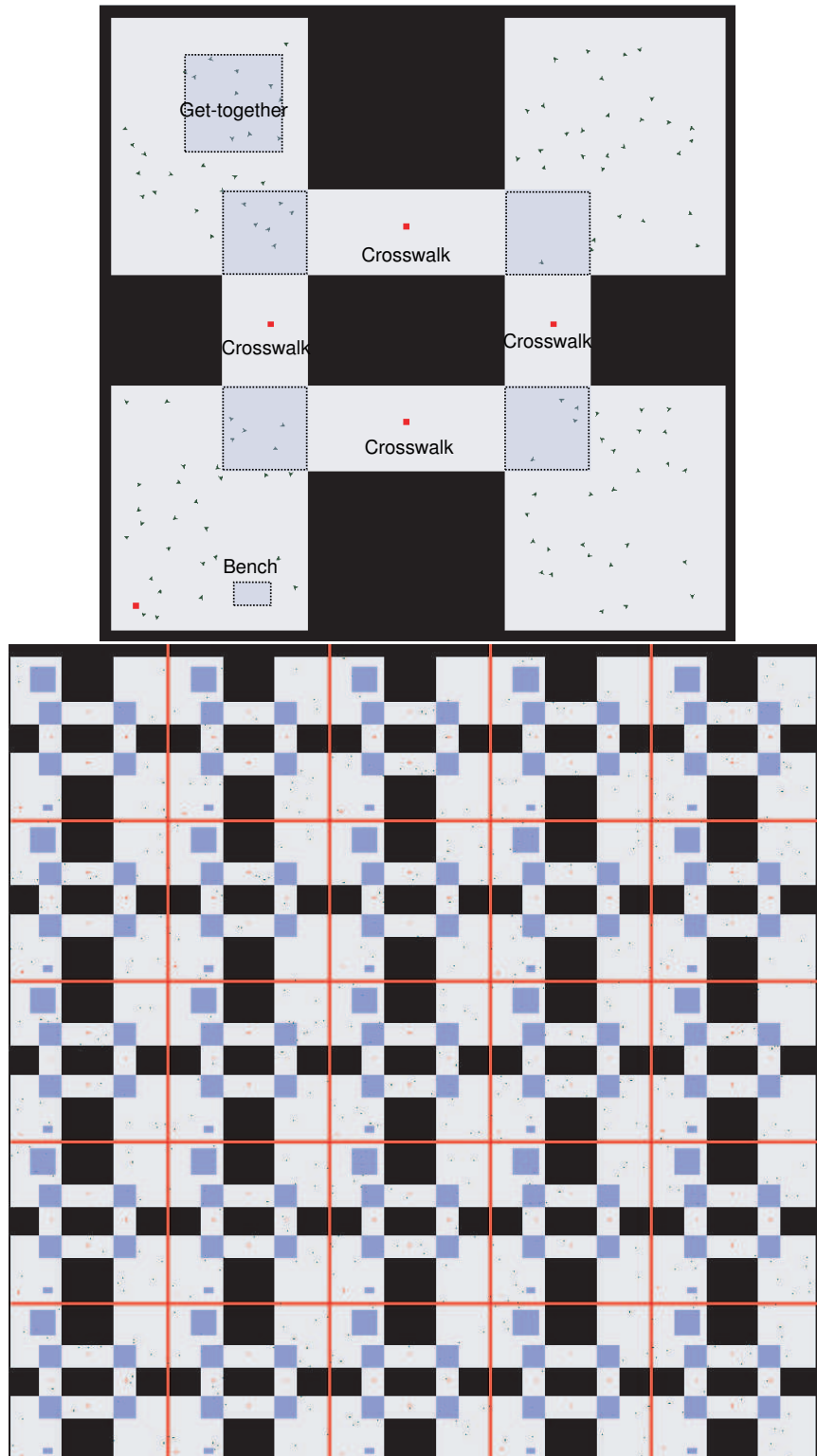


signs, and (3) a *rule* for determining which behavior functions should be composed for a given traffic sign are attached to the characters when they are in the crosswalk situation. To control the crowd's moving direction at a crosswalk, two opposite situations located between roads are connected to each other to provide a target position for each character. Crowd behaviors are determined by traffic signs, which we turn on and off manually, timers of which turn themselves on and off automatically.

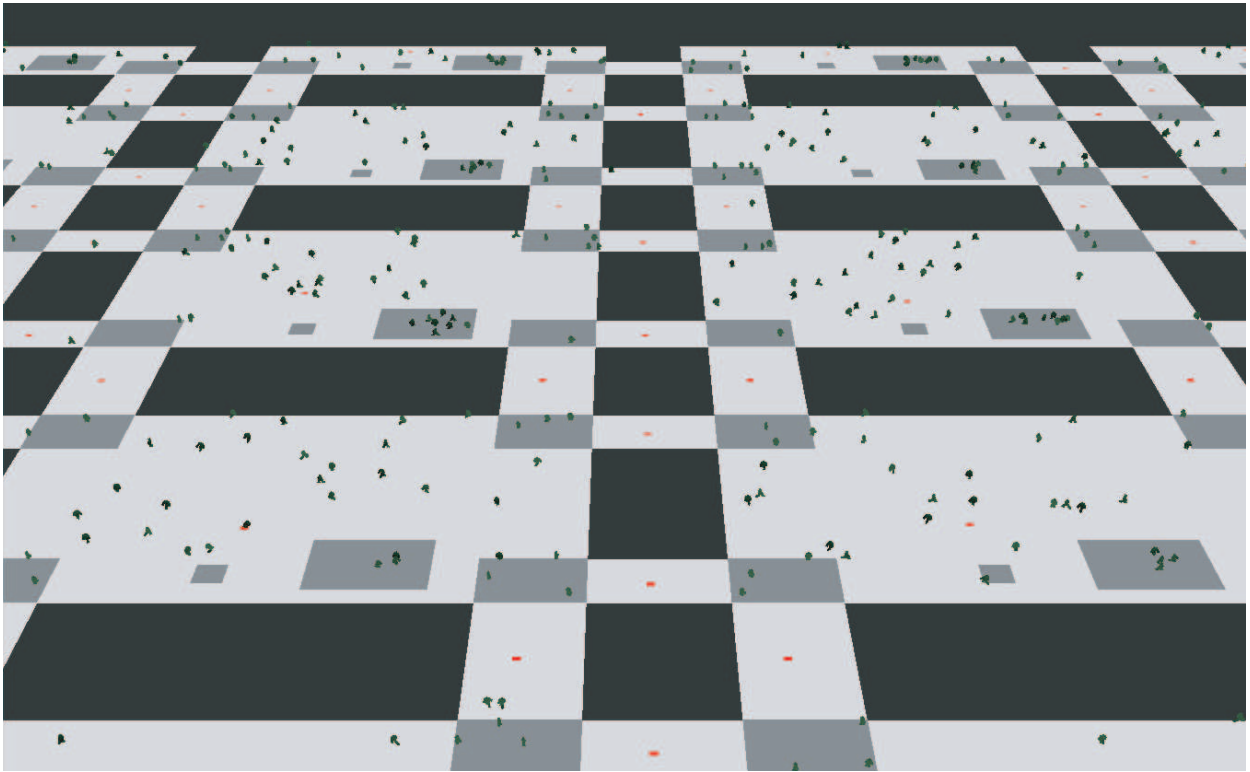
- **Bench situation:** Someone might need to rest for a while on a bench. The bench situation plugs an *empty* sensor into characters so that they can check whether someone occupies the bench or not. If it is empty, the characters call the constrained motion synthesis. The bench situation provides information such as location and orientation of the bench to the constrained motion synthesis. Then, the constrained motion synthesis creates a series of motions that makes the characters exactly sit down on the chair. After spending a random amount of time on the chair, the characters stand up and go away.
- **Get-together situation:** People often find an interesting place on the street. In this case, they gather together at that place for a while. The get-together situation plugs the *approach* behavior function into the characters and makes characters gather together around the pre-defined position. The time duration and the gathering position are given by the situation. Once the time duration expires, the people disperse.

