GROUP BEHAVIOR SIMULATIOIN IN COMPLEX ENVIRONMENTS

GROUP BEHAVIOR SIMULATION IN COMPLEX ENVIRONMENTS ZHOU BO 2006

ZHOU BO

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2006

Group Behavior Simulation in Complex Environments

Zhou Bo

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering

2006

Acknowledgments

I would like to take this opportunity to express my gratitude to the following people, who have helped me so much throughout my graduate studies.

First and foremost, I would specially thank my supervisor, Dr. Zhou Suiping, for his remarkable guidance and encouragement during my research work in NTU. Dr. Zhou has always been willing to answer my questions and discuss the future directions. He had inspired my creativity with his precise and valuable insights.

I also want to express my appreciation to all staff and technicians in Parallel & Distributed Computing Center, especially Mrs. Ng-Goh Siew Lai, who was so patient to help me with the technical problems and has done a lot of work in building the testbed.

And I believe I owe many thanks to my parents who were very supportive to my academic venture.

I also want to give my thanks to all my friends, who have helped me with the difficulties I met. It is them who always encouraged me to face all the challenges in my life and study.

Abstract

A group is a collection of two or more interacting individuals which maintain stable patterns of relationships, share common goals, and perceive themselves as being a group. Group behaviors are generally complex to understand or control, so it is not easy to simulate them in virtual environments, especially in complex environments with dynamic obstacles. In order to simulate and investigate the features of natural groups and people crowds, many researchers from diverse fields of scientific and engineering disciplines have proposed many different models.

While many methods to simulate group behaviors have been proposed, most of these methods have a polynomial complexity. How to simulate the behaviors of a large group is a new challenge because of the increased computational cost. Group behavior model in some specific domains is also interesting to be investigated. It requires domain knowledge to give group members intelligence to some extent to cooperate with other members and achieve designated goals.

In this thesis, some models for group behavior simulation are discussed. Based on a well-known model, enhancements from two different aspects are investigated. Firstly, an enhanced parallel algorithm with new mechanisms is proposed to simulate large groups more quickly. Simulation results show that, compared with other implementations, the proposed algorithm provides better speedup in generating flocking behaviors. Secondly, motion planning is investigated to give each individual more intelligence. A simple goal-pursuing task is assigned to the leader of a group. Then a self-adaptive probabilistic roadmap method (PRM) is applied for the group leader

in dynamic environments. Hierarchical motion planning is also implemented in a structured-environment. Experimental results show that these two methods are very helpful to make the group behave intelligently in complex environments with dynamic obstacles.

Finally, the thesis also outlines the future work in the area of squad behavior and military operation in urbanized terrain. More sophisticated technologies involved in motion planning are also introduced.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

From bird flocking to fish schooling, seeming to share a single collective mind, the coordinated behavior appears everywhere in the natural world (see Figure 1.1). However, creating this beautiful effect can be extremely difficult. In computer animations or computer games, it is a nightmare to calculate each individual path for a large number of bodies. Within a dynamic environment such as a large interactive computer game it would be very difficult to write individual movement scripts that looked realistic [84]. Although the autonomous intelligent graphical characters can address the above problems, automating the production of these characters is very difficult. Nevertheless, automatic production of intelligent group behaviors has captured the attention of numerous researchers.

Figure 1.1: Bird flocking and fish schooling

Today, automatic production of intelligent group behaviors is no longer restricted to the incredible abilities of nature. In the recent years, there have been many ongoing research efforts to simulate such behavior in virtual environments, computer games, robotics and artificial life. Some examples are listed in Table 1.1.

Although many different methods have been proposed and some are well-performed in specific areas, we still face many problems in some cases, e.g., when the group has a very large number of members. Evidence shows that most of these methods have a polynomial complexity. For this reason, attention has been focused on using parallel model to reduce the computing time. On the other hand, small groups in computer games still face other problems concern intelligent movement to achieve designated goals. Adversaries in groups should make a threat assessment, recruit others and then plan a coordinated attack against the player to increase the fun of combat. All these tasks require a robust motion planning mechanism to help the groups move in complex environments.