## 6.2 Results

We have tested our algorithm on a standard Pentium-IV PC (1G memory) that has a standard graphics acceleration card. Figure 6.2 shows the snapshot of simulation in which 2,000 people occupy the virtual city, and Figure 6.3 shows crowd behaviors that are more detailed and that encompass the three situations.



**Figure 6.1:** Top: One block of a city; it contains four crosswalks, one bench and one get-together. Bottom: A city that has 25 blocks from the copied and pasted block above.

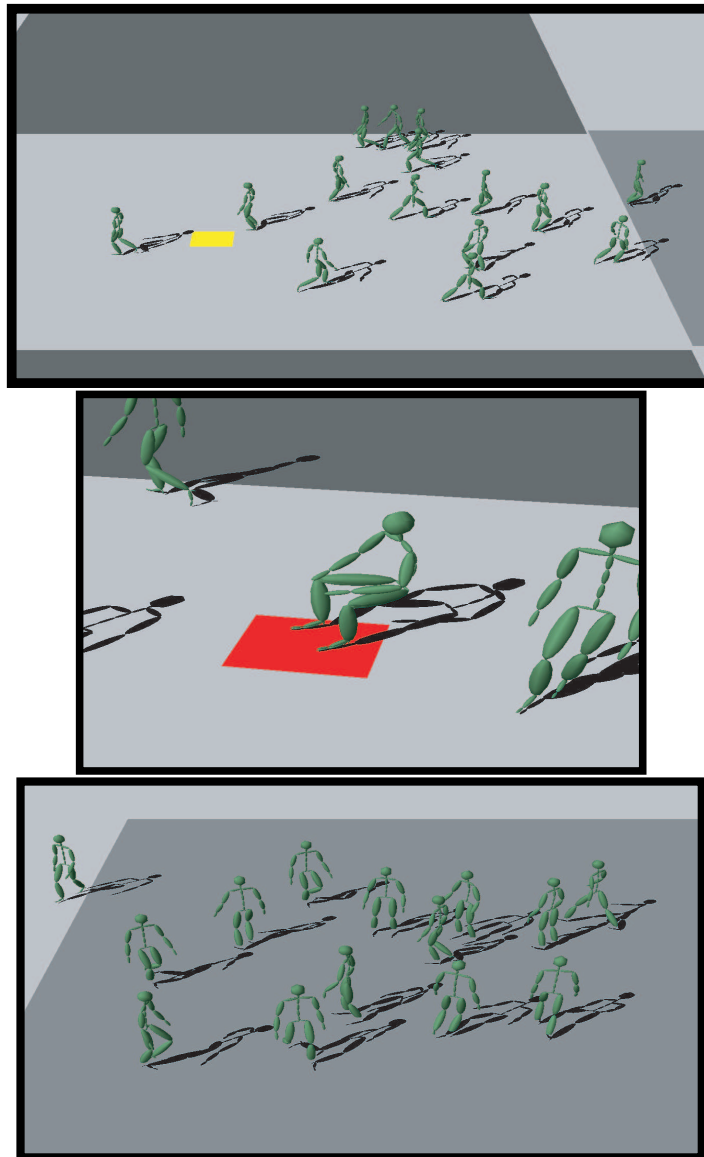


**Figure 6.2:** Crowd (2,000 people) simulation on a city environment.

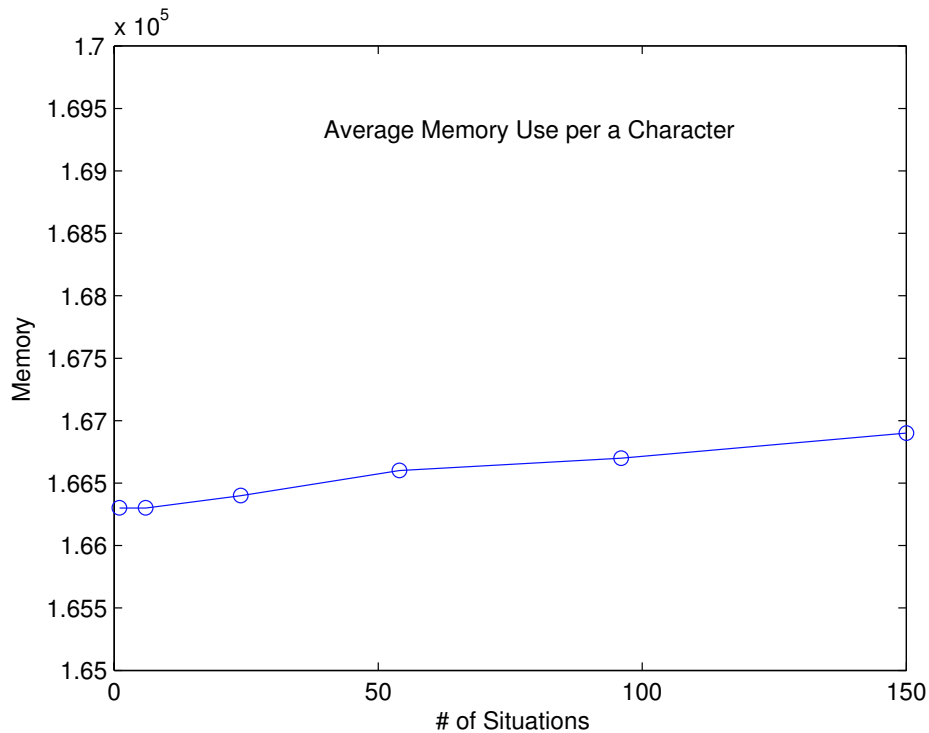
The validation of the four demands' satisfaction is as follows:

- Scalability:** To validate scalable memory demand, we have created 1,000 characters on the city and computed the average memory use per a character for 2,000 simulation steps as we increase the number of situations. The results are shown in Figure 6.4. As we can see in the graph, the average memory use for a character grows very slowly even when the number of situations increase. Underlying this outcome is the fact that each character can be under only a few situations at the same time, and once he or she is out of the situations, all information related to the situations leaves the character automatically. This fact limits characters' memory increase.

Scalable authoring is another specific demand of scalability. Because the situation encapsulates all information for particular local behaviors as a highly compact form, a big and



**Figure 6.3:** **Top:** People are crossing the street only when the traffic sign says “walk” (yellow box). **Middle:** When a bench is empty, some people sit down on the bench. **Bottom:** People are getting together at some place for a while.



**Figure 6.4:** The average memory use of 1,000 characters for 2,000 simulation steps.

complicated environment could be easily created through a simple copying and pasting of existing situations. In this manner, we created the city environment by repeatedly copying and pasting one block of the city, which contains a set of situations, onto the environment. Our painting interface is quite useful in this case because it allows us to directly edit the situations.

- **Controllability:** For control of the crowd flow, two opposing *crosswalk* situations that are located between the road connect to each other. As we can see in the Figure 6.1, the intersection in the middle of the block has four crosswalks, and each *crosswalk* situation is connected with two other crosswalks across the street. When traffic signs change to “walk,” the *crossing* behavior function obtains random positions of situations on the other side and sets those positions as goal positions for characters. Then, the *target-finding* behavior function gives a

high probability to the actions that make the characters cross the street and approach to their target positions. In this way, we can control the crowd flow by simply turning the traffic sign on and off, which meets the high-level controllability demand. The probability scheme infuses variability into the crowd motion because the selection of subsequent actions occurs through random sampling. Therefore, not all characters show the same motion pattern over time.

Similarly, the *get-together* situation features the *approach* behavior function, which makes crowds gather together around a pre-defined position by setting high probabilities for the actions that occur close to that position.

Unlike crosswalk and *get-together* situations, where a character's movements toward a particular position do not need to be exact, the *bench* situation uses the constrained-motion synthesis because the character needs to sit down on a bench in a precise way. Even though the bench can be located in anywhere and with arbitrary orientation, the simulation result confirms that our constrained motion synthesis is able to find a series of motions that make characters sit down on the bench in a precise way. In short, this result validated the controllability demand of the constrained motion synthesis techniques.

- **Convincingness:** First, the simulation results (please refer to the animation video) showed that there was no collision between characters and that there was no discontinuity in the synthesized motions of the crowd. This fact validates the visual convincingness demand.

Second, depending on the animator-installed environmental situations, crowds exhibited in an appropriate way particular behaviors such as waiting, street crossing, and bench sitting. This result validates the semantic-convincingness demand.

- **Efficiency:** The efficiency demand can be validated by checking the speed of algorithms. Specifically, our goal was to speed up the efficiency of three algorithms, which relate to 1) fast motion synthesis, 2) fast collision detection and 3) fast simulation performance.

First, table 6.1 shows the average time for collision detection and the constrained motion synthesis relative to 1,000 characters for 2,000 simulation steps for validating the second and the third algorithm. The density of crowds is 4.85 and the physical complexity of environment is 30.3. Remember that constrained motion synthesis was used for the bench situation, where characters needed to find a series of motions before sitting down on the bench. Our simulation result showed that our MOBB tree technique was around 2.2 times faster than simple frame-by-frame method and the constrained motion synthesis technique could synthesize motions in real time (more than 30 frames for a second). These two results validate the second and third demand above.

Second, the overall simulation performance is validated through checking the total simulation time as we increase the number of characters. The speed of algorithm is affected by several factors. First, the density of crowd affects the overall performance significantly because the collision testing takes long time when a lot of characters cram together. Second, the physical complexity of the environment is also closely related to the simulation speed because synthesized motions should test for collisions with all objects in the environment. Especially, the constrained motion synthesis might need many perturbations in the random search process to avoid collision with other obstacles in the environment, which makes system performance slow.

We have performed two experiments to test how the crowd density influences the overall simulation performance. Our first experiment was the case when we increased the number of characters while keeping the size of the environment fixed (increasing crowd density). Out

Efficiency Demand	Average Speed
Fast Collision Detection through MOBB	1.70695e-005 sec (frame-by-frame :3.74344e-005)
Fast Constrained Motion Synthesis	0.106 sec (total number of frame : 580)

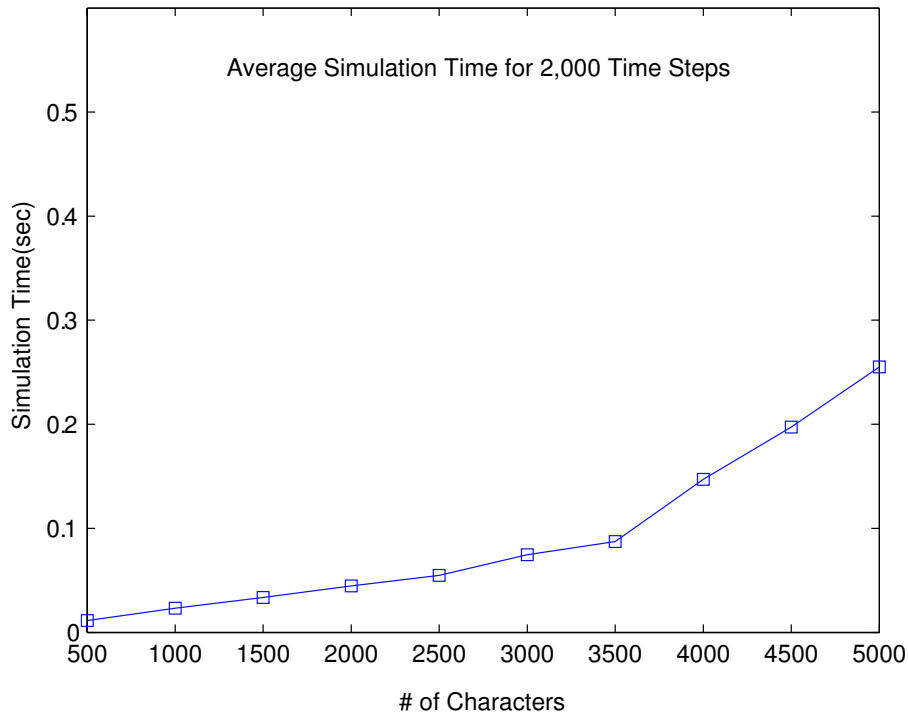
**Table 6.1:** Average speed of collision detection and constrained motion synthesis of 1,000 characters for 2,000 time steps.

second experiment was the case when the crowd density was fixed (variable environment). In these two experiments, the physical complexity of environment was fixed (30.3).

**Fixed size of environment:** Figure 6.5 depicts the average simulation time (excluding rendering) of a crowd when, for 2,000 simulation steps, the crowd features a fixed number of situations and a fixed size environment. The number of situations was 150 and the size of the environment was 25 blocks. To obtain the average simulation time, we first computed the time cost of a pre-defined number of characters for each time step, and we averaged the time cost for 2,000 time steps. We repeated the experiment as we increased the number of characters. As we can see in the graph (Figure 6.5), the simulation time increased slowly as we increased the number of characters until it reached up to 3,500. However, when it exceeded 3,500, the slope of the graph got stiff because the density of the crowd was too high for the environment. At this moment, the crowd simulation spent a great deal of time in collision detection among characters. Our experiment showed that the density at this point was around 17.0.

**Fixed crowd density:** Figure 6.6 depicts the average simulation time when the environment applies a constant density to the crowd (15.0). For maintaining constant density, the environment size should increase with crowd growth. As a result, the average simulation speed grew slowly (in a linear fashion) as the number of characters increased.



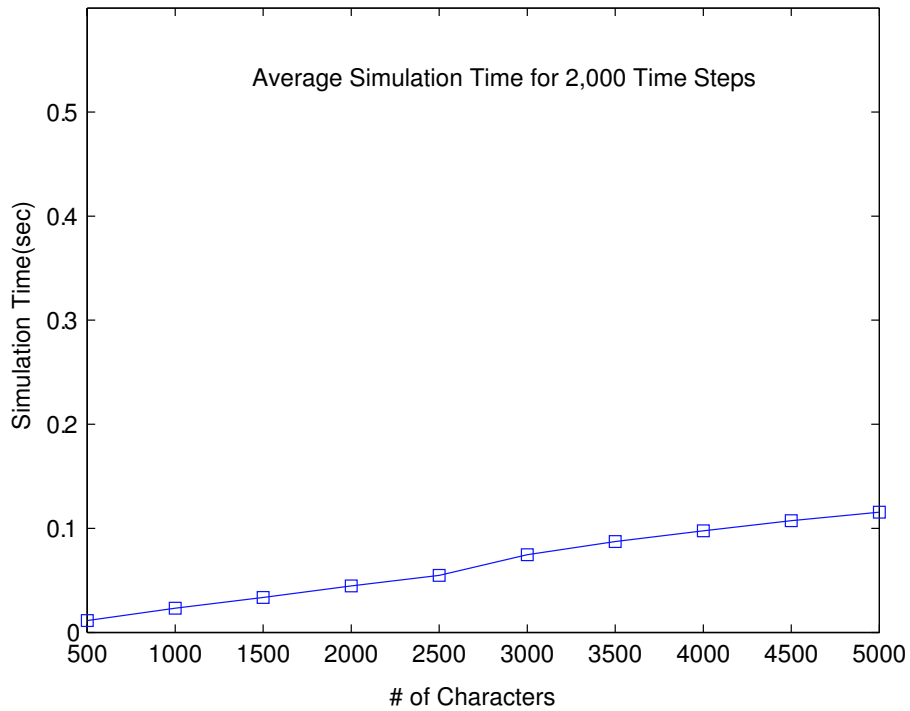


**Figure 6.5:** The average simulation time (excluding rendering time) of crowds with the fixed number of situations (150) and the fixed size of environment for 2,000 simulation steps.