Table 1.1: Examples of projects involved in group behaviors

Field	Examples
Virtual environment	Virtual human crowd [27, 50], Virtual birds [58]
Computer games	Squad behavior [76], Synthetic adversaries for MOUT [88]
Artificial life	Particle system [5], Behavioral animation [58], Artificial fish [74]
Robotics	Multiple mobile robots [80, 26, 21]

Recently, some results in the above two directions have been produced. They have been applied to improve the performance and interactive ability of groups in several domains: computer games, robotics, virtual environments, etc...

1.2 Motivation

The aggregate motion of a group is common in nature. Nature provides outstanding models of functioning systems consisting of large numbers of more or less intelligent

elements. Birds in flocks, fish in schools, ants in swarms and sheep in herds all appear united in motion. Why do they flock together, and how do we generate this kind of behaviors in virtual environments?

To explore the nature of group behaviors, some researchers investigated the natural groups such as fish schools and bird flocks. They found that a group was always made up of discrete members yet kept fluid overall motion. Although all members actually act solely on the basis of their own local perception of the environment, the group exhibits an intentional, centralized control. For example, the observed aggregate behaviors of social insects always exhibit a great complexity, while the individual animals behave much simpler.

Over the last few years, the problem of coordinating the motion of multiple autonomous agents has attracted significant attention. A synthetic approach to the study of such group behaviors was pursued by Craig Reynolds [58] in order to develop a realistic-looking animation sequence of a flock of birds. Following the work of Reynolds, several other models have appeared and led to the creation of a new area in computer graphics which was known as Artificial Life [74].

The conventional techniques for group behaviors, however, are not desirable when addressing the new challenges appeared in the following areas:

- Large groups in grand scenes:

  Today's computer animation and computer games contain more and more grand scenes composed of a large amount of groups. The traditional techniques for group simulation mainly focus on the sequential model, and try to improve the model to reduce the complexity. It is definitely important but not enough for some applications requiring very fast production, sometimes even in real-time, because in general, the sequential models are polynomial to the number of group members. So how to generate large groups in real-time is a great challenge.

- Intelligent agents in the group:

  Traditional models emphasize the appearance of the group, banking, fleeing, aligning, splitting, reforming, etc. Now it is also important to make each agents intelligent, or even sentient in the group. To adopt practical tactics appeared in natural group motion or human crowd motion, a hierarchical action model is needed for each agent to coordinate the locomotion with others and pursue designated group tasks. To imitate humans, the advanced virtual human crowd should take these steps to be intelligent: perceiving, thinking, interacting, learning and evolving. So how to make a group move intelligently with some information is another challenge.

- Robotics with physical bodies:

  In the field of robotics, group behaviors of multiple mobile robots are still under development. When multiple mobile robots are expected to cooperate, they should work together towards a common goal without interfering with each other physically. There are many issues about the environment, goals, and robot architecture that need to be addressed before efficient coordination can be achieved. Dynamic, hostile environments require robust coordination strategies to be developed for the robots team to achieve their common goal. In this field, more complicated problems about heterogeneous multi-robot cooperation are introduced to realize more complex and realistic robot groups.

Most of the computer simulation research works on group behaviors mainly focus on the former two areas. They are commonly based on Reynolds' distributed behavioral model, which is computationally expensive and not feasible for real-time applications if the group is very large. In dynamic environments, intelligent agents in group are also difficult to achieve. As yet, there is still not any practical and extensible method for generating large groups quickly and producing intelligent groups in dynamic environments.

All the above-mentioned reasons provide us a strong motivation and incentive to develop an effective and intelligent group behavior simulation in complex environments.

1.3 Scope and Objectives

Considering the increasing large battle scenes in movies and animations and the tactical team cooperations in computer games, group behavior simulation is indeed a promising and challenging research field. The scope of the research mainly focuses on developing models and algorithms for large groups simulation and intelligent interactive groups in complex environments.

The architecture of a common group simulation system is described in Figure 1.2. The group simulation system consists of three components: group, environment and motion planning. The motion planning component solves the motion planning problem within complex environments. In this system, if the group is composed of a large number of members, it will be difficult to run the simulation in real time. And if the environment contains a lot of dynamic obstacles, it will increase the difficulty of designing an effective motion planning component.

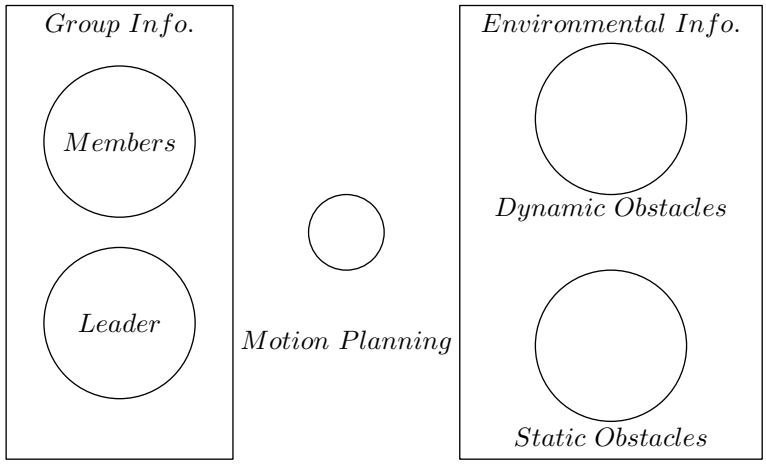


Figure 1.2: Overview of the group simulation system architecture

In this project, the two directions mentioned above are considered. We will have to develop a new parallel simulation algorithm to improve the ability of Reynolds' flocking model to generate large group scenes more quickly. We will also have to investigate how the addition of global information, such as the probabilistic roadmap method(PRM) based on the global environment could enable more sophisticated group behaviors and support global navigation and planning. To this end, a set of new mechanisms will be investigated based on some motion planning algorithms and behavioral rules. Issues to be investigated may include:

- Parallel simulation of group behaviors: to speedup the production of large group simulation, an effective and scalable parallel algorithm has to be proposed and implemented.

- Characterization and communication of the global information about the dynamic environment. Efficient motion planning in dynamic environment is needed for a group to behave intelligently.

- An efficient hierarchical behavioral model for autonomous characters: to generate more complicated behaviors, global knowledge about the dynamic environment can be exploited to provide general guidance for the longer-term actions of an autonomous character, whereas local knowledge influences the shorter-term, reactive actions.

To demonstrate the ideas and mechanisms developed in the research, some sophisticated animation sequences will have to be generated to verify the performance of the proposed computational models.

In our research, the following algorithms will be developed based on Reynolds' flocking model:

- Parallel simulation algorithm:

  Partitioning and communication mechanisms will be improved. Load balancing will also be considered to make the parallel flocking simulation more efficient.

- Motion planning algorithm:

  The PRM approach will be combined with the flocking technique. The group leader will use the roadmap created by PRM to plan the motion of the group based on their current locations and states.

Therefore, the objective of the research is to investigate techniques to simulate realistic and complex group behaviors in a complex virtual environment. The methods should be scalable to support simulating large groups in real time. The proposed parallel algorithm will be implemented in a PC-cluster. The performance metrics of the simulation with the parallel algorithm, e.g., execution time and speedup, will be measured, evaluated and compared with other implementations. It is difficult to form a metric which defines the characteristics of a "good motion planning". Besides some statistical analysis on motion planning, we will also perform qualitative analysis for different situations.

1.4 Organization

The rest of the report is organized as follows: Chapter 2 summarizes related research work on group behaviors in computer animation, computer games and robotics. Two research directions are also discussed to enhance the performance. Having conducted the survey, our solution to quickly generate large groups is given in Chapter 3. Comparisons of the performance with other implementations under different configurations are also given. In Chapter 4, the motion planning procedure is implemented based on PRM to give the group more intelligence. A new self-adaptive PRM method and a hierarchical motion planning mechanism in dynamic environments are investigated

and implemented. Chapter 5 presents a summary of this project as well as its contributions, and outlines the future works and their potential solutions.