## 6.3 Discussion

The primary reason that the scalable authoring demand can be satisfied is that we embed the *Python* script language into our crowd simulation system. Inside the system, all situations can be specified as simple *Python* scripts. By manipulating the scripts, we can add or delete situations as we need to. The *Python* script language also facilitates the creation of rules of the situations. The rules access a character's memory in a way that determines which behavior functions should be composed at any given moment.

The simulation speed is influenced heavily by crowd density in the environment. If the density is too high, then it takes a long time to check collision between characters. This makes overall system speed slow. Other crowd modelling techniques such as the particle-based-crowd model or



**Figure 6.6:** The average simulation time ( excluding rendering time) of a crowd when that crowd has a constant density (15.0) for 2,000 simulation steps.

the social force model are more desirable in this special case than is our method because these other methods can specify physical force between characters easily. Unless our goal is to simulate such “stuck-and-jammed” situations, fast simulation requires proper crowd density.

# Chapter 7

## Conclusion

### 7.1 Summary

Simulating crowds is a difficult task — not only because we need extra computational time for animating many characters, but also because the crowd behaviors are highly complex, and because it is hard for them to maintain the visual and semantic convincingness. This dissertation has established four specific demands for crowd simulation (scalability, controllability, efficiency and convincingness), and proposed a novel two-level crowd simulation framework that satisfied these demands at the same time.

In the two-level crowd simulation framework, the high-level part was responsible for providing information to characters and the low-level part synthesized motions by using that information. Specifically, the high-level part adopted the distributed crowd control mechanism called *situations*. Each situation contained behavior functions, local motions, rules, sensors and constraints. All information was automatically augmented to the characters when they entered a particular region in the environment or if they have a relationship with other people. This information leaves the characters when they exit the situation. By using that information, at the low level, the *probability scheme* or the *constrained motion synthesis* was used for synthesizing motions for individual characters. While the probability scheme applied to aggregate behaviors where the individual has

no specific constraints to satisfy (such as position or orientation), the constrained motion synthesis generated motions for a specific individual that meets the constraints that either the animator specifies directly or through situations.

To set the particular situation into the environment, we have adopted the easy-to-use painting interface by which we could use to specify either a region that each situation affected or a group of people who had a common relationship.

For fast collision detection between two motions in the motion-synthesizing step at the low level, we proposed the MOBB tree representation of motion, which encapsulated the details of motion into a simple bounding-box structure. To accelerate the test, we hierarchically tested the collision between two bounding boxes, from the top node to the bottom node of the two MOBB trees.

Throughout this dissertation, we made three primary contributions:

1. **A scalable, controllable and convincing framework of crowd simulation:** We introduced a novel two-level crowd simulation framework. Chapter 3 presented the high-level situation-based approach with the low-level probability scheme, which enabled us to simulate the complicated crowd behaviors and, at the same time, to avoid complicating the character architecture through both the situation composition and the probability composition. The situation composition made it possible to emulate complicated situations where several situations influence crowds simultaneously, and the probability composition made it possible to combine factors that affect the final action selection probabilistically. In addition, we showed that the user-friendly painting interface improved the controllability and semantic convincingness by setting every situation in the environment directly. The use of the graph-based motion synthesis at the low level showed visually convincing motions of crowds. We verified the demands by performing various experiments.

2. **Fast collision detection between motions:** Chapter 4 showed the fast collision detection between characters through MOBB trees. The MOBB (Motion Oriented Bounding Box) representation made it possible to encapsulate the detail of motion into a simple spatio-temporal hierarchical tree. Our experiments confirmed the assertion that the use of MOBB accelerated collision testing among many characters, which is an outcome that improved the efficiency demand. Because bumping among characters is the most noticeable artifact that weakens visual convincingness, we used this method to strengthen the visual convincingness.
3. **Fast and accurate constrained motion synthesis for crowds:** Chapter 5 illustrated the constrained motion synthesis algorithm, which is one of the two action selection mechanisms at the low-level. Given position, orientation, particular pose and time duration constraints for a specific individual, our motion synthesis algorithm created a series of motions that exactly met the constraints by performing efficient and fast random search on a structured motion graph. The primary benefit of this algorithm was its ability to satisfy the controllability demand. We could set arbitrary constraints on a character at any time as long as the constraints were legal (e.g., not inside the obstacles). Fast solution finding also improved the efficiency demand because hundreds of character motions could be synthesized at the same time. We performed several experiments to validate our demand satisfaction.

Table 7.1 recaps the demand satisfaction of proposed techniques.

## 7.2 Applications

Our methods are appropriate for a variety of applications. For a special effect in a movie, our constrained motion synthesis algorithm can create realistic animations of several hundreds of characters in real time. In these scenes, the target constraints might be given to each character automatically when he or she is at some particular situations, or when he or she is created. By

	<b>Scalability</b>	<b>Controllability</b>	<b>Efficiency</b>	<b>Convincingness</b>
Situation-Based Simulation with Probability Scheme	x	x		x
Collision Detection Using MOBB Trees			x	x
Constrained Motion Synthesis		x	x	x

**Table 7.1:** Summary of demand satisfaction.

setting constraints serially for multiple characters, we can also make characters interact (e.g. fight) with each other.

Our situation-based approach enables us to build a rapid prototype of wide-ranging virtual environments easily. For example, a virtual shopping mall, a virtual theater, or a virtual street environment can be easily populated with crowds that react to the target environment appropriately. This populating can be realized through the installation of specific situations into the environment. Such virtual environments can be used in many different ways. For urban planning, virtual crowds provide information to architects that helps them to predict actual crowd movement before a building's construction is started. For police or military training with regard to the handling of demonstrations or riots, virtual crowds in a virtual environment provide a realistic training environment. Virtual crowds provide important information to the psychology community as well because members of the community study crowd behaviors as a particular research area.

Other than these applications, one application for which our method should undergo further development is the online animation of user-controlled characters, such as the animation of characters in a game. Current war strategy games need a lot of character movement. Unfortunately, current character animation lacks convincing visual quality because the motions are created through a hand-crafted key frame technique or through specific procedural techniques. Our graph-based motion synthesis, which uses a constructed motion graph, produces realistic character motions by

using motion-capture data. One drawback to the use of motion-capture data is the lack of responsiveness of user control. Since each motion clip is composed of a number of frames, all user control inputs are ignored until the last frame of the current motion is shown, and this creates latency of response between user input and actual animation of the character. Thus, to use motion capture data for games, we need to develop specially designed tools for motions so that rapid transitions can be made realistically.

Our situation-based approach is also quite useful for the development of a large complex game environment. By associating a particular game scenario with a particular situation, we can make characters follow the scenario. For example, if the main character enters some room, then all enemy characters respond to the main character by following a pre-defined situation. The situation specification might include motions, rules, sensors, and finite state machines for controlling the main character's enemy.

## **7.3 Limitations and Future Work**

Our proposed techniques have some limitations. This section introduces the limitations of our methods and discusses future research work.

### **7.3.1 Limits of the two-level crowd simulation framework**

One limitation of our crowd simulation framework is with the natural flow control. For example, if we want to simulate a large group of people exiting through small gates when some emergent event happens, people might be in a panic and try to get to the gates as fast as they can, which would cause a bottleneck. When a bottleneck occurs, an individual's movement slows down but still continues toward the gate to exit the gates one by one. Our situation might be able to control the crowd so that they can stop walking when the bottleneck is predicted, but it cannot simulate this realistic flow. The primary reason is our graph-based motion synthesis. Our graph-based

motion synthesis cannot control the speed of the motions. Thus, we cannot set arbitrary speed of the motion clips, which is required for animating slow walking motion when a bottleneck is anticipated. Another drawback of fixed motion data is that we cannot control gait length of walking motion, which is also required for bottleneck situations. Other crowd modelling approaches such as a social force model is useful on this scene because an inter-agents force can be specified easily.