Chapter 2

Related Work

2.1 Basic Problems

Group behavior is a concept from the natural world. In the 17th century, an anonymous poet described the beauty of this behavior and wondered how it came:

> *...and the thousands of fishes moved as a huge beast, piercing the water. They appeared united, inexorably bound to a common fate. How comes this unity?*

The answer to this question was revealed gradually. At first, a flock was thought to be a decisive sign of life, some noble formation only life could achieve. Via the flocking rules discovered by Craig Reynolds [58], biologists concluded that the flocking of real birds and fish must emerge from a similar set of simple rules. In the chapter of Hive Mind of *Out of Control* [34], a flock is described as "an adaptive trick suitable for any distributed vivisystem, organic or made" [34].

For a bird to participate in a flock, it must have behaviors that allow it to coordinate its movements with those of its flock members. These behaviors are not particularly unique; all creatures have them to some degree. In sociology, a group is "a collection of two or more interacting individuals with a stable pattern of relationships who share common goals and who perceive themselves as being a group" [23]. To individual members, they always join the group for security, status, self-esteem,

power or goal achievement [81]. The sociological and psychological aspects of crowds are important to understand the behaviors in group, because "we have a very imperfect knowledge of the human heart if we do not also examine it in crowds" [62].

Group behavior is a type of complex motion which biologists always concern with before. In recent years, inspired by the features of group behavior, many research works have been done to simulate the group behavior and to apply the technologies to different areas.

In this chapter we classify some basic research works about group behavior simulation into three different fields, computer animation, computer games and robotics. The attempt to explore the benefits of group behavior in different areas was presented in these basic works. They can help classify why it is important to do further research in this area. They are also fundamental to continue the investigation, especially the flocking model of Reynolds.

Additionally, for discussing the enhancements in these fields, two directions are distinguished: to make the group larger and to make the group smarter. These works, which consider large system and intelligent agents, have made progress to intelligent group behavior in complex environments. The challenging problems in intelligent agents include the motion planning for agents to make them move intelligently, the emotional intelligence to make them more believable and the synthetic characters to help them perform some strategically cooperation tasks. They provide background about our methods to enhance the group behavior simulation proposed in later chapters and also the directions to be investigated in future.

2.1.1 Computer Animation

Birds flying in a flock demonstrate the ability to synchronize their flight patterns with seemingly little communication. To simulate this behavior for computer animation, it is better to rely on a simulation model to coordinate the movement of the whole flock, rather than explicitly scripting the behavior of each bird.

Eurythmy by Susan Amkraut and Michael Girard contained the first procedural animation of flocks when it was shown at the Electronic Theater of SIGGRAPH '85 [3]. In the film, a flock of birds flies up and passes between a series of columns while avoiding collision with their flock members. The basic idea of this film is the force-field driven animation. There are rejecting forces around each bird and around obstacles. A certain force is also applied at each point in environment. It means that the environment that birds move in is designed using force-fields to help the birds avoid obstacles. This is a proven useful method to steer the flock. However, the force-field model is difficult to find suitable parameters for the whole flock to steer away from any obstacle because the model depends on the shape and size of obstacles. In addition, birds will be confused by obstacles with overlapping force fields [65].

Reynolds was not the first person to try and model flocking behavior, but he proposed the most popular distributed behavioral model to simulate this kind of emergent behaviors. In 1987 he presented his paper entitled "Flocks, Herds, and Schools - A Distributed Behavioral Mode" at the ACM SIGGRAPH 1987 Conference [58]. This paper has become the basis of the majority of work on modeling group behaviors, and was cited in numerous other papers. In this paper, he investigated behavioral model to represent the bird-like objects that he called "boids" in simulation. The boids are similar to the point masses used in particle systems except that each boid has an orientation. They maintain position and orientation in the flock by balancing their desire to avoid collisions with neighbors, match the velocity of nearby boids, and move towards the center of the flock. Each boid uses only information about nearby boids.