Another limitation of our approach is that spatial situation, assumes that the regions, associated with the situations, do not change during simulation. However, for special cases, we need to change the region dynamically. For example, if we want to simulate the crowd inside of a bus or a train, the bus situation or the train situation should move in global coordinates depending on the location of the bus or the train. Our current situations cannot simulate such special cases.

The ability to dynamically coordinate the local behavior of a group of people is another drawback of our simulation model. The non-spatial situation can specify the socially related local behaviors to particular group of people easily. However, such situations are set before the simulation begins, and once simulation begins, they cannot be changed to other non-spatial situations. For some applications, we might need to change the current situation to other non-spatial situation so that all group members can show different local behaviors. However, our current system does not allow us to coordinate a series of different socially related behaviors over time.

### **7.3.2 Expressiveness**

Our situation-based approach is quite general because it considers the social relationship among crowds as well as the locational information effect. However, our current implementation cannot simulate complicated human reasoning processes although human motion is a product of these processes. People usually behave according to their internal state and condition. For example, a person who is hungry tries to find a restaurant by looking around and sometimes checks road signs. Our approach does not maintain internal emotional state in a character's memory. This



lack of internal state limits the expressiveness of our approach. Our situation approach expresses motions that are not initiated from the characters's internal reasoning process. They can only represent behaviors that are initiated by their location and social relationships. This is inevitable because we cannot specify the infinite number of internal states given the fact that one of our goals is to create the crowds that are applicable to many different environments. For the same reason, it is impossible to come up with infinite number of rules for specifying the relationship between behaviors and internal status. If our goal is not to create generic crowds, we can relax our requirement so that we can put a fixed size of storage for internal state for character. This storage can be used for storing emotional states of character, and we can specify the special behaviors which are associated with the internal states.

In our approach, there are only four different kinds of sensors, which are signal sensor, agent sensor, empty sensor and proximity sensor. But, for some applications, we might require more complicated sensors instead. For example, people try to avoid jammed situation when they walk. To check the density of crowds, we might need a "density" sensor that captures how many people there are in front of the character.

For animators, it is quite easy to set a particular situation to environment because we provide a painting interface that enables them to specify the situations on the environment directly. But, it might be hard for them to create a new situation and behavior functions. Our current behavior functions are all hard-coded, and we do not provide user interfaces to create new behavior functions. Providing a "template" behavior function and situation is able to improve the expressiveness of our approach, which will be part of the future work.

### **7.3.3 Limits to Motion Data**

The wide range of crowd behavior depends on the available motion clips. If there is no proper motion, then the behavior, which is just a pattern of motion selection over time, cannot be shown.

Because it is infeasible for us to capture all required motions, we need to adjust current motion clips so that we can use a small number of motions in various kinds of applications. For instance, by using a small set of reaching motions, we should be able to create motions that have different reaching points. In this case, motion blending or Inverse Kinematics (IK) technique makes it possible to create new motions from the existing motions. However, for the crowd-simulation problem, the fast processing of those techniques is in high demand, which is a fact that we will leave for future research that needs to be conducted.

### **7.3.4 Off-line Constrained Motion Synthesis**

Our current constrained motion synthesis is not on-line even though it is quite fast. A major bottleneck that keeps the algorithm off-line is that the time is required to repeatedly search on the motion graph when the initial seed paths are perturbed until they get close enough. One solution that might result in an online algorithm hinges on the storing of intermediate perturbed motions in a cache and reuse those motions when similar initial and target constraints are given to the character. A similarity metric is required to check how closely the cache entries match up to the given new constraints. If they are close enough, then it represents the cache-hit. Otherwise, it represents the cache-miss. If the cache hit occurs, then the intermediate motions are retrieved and used for finding motions that meet the constraints. Otherwise, random searching is performed for the given new constraints.

### **7.3.5 Character Rendering**

Our current approach does not consider the time needed to render the crowd even though the experiments prove that rendering is the most time consuming job in crowd simulation [AW04]. The fast drawing of many characters requires a special rendering technique. The LOD (Level of Detail) of character geometry or motion data reduce the rendering time significantly. In particular,

the LOD representation of motion data is quite useful because detailed motions are not necessary for characters whose location is far from the camera. Hardware-supported skinning or animation is another method for the improvement of the visual quality of crowds although these methods might require an intermediate hardware control language such as *Cg* or GLSL(OpenGL Shading Language).

### 7.3.6 Intuitive User Interface

The user interface for crowd control presents another problem that needs to be solved. Our painting interface is useful for controlling the local behavior of crowds, but it is not appropriate for global control over crowds. For example, for the scene in which a large crowd is marching, the animator might need to control the crowd's overall shape over time. In this case, the shapes of crowds over time become constraints, and each individual should communicate with other characters to maintain the constraints during simulation [AMC03]. Therefore, animators need an additional user interfaces to set these shape constraints and the specification of particular behaviors while the crowds are marching. For instance, a sketch-based interface is desirable for global shape constraints. In addition to this, we also need a way to specify particular behaviors for each character during the simulation. A simple time-line interface might be the best choice for a single character, however, when many characters are required, it is not reasonable to have a time-line for each character. Therefore, for multiple characters, user interface which is higher level and can be shared by multiple characters is desirable. This problem is also left for future research.

# LIST OF REFERENCES

- [AC01] AYLETT R., CAVAZZA M.: Intelligent virtual environment-a state of the art report. In *Proceedings of Eurographics 2001 STARs* (2001).
- [AF02a] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics* 21, 3 (2002), 483–490.
- [AF02b] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics* 21, 3 (2002), 483–490.
- [AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *ACM Transactions on Graphics* 22, 3 (2003), 402–408.
- [AI-] Ai-implant. <http://www.biographictech.com/>.
- [AMBJ02] ABDEL-MALEK K., BLACKMORE D., JOY K.: Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling* (2002). Submitted.
- [AMC03] ANDERSON M., MCDANIEL E., CHENNEY S.: Constrained animation of flocks. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 286–297.
- [AW04] AHN J., WOHN K.: Motion level-of-detail:a simplification method on crowd scene. In *Proc. of the 2004 Computer Animatio and Social Agents* (2004).
- [BC89] BRUDERLIN A., CALVERT T.: Goal-directed dynamic animation of human walking. In *Proceedings of ACM SIGGRAPH 89* (Aug. 1989), Annual Conference Series, ACM SIGGRAPH, pp. 233–242.
- [BC97] BOUVIER E., COHEN E.: From crowd simulation to airbag deployment:particle system, a new paradigm of simulation. *Journal of Electronic imaging* (1997), 94–107.

- [BEG\*04] BASCH J., ERICKSON J., GUIBAS L. J., HERSHBERGER J., ZHANG L.: Kinetic collision detection for two simple polygons. *Computational Geometry* 27, 3 (2004), 211–235.
- [BH00] BRAND M., HERTZMANN A.: Style machines. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Annual Conference Series, ACM SIGGRAPH, pp. 183–192.
- [BLA02] BAYAZIT O. B., LIEN J., AMATO N.: Better group behaviors in complex environments using global roadmaps. In *Proceedings of the 2002 Artificial Life (ALIFE)* (Dec. 2002).
- [BMdB03] BRAUN A., MUSSE S., DEOLIVERIA L., BODMANN B.: Modeling individual behaviors in crowd simulation. In *Computer Animation and Social Agents(CASA)* (2003).
- [BMTT90] BOULIC R., MAGNENAT-THALMANN N., THALMANN D.: A global walking model with real-time kinematic personification. *Visual Computer* 6, 6 (1990), 344–358.
- [BUT04] BOULIC R., ULICNY B., THALMANN D.: Versatile walk engine. *Journal of Game Development* 1, 1 (2004).
- [BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. In *Proceedings of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 97–104.
- [CK00] CHOI K.-J., KO H.-S.: On-line motion retargeting. *Journal of Visualization and Computer Animation* 11 (2000), 223–243.
- [DHOO05] DOBBYN S., HAMILL J., O’CONNOR K., O’SULLIVAN C.: Geopostors: a real-time geometry/impostor crowd rendering system. *ACM Trans. Graph.* 24, 3 (2005), 933–933.
- [Ebe01] EBERLY D.: *3D Game Engine Design: a practical approach to real-time computer graphics*. Academic Press, 2001.
- [FBT99] FARENC N., BOULIC R., THALMAN D.: An informed environment dedicated to the simulation of virtual humans in urban context. In *Proc. of EUROGRAPHICS 1999* (1999), Annual Conference Series, pp. 309–318.
- [Feu00] FEURTEY F.: Simulating the collision avoidance behavior of pedestrians. In *M.S thesis* (2000), Dept. of EE, the Univ. of Tokyo.
- [FH93] FOISY A., HAYWARD V.: A safe swept volume method for collision detection. In *The sixth international Symposium of Robotics Research* (1993), pp. 61–68.