At every step of the animation, each boid would perceive the environment around it, and use this information to calculate three force vectors [58]:

(i) **Collision Avoidance**: avoid collisions with nearby flock members

(ii) **Velocity Matching**: attempt to match velocity with nearby flock members

(iii) **Flock Centering**: attempt to stay close to nearby flock members

Each vector contains the priority to reflect how urgent it is considered. At each update point, each boid makes an independent decision based on the combination of above vectors. Because of these simple rules, boids appear to be acting in group pattern with collective consciousness.

Reynolds' work convincingly demonstrates that the global flocking behavior could emerge from the application of simple local rules to each boid. Using the local rules, each boid makes its own independent decisions based on the information it perceives about the environment around it. Reynolds also introduced two obstacle avoidance schemes, force-field around obstacles and active steer-to-avoid [59]. Some examples of Reynolds' flocking model can be seen in movies, such as the stampede of the antelopes in The Lion King, the herds of dinosaurs in Jurassic Park, and the massive battle scenes in The Lord of the Rings.

Tu and Terzopoulos extended Reynolds' flocking system to create more realistic, self-animating characters through physical modeling of the fish [78]. They created a virtual marine world with fish that hunt, flee, mate, and wander. An important contribution of their work was that the fish decide what action to take by use of a form of finite state machine, the intention generator which selects between various intentions (see Figure 2.1). They also proposed the idea of cognitive modeling, which controls how characters gather knowledge and how they act [18]. Cognitive modeling enables heterogeneous flocks – different behaviors for different flock members.

In recent works, Bouvier [10], Brogan and Hodgins [11, 12] expanded Reynolds' work by applying similar control algorithms to dynamically simulated characters. They used particle systems and dynamics respectively for modeling the motion of groups with significant physics. Helbing's team described methods to simulate the movement of pedestrians. A model of pedestrian behavior was used to investigate the mechanisms of panic and jamming by uncoordinated motion in crowds [27]. Musse

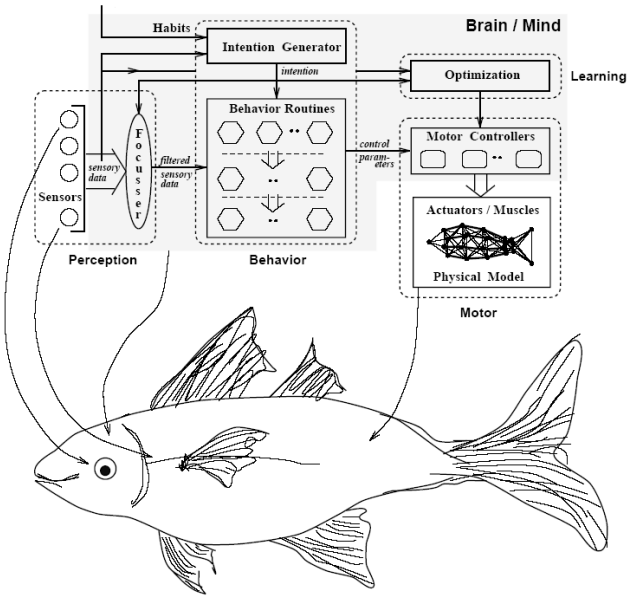


Figure 2.1: Functional diagram of the artificial fish [75]

and Thalmann presented a hierarchical model to describe crowds with different levels of control [50].

"The modeling and simulation of living systems for computer graphics resonates with an emerging field of scientific inquiry called artificial life, or Alife" [74]. Since Reynolds's landmark "boids" experiment bridged the gap between artificial life and computer animation, behavioral modeling, as a major Alife approach (see Figure 2.2), has been proven effective for group behavior animation. Behavioral animation techniques now represent a major trend in computer animation, and they have been applied extensively in the motion picture industry.

2.1.2 Computer Games

Compared with characters in computer animation, characters in computer games need more AI techniques to acquire the interactional freedom in a virtual environment. Automatous characters (NPCs – Non Player Characters) should move around the environment smoothly and interact with players intelligently. In the field of computer games, it is more important to design a control structure that results in agents ap-

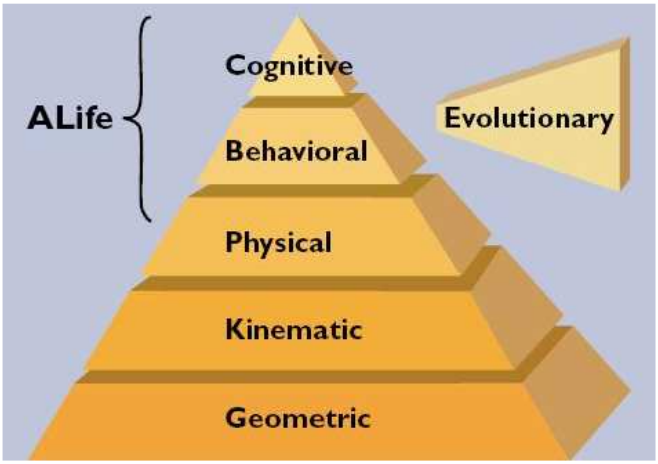


Figure 2.2: Alife modeling and the CG modeling hierarchy [74]

pearing to move intelligently. The well-cooperated combatants can raise the player's interest in games.

A typical group behavior in games is squad behavior (see Figure 2.3). It requires cooperation among adversaries to increase the fun of combat. It also requires the cooperation among player's group to realize the intelligent group attack. "Adversaries can make a threat assessment, recruit others and then plan a coordinated attack against the player" [76].

Reynolds' flocking behavior is very successful in a variety of commercial titles. It provides a powerful tool for unit movement and for creating realistic environments the players can explore [86]. In a real-time strategy or role-playing game, flocking can be used to allow groups of animals to wander the terrain more naturally and for realistic unit formations or crowd behaviors [87]. For example, a group of soldiers can be made to move realistically across bridges or around obstacles, such as boulders in the environment. Alternatively, in first-person shooter games, monsters can wander in a more believable fashion, avoiding players and waiting until their flock group is large enough to launch an attack [72].

In the area of first-person shooters, Half-Life(Sierra) from Valve Software employs the most advanced AI in the field. Enemies exhibit squad and flocking behavior

Figure 2.3: Squad behavior is a key to success in Unreal Tournament [19]

simulated from real-life animal flocking patterns, and organize attacks with fellow members by assessing their chances of winning in an all-out battle. Half-Life is seen as something of a turning point in computer game AI. Traditional game AI is a set of hard-coded "if-then" decisions, such as, "If there is a bad guy in this room then shoot at him." "Valve took another approach, designing a module-based AI system that provides practically infinite flexibility and monsters growth potential" [76].

In Half-Life, the cooperated group behaviors are achieved by two mechanisms, tasks and schedules. "A task is a specific action performed by a monster, like playing a sequence, running to take cover, or throwing a grenade. A schedule is a set of tasks performed in a specific order, like finding a path to a corpse, running to the corpse, and performing a specific animation" [15]. Combining scripting for fine detail control with a much more general architecture led to the ground breaking AI in Half Life. It was further extended in Counter Strike [68] which was based on the same engine.

Besides Half-Life, Unreal(Epic) and Enemy Nation(Windward Studios) also used flocking [85]. Unreal used flocking for many of the monsters as well as the other creatures like birds and fish. Enemy Nations used a modified flocking algorithm to control unit formations and movements across a 3D environment [86]. In summary, flocking is currently used widely in games where groups of animals or monsters are needed. It is a relatively simple algorithm and only composes a small component of a game engine. However, it makes a significant contribution to games by making

an attack by a group of monsters realistic and coordinated. It is ideal for real-time strategy or first-person shooter games that include flocks, fishes or herds.

2.1.3 Robotics

Inspired by the simulation models of group behaviors, researchers in robotics have studied the movements of multiple mobile robots for a few years. A system populated by many simple robots is preferable to one made up of just one large and powerful robot. In an extensive survey [13], a multiple-robot system is considered easier, cheaper, more flexible and more fault-tolerant than a single powerful robot.