- [FMS98] FARENC N., MUSSE S. R., SCHWEISS E.: One step towards virtual human management for urban environment simulation. In *Proceedings of the ECAI Workshop on Intelligent User Interfaces* (1998).
- [FMS00] FARENC N., MUSSE S., SCHWEISS E.: A paradigm for controlling virtual humans in urban environment simulations. In *Applied Artificial Intelligence* (2000), vol. 14, pp. 69–91.
- [FTT99] FUNGE J., TU X., TERZOPOULOS D.: Cognitive modeling: knowledge, reasoning and planning for intelligent character. In *Proceedings of ACM SIGGRAPH 99* (1999), Annual Conference Series, ACM SIGGRAPH.
- [FvdPT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001* (July 2001), Annual Conference Series, ACM SIGGRAPH, pp. 251–260.
- [FW01] FORBUS K. D., WRIGHT W.: Some notes on programming objects in *The Sims*, 2001.
- [Gle98] GLEICHER M.: Retargeting motion to new characters. In *Proceedings of ACM SIGGRAPH 98* (July 1998), Annual Conference Series, ACM SIGGRAPH, pp. 33–42.
- [GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics 30*, Annual Conference Series (1996), 171–180.
- [GLM99] GOLDSTEIN S., LARGE E., METAXAS D.: Non-linear dynamical system approach to behavior modeling. In *Applied Artificial Intelligence* (1999), vol. 15, pp. 349–364.
- [GR96] GUO S., ROBERGE J.: A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proc. of Eurographics Workshop on Computer Animation and Simulation '96* (Aug. 1996), pp. 95–107.
- [GSKJ02] GLEICHER M., SHIN H., KOVAR L., JEPSEN A.: Snap-together-motion. In *Proceedings of ACM SIGGRAPH 2002 Symposium on Interactive 3D Graphics* (2002), ACM SIGGRAPH.
- [GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: assembling run-time animations. In *Proc. of the 2003 Symposium on Interactive 3D graphics* (2003), pp. 181–188.
- [HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamics feature of escape panic. In *Nature* (2000), pp. 487–490.

- [HGP04] HSU E., GENTRY S., POPOVIĆ J.: Example-based control of human motion. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 69–77.
- [HM95] HELBING D., MOLNAR P.: Social force model for pedestrian dynamics. In *Physical Review* (May 1995), pp. 4282–4286.
- [HWBO95] HODGINS J., WOOTEN W., BROGAN D., O'BRIEN J.: Animating human athletics. In *Proceedings of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 71–78.
- [JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005)* 24, 3 (Aug. 2005).
- [KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics* 21, 3 (2002), 473–482.
- [KKS03] KIM D., KIM H. K., SHIN S. Y.: *An Event-Driven Approach to Crowd Simulation with Example Motions*. Tech. Rep. CS/TR-2003-186, KAIST, 2003.
- [KL94a] KAVRAKI L., LATOMBE J.-C.: Randomized preprocessing of configuration space for fast path planning. In *IEEE Int. Conf. Robotics and Automation* (1994), pp. 2138–2145.
- [KL94b] KAVRAKI L., LATOMBE J. C.: Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation* (1994), pp. 2138–2145.
- [KO04] KAMPHUIS A., OVERMARS M. H.: Finding paths for coherent groups using clearance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2004), ACM Press, pp. 19–28.
- [Koe] KOEPEL D.: Massive attack. <http://www.popsci.com/popsci/science/article/0,12543,390918-1,00.html>.
- [Kov04] KOVAR L.: Automated methods for data-driven synthesis of realistic and controllable human motion. *Ph.D Thesis, Computer Sciences department, University of Wisconsin-Madison* (2004).
- [KPS03] KIM T., PARK S., SHIN S.: Rhythmic-motion synthesis base on motion-beat analysis. *ACM Transactions on Graphics* 22, 3 (2003), 392–401.
- [KT98] KALLMANN M., THALMANN D.: Modeling objects for interaction tasks. In *Proceeding of Eurographics Workshop on Animation and Simulation 1998* (1998), pp. 73–86.

- [KVLM03] KIM Y. J., VARADHAN G., LIN M. C., MANOCHA D.: Fast swept volume approximation of complex polyhedral models. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 11–22.
- [LCF05] LAI Y.-C., CHENNEY S., FAN S.: Group motion graphs. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 281–290.
- [LCR\*02a] LEE J., CHAI J., REITSMA P., HODGINS J., POLLARD N.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics* 21, 3 (2002), 491–500.
- [LCR\*02b] LEE J., CHAI J., REITSMA P., HODGINS J., POLLARD N.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics* 21, 3 (2002), 491–500.
- [LJC01] LI T., JENG Y., CHANG S.: Simulating virtual human crowds with a leader-follower model. In *Proc. of Computer Animation* (2001), The Computer Graphics Society and the IEEE Computer Society, pp. 93–102.
- [LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 79–87.
- [LS99] LEE J., SHIN S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 99* (Aug. 1999), Annual Conference Series, ACM SIGGRAPH, pp. 39–48.
- [LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: A two-level statistical model for character motion synthesis. *ACM Transactions on Graphics* 21, 3 (2002), 465–472.
- [mas] Massive software. <http://www.massivesoftware.com/>.
- [MBC01] MIZUGUCHI M., BUCHANAN J., CALVERT T.: Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations* (Sept. 2001).
- [Men00] MENACHE A.: *Understanding Motion Capture for Computer Animation and Video Games*. Academic Press, 2000.
- [MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Transaction on Graphics* 22, 3 (2003), 562–568.
- [Mir96] MIRTICH B.: *Impulse-based Dynamics for Rigid-Body Simulation*. PhD thesis, University of California, Berkeley, 1996.



- [MT01] MUSSE S. R., THALMANN D.: Hierarchical model for real time simulation of virtual human crowds. In *IEEE Transaction on Visualization and Computer Graphics* (2001), pp. 152–164.
- [OS94] OVERMARS M., SVESTKA P.: A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics* (1994), pp. 19–37.
- [Per95] PERLIN K.: Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (Mar. 1995), 5–15.
- [PG96] PERLIN K., GOLDBERG A.: Improv: a system for scripting interactive actors in virtual worlds. In *Proceedings of ACM SIGGRAPH 96* (Aug. 1996), ACM SIGGRAPH, pp. 205–216.
- [PLS03] PETTRÉ J., LAUMOND J.-P., SIMÉON T.: A 2-stages locomotion planner for digital actors. In *SCA '03: Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 258–264.
- [PSS02] PARK S. I., SHIN H. J., SHIN S. Y.: On-line locomotion generation based on motion blending. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 105–111.
- [PSS04] PARK S. I., SHIN H. J., SHIN S. Y.: On-line motion blending for real-time locomotion generation. In *Computer Animation and Virtual Worlds 15* (2004), pp. 125–138.
- [RCB98] ROSE C., COHEN M., BODENHEIMER B.: Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Application 18*, 5 (1998), 32–40.
- [Rey87] REYNOLDS C.: Flocks, herds, and schools: A distributed behavior model. In *Proceedings of ACM SIGGRAPH 87* (July 1987), Annual Conference Series, ACM SIGGRAPH.
- [RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies. In *Proceedings of Eurographics Conference* (2002).
- [RKL\*04] REDON S., KIM Y., LIN M., MANOCHA D., TEMPLEMAN J.: Interactive and continuous collision detection for avatar in virtual environments. In *Proceedings of Virtual Reality Conference 2004(VR)* (Mar. 2004), pp. 117–283.
- [RKLM04] REDON S., KIM Y., LIN M., MANOCHA D.: Fast continuous collision detection for articulated models. In *In Proceedings of ACM Symposium on Solid Modeling and Applications* (2004).
- [RP04] REITSMA P., POLLARD N.: Evaluating motion graphs for character navigation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2004* (Aug. 2004).