In all the related studies, the most important part is the motion planning for the robot crowd. It must consider the collision avoidance, the decision of trajectory and the behavior types of the whole group. This problem is challenging because of the high degrees of freedom (DOF) involved when the number of members in robot crowd becomes large.

Generally speaking, there are two main approaches to the planning problem for multiple robots: centralized approach and distributed approach [17]. According to surveys of motion planning algorithms [39], the centralized planning typically considers all robots and their degrees of freedom altogether and therefore usually must deal with a high dimensional search space, while the distributed approach considers each robot individually, making it plan and adjust its paths in parallel with other robots until feasible paths for all robots are found.

Coordination of multiple manipulator systems is based on centralized control. Central control requires extensive communications between agents and map knowledge of the environment. Since the size of the configuration space is overwhelming, it is impractical to search the C-space systematically. Therefore, most motion planners with the centralized approach use a randomized algorithm to achieve probabilistic completeness. Early randomized planners use potential fields built in the workspace

to guide the search in C-space and use random walks to escape local minima. Typical planners with this approach are the RPP (Randomized Path Planner) [6] and PRM (Probabilistic Roadmap Method) [33]. "A common feature of the randomized planners is that they are probabilistic complete, which means that if there exists a feasible path and there are no time limits, then the planner will be able to find it eventually" [41].

Distributed robotic systems (DRS), multi-robot systems based on instinctive responses and cooperation other than central control, have many advantages over centralized systems, especially when high reliability is required. Previous experimental works on multiple mobile robots include ACTRESS (ACTor Based Robots and Equivalent Synthetic Systems) developed by Habib et al. [25]. ACTRESS, as an autonomous and decentralized robotic system, does not only have mobile robots, but also any kind of robotic system or computers. Distributed control is a very promising framework for the coordination of multiple robots.

In [70], Sugihara and Suzuki gave a method for motion coordination of a group of mobile robots. Each robot plans its motion individually based upon a defined goal and detected position of other robots. This method is fully distributed in that sense and shows that intelligent behavior can emerge from simple individual behavior. Although the proposed algorithms are shown to work quite well, most of them are based on the assumption that each robot can detect the distance from the farthest and nearest teammates.

Inspired by Sugihara and Suzuki's work, Unsal and Bay [80] used simple geometric properties of desired shapes to create repulsive or attractive force fields. They used simple behavioral rules for individual agents to achieve self-organization, instead of controlling them centrally. Based on feasible navigation and communication assumptions, algorithms defined in this paper use the notion of switching distance. This self-organizing robots system are adaptive. When an agent fails, the gravitation field definitions change, and all agents move to their new position consequently.

In [21], a flocking model was used for a set of mobile robots to follow a leader robot. The followers in the flock just follow the leader wherever it goes while keeping a predetermined formation. In [26], an investigation of flocking by teams of autonomous mobile robots using principles of Swarm Intelligence was presented. A leaderless distributed flocking algorithm (LD) was proved to be more conducive to implementation on embodied agents than the established algorithms used in computer animation. It demonstrated that a group of real robots executing LD with emulated sensors could successfully flock and that systematic characterization of real robots parameters was achievable (see Figure 2.4).

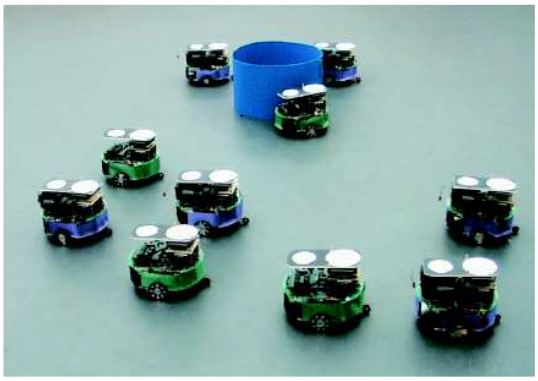


Figure 2.4: The Moorebots flocking arena [26]

Besides motion planning, there are also some other problems in a group architecture for mobile robots. A heterogeneous group of robots introduces complexity since task allocation becomes more difficult, and agents have a greater need to model other individuals in the group. The communication structure is also important to determine the types of interaction among agents.