- [SCG04] SUNG M., CHENNEY S., GLEICHER M.: Scalable behaviors for crowd simulation. In *Computer Graphics Forum (EUROGRAPHICS '04)* (2004), vol. 23, pp. 519–528.
- [SE02] SCHÖDL A., ESSA I.: Controlled animation of video sprites. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002* (Aug. 2002).
- [SKG05] SUNG M., KOVAR L., GLEICHER M.: Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005* (2005), pp. 291–300.
- [SMH05] STARCK J., MILLER G., HILTON A.: Video-based character animation. In *ACM SIGGRAPH Symposium on Computer Animation* (July 2005), ACM.
- [Sof] Softimage—behavior. <http://www.softimage.com/products/behavior/v2/default.asp>.
- [SSSE00] SCHÖDL A., SZELISKI R., SALESIN D., ESSA I.: Video textures. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Annual Conference Series, ACM SIGGRAPH, pp. 489–498.
- [ST05] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 19–28.
- [Stu94] STURMAN D.: A brief history of motion capture for computer character animation. In *"Character Motion Systems", Notes for SIGGRAPH '94, Course 9* (1994).
- [TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environment. In *Eurographics Rendering* (2000).
- [TKOR05] THALMANN D., KERMEL L., OPDYKE W., REGELOUS S.: Crowd and group animation. *Course Note - SIGGRAPH 2005* (2005).
- [TLC02a] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Comput. Graph. Appl.* 22, 2 (2002), 36–43.
- [TLC02b] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. In *Computer Graphics Forum* (Nov. 2002), vol. 21.
- [TLCC01] TECCHIA F., LOSCOS C., CONROY R., CHRYSANTHOU Y.: Agent behavior simulator(abs): A platform for urban behavior development. In *Proceedings of GTEC 2001* (2001).
- [UCT04] ULICNY B., CIECHOMSKI P., THALMANN D.: Crowdbush: interactive authoring of real-time crowd scenes. *Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2004* (Aug. 2004).

- [UT01] ULICNY B., THALMANN D.: Crowd simulation for interactive virtual environment and vrtraining systems. *Proceedings of Eurographics workshop on Animation and Simulation* (2001), 163–170.
- [WCP\*] WRAY R., CHONG R., PHILLIPS J., ROGERS S., WALSH. B.: A survey of cognitive and agent architecture. <http://ai.eecs.umich.edu/cogarch0/>.
- [WH95] WOOTEN W., HODGINS J.: Dynamic simulation of human diving. In *Proceedings of Graphics Interface (GI'95)* (May 1995), pp. 1–9.
- [WH00] WOOTEN W., HODGINS J.: Simulating leaping, tumbling, landing, and balancing humans. In *IEEE International Conference on Robotics and Animation* (2000), vol. 1, pp. 656–662.
- [WP95] WITKIN A., POPOVIĆ Z.: Motion warping. In *Proceedings of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 105–108.
- [WP01] WATT A., POLICARPO F.: *3D Games: Real time rendering and software Technology*. Addison-Wesley, 2001.

## APPENDIX

### Behavior Functions

In this appendix, we describe several important behavior functions more in detail and gives a complex behavior that need to compose several behavior functions. Remember that the basic goal of behavior function is to implement conceptual notion of behavior by computing probabilities of all current available actions.

#### A.1 *Don't turn* behavior function

This behavior function implements a behavior that makes character walk as straight as possible. Given all available actions and the current actions, this behavior function computes the angle between current action and all available actions. Depending on the angle, it returns different probabilities. As the angle close to the 180 degree, which corresponds to straight line, the behavior function returns a high probability.

Let's say that current character position is  $P_c \in \mathbf{R}^2$  and previous position is  $P_p \in \mathbf{R}^2$ . Then, for all available actions  $A = \{A_1, A_2, \dots, A_n\}$ , the behavior function computes the angles  $v(i)$  between two normalized vector  $\overline{P_c P_p}$  and vector  $\overline{P_c P(A_i)}$  where the  $P(A_i)$  is the 2D position of the final frame of action  $A_i$ . The probability  $Prob(i)$  of action  $A_i$ , where  $1 < i \leq n$ , is then computed by sigmoid function as defined as equation 3.1. The  $\alpha$  can be changed from 0.01 to 0.1 as a parameter.

$$Prob(i) = \text{sigmoid}(v(i), \alpha)$$

## A.2 *Collision* behavior function

The *collision* behavior function prevents collision between characters. For this goal, this behavior function checks all available actions, returns zero probabilities for the actions that cause collisions, and returns high probabilities for the other actions. As a preprocessing, we subdivide the entire environment into a regular grid and make each cell of grid contain a list structure that tells us which agents located in the cell. As the character moves around the environment, the list structure of each cell is updated dynamically. The rough processing step of this behavior function is as follows:

1. Given agent *Myself*, obtain the list of agents in neighbors with distance  $d$  from the *Myself*.
2. For each agent in neighbors, which we call *Neighbors*, call the collision detection function **MOBB**(*Myself*, *Prob*, *Neighbors*).
3. The **MOBB** checks the collision of all available actions of *Myself* against current action of *Neighbors* and returns probabilities. If collision is anticipated, then it sets zero to *Prob*. Otherwise, it sets one.

The MOBB function is described in Chapter 4.

## A.3 *ImageLookup* behavior function

Our simulation has a bitmap environment image that layouts the entire environment. The goal of *ImageLookup* behavior functions is to look into the environment image to see what the probability of a character being there is. This behavior function enable to prevent the collisions between

character and obstacles in the environment because it returns low-probability for actions that cause collision with obstacles. For this purpose, the environment image is created as a greyscale image in which the black color corresponds to area for obstacles whereas the white color represents to “walkable” area of the environment. Suppose an agent has actions  $A = \{A_1, A_2, \dots, A_n\}$ , then for each action  $A_i$ , the *ImageLookup* behavior function first obtains the pixel value,  $v_i$ , of the image at the location of final frame of  $A_i$ . Then, the behavior function converts the pixel value into probability  $Prob(i)$  by computing following equation.

$$Prob(i) = slope * v(i)/max + intercept$$

where *slope* and *intercept* are tuning parameters and *max* is the maximum possible value of the pixel. The *slope* and *intercept* controls the probability linearly.

## A.4 *TargetFinding* behavior function

The *TargetFinding* behavior moves characters to their target positions as close as possible. Our simulation adopts the PRM(Probabilistic Roadmap Method) as a global path planning algorithm. Once a new target position is given to an agent, the agent issues a PRM query and obtains a series of way points  $w = \{w_1, w_2, \dots, w_m\}$  as a result which starts from the initial position and ends at the target position. Given current way point,  $w_c$ , which corresponds to the closest way point to the current character position, *TargetFinding* behavior function inputs all available actions  $A = \{A_1, A_2, \dots, A_n\}$  and computes the distance between the  $w_c$  and action  $A_i$  and sorts them in reverse order. The behavior function sets high probabilities for actions whose distance is short, and sets low probabilities for the other actions. To set the probability, this behavior function categorizes the whole actions into several groups  $G = \{G_1, G_2, \dots, G_m\}$  depending on distance from current

way point where the distance of group  $G_i$  is less than distance of group  $G_{i+1}$ . Then, it gives same probability to all actions in the same group.

One problem is that we have to decide when we move the current way point to the next way point. If it is too late, all standing-still actions get the highest probabilities once the character arrives at the current way point, and it makes characters stop too often before they reach final target position. We defines the minimum distance  $min_{dist}$  for this purpose. If the distance from the current character to the current way point is within the  $min_{dist}$ , then algorithm moves the current way point to the next way point.

## A.5 *Stop* behavior function

The *stop* behavior function implements a behavior for making character stop to walk. Given all available actions  $A = \{A_1, A_2, \dots, A_n\}$ , the behavior function computes the distance  $d$  from the first frame to the last frame of each motion  $A_i$ . if the distance  $d$  of an action is smaller than threshold value, then it represents a walking motion, therefore, the action gets high probability. Otherwise, it gets low probability.

## A.6 *StayInBox* behavior function

The *StayInBox* behavior function traps crowds inside a particular area in the environment where the area is specified as a polygon. Given all available actions  $A = \{A_1, A_2, \dots, A_n\}$ , this behavior function returns low probability to actions that moves character out of the area, and gives high probability to action that moves inside the area. In theory, all frames of a motion should be inside the area to get the high probability. But, it takes long time to check every frames of the motion. Instead, we use the bounding box of the root node of MOBB tree. If the four vertices of the bounding box are inside the area, then we assume that the motion does not move the character out

of the area. Whether or not a point  $(x, y)$  is inside a polygon is tested through following code:

(Refer to Graphics Gem IV)

```
//nPol : number of vertices of polygon
//xp : array of x coordinate of polygon
//yp : array of y coordinate of polygon
//x, y : sample point to be tested
int pnpoly(int npol, float *xp, float *yp, float x, float y)
{
    int i, j, c = 0;
    for (i = 0, j = npol-1; i < npol; j = i++) {
        if (((yp[i] <= y) && (y < yp[j])) ||
            ((yp[j] <= y) && (y < yp[i]))) &&
            (x < (xp[j] - xp[i]) * (y - yp[i]) / (yp[j] - yp[i]) + xp[i]))
            c = !c;
    }
    return c;
}
```

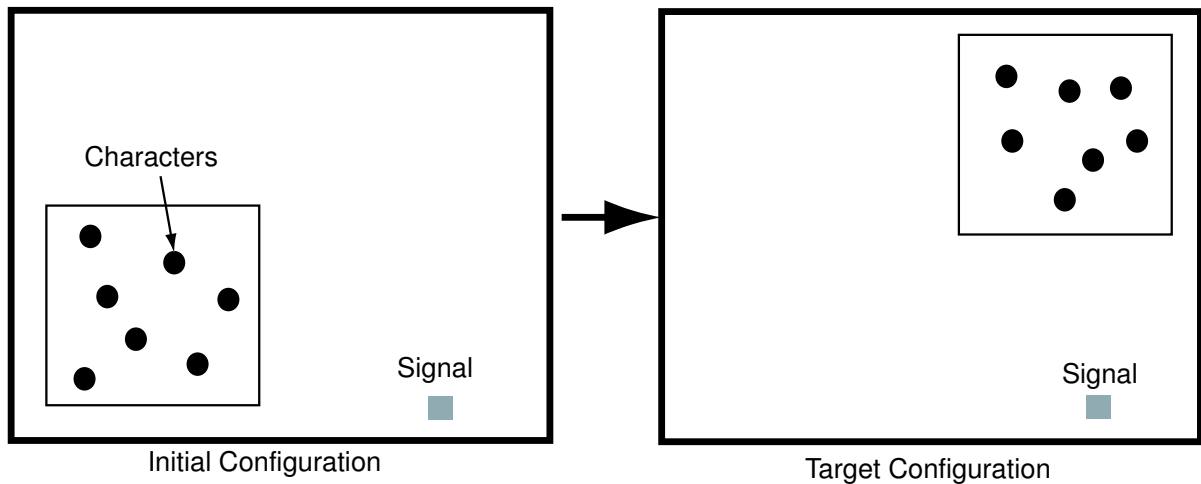
## A.7 An example of complex behaviors

In this section, we give an example of complex behaviors which are composed of several simple behavior functions described above. The goal of this behavior function is to move crowd from randomly distributed initial positions to the specific region of the environment according to the signal. This *move-to-region* behavior is composed of these following behaviors.

1. *Collision* behavior: prevents collisions between characters.
2. *ImageLookup* behavior: prevents collision with obstacles.
3. *TargetFinding* behavior: moves crowds to the target positions.
4. *StayInBox* behavior: traps crowds in a specific region.

Essentially, this behavior makes crowds wander around the initial region without collision and keep checking the event signal. Once the signal is on, it moves the crowd to the target region(Figure A.1).





**Figure A.1:** An complex *move-to-region* behavior.

A python script for the situation that controls the behaviors is following.

```

1. NAME = "MOVE_TO_REGION"
2. e = Situation(NAME)

#set capacity : -1=infinite
3. e.setCapacity(-1)

#region specification : x1-x4, y1-y4 = vertex positions
4. e.addSitPoint(x1,y1)
5. e.addSitPoint(x2,y2)
6. e.addSitPoint(x3,y3)
7. e.addSitPoint(x4,y4)

#add sensors : sx,sy = sensor position
8. s = SignalSensor(e,101)
9. s.setPos(sx,sy)
10. e.addSensor(s)

#add rules
11. e.addRules("move.py")

#add behaviors
12. e.addBehavior(CollisionBehavior("Collision"))
13. e.addBehavior(ImageLookupBehavior("ImageLookup"))

```

```

14. e.addBehavior(TargetFindingBehavior("TargetFinding"))
15. e.addBehavior(StayInBoxBehavior("StayInBox", x1,y1,x2,y2 ))

```

The line 3 specifies the capacity of the situation. If it is -1, then it means an infinite number. Lines from 3 to 7 specify the vertex positions of a region of the situation. Lines from 8 to 10 specify an event sensor. Line 11 adds a rule for controlling behavior functions. Lines from 12 to 15 add the behavior functions. The order of adding behavior function (line 12-15) determines the order of behavior composition. In the main simulation loop, when a character enters the situation, the situation adds a sensor, a rule and behavior functions. Once the characters have a sensor, they sense events happening in the environment. After they sense the events, the captured information is stored in the character memory. Then, it calls the event rule (move.py) to decide which behavior functions should be composed. The rule specification is following.

```

#a : agent , vm : agent's memory

1.  vm = a.getMem()

#get information from sensor
2.  sense = a.getSensor(vm,101)

3.  if sense.event() == 1:
4.      a.setGoal(rx, ry)

5.      a.BehaviorCompose(CollisionBehavior)
      a.BehaviorCompose(ImageLookupBehavior)
      a.BehaviorCompose(TargetFindingBehavior)

      else :
6.      a.BehaviorCompose(CollisionBehavior)
      a.BehaviorCompose(ImageLookupBehavior)
      a.BehaviorCompose(StayInBoxBehavior)

```

The lines 1 and 2 get the information from the sensor. If the sensed information turns out to be “on” (line 3-4), the rule gets a random position from the target region and sets this position

as the goal position for the character (line 4). Then, in the behavior composition process, three behavior functions (*Collision*, *ImageLookup* and *TargetFinding*) are composed to choose the next action (line 5). If the signal is “off”, then other three behavior functions, (*Collision*, *ImageLookup* and *StayInBox*), are composed (line 6). Due to the *TargetFinding* behavior, the character chooses actions that move him (or her) to the target position closer when the signal is “on”. On the other hand, while the signal is “off”, the *StayInBox* behavior keeps him (or her) in the initial region.