2.2 Enhancements

As mentioned before, conventional techniques for group behaviors are not desirable when addressing the new challenges appeared in some areas. Therefore, researchers try to make the related group behavior techniques more powerful to be used in computer

animation, computer games and robotics. Generally speaking, there are two main problems that must be considered. On one hand, large scenes composed of many flocks or crowds require the system to quickly generate large groups, even in real-time. On the other hand, the interactive requirement of a system demands each agent to perform intelligently in coordination with others.

Following is the work that has been done to address the two main problems respectively.

2.2.1 Large Groups

While many methods to simulate group behaviors have been proposed, these methods are usually applied to sequential computing. Since most of these methods have a polynomial complexity, it is difficult to simulate a large group in real-time using these methods.

In Reynolds' flocking system, the nature of the system is that efficiency was always going to be a large problem. When each boid might well have to consider every other boid in the system there was great potential for it to be a highly inefficient system. "The complexity of the flocking algorithm described is basically O(N^2)." [58], however some improvements have been seen in efficiency. Flocking simulation can also be regarded as a special kind of the N-body simulation. The N-body simulation is to simulate the movement of a set of bodies (or particles) under various types of forces. Parallel N-Body simulation of particle systems have been widely investigated based on a class of tree-based methods. Barnes and Huts [5] had investigated splitting the world space down into nested cubes. Warren and Salmon [83] and Liu and Bhatt [42] had implemented Barnes-Hut's O($NlogN$) method using the message passing programming paradigm. Some parallel implementations of the fast multipole method (FMM) [24] have also been investigated by Blelloch and Narlikar [9]. Lorek and White [43] implemented Reynolds' model on a Meiko Transputer System (see

Figure 2.5). In their implementation, each processor deals with a fixed number of birds. With P processors, its communication scheme requires P-1 steps to collect the data from other processors. To reduce the computational load, the boid needs only consider the other boids in it's vicinity.

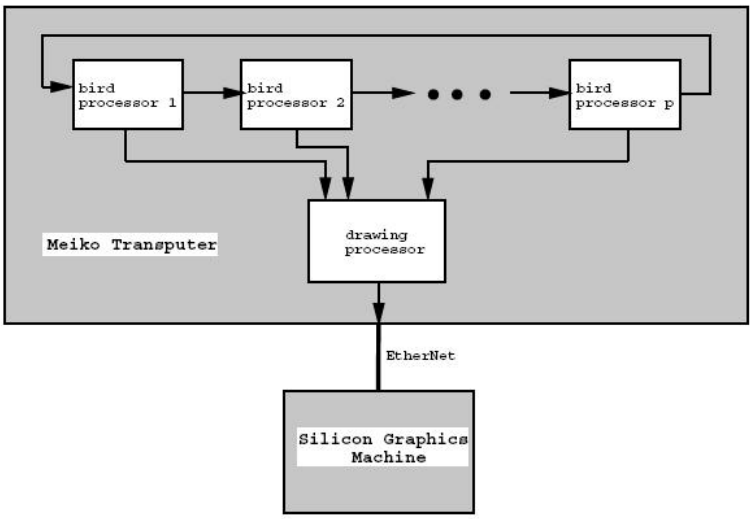


Figure 2.5: The architecture of Lorek and White's implementation [43]

Reynolds also exploited "the use of a lattice-bin spatial subdivision technique to accelerate the locality queries needed in an interactive group of characters" (see Figure 2.6) [60] in "Pigeons in the Park". Many measures were tried to improve the efficiency of the system. These included the method of calculating the flock center by the system once per frame rather that by each boid every time it moved.

Given the power that is now available in average personal computers, large group simulation now cannot achieve the real-time performances. Because of the complex feature of this kind of simulation, improvements through hardware are more important than those from software consideration. Parallel and distributed architecture seems hopeful to solve this grand problem. It is also important to design deliberate model to fully explore the power of new computational architectures.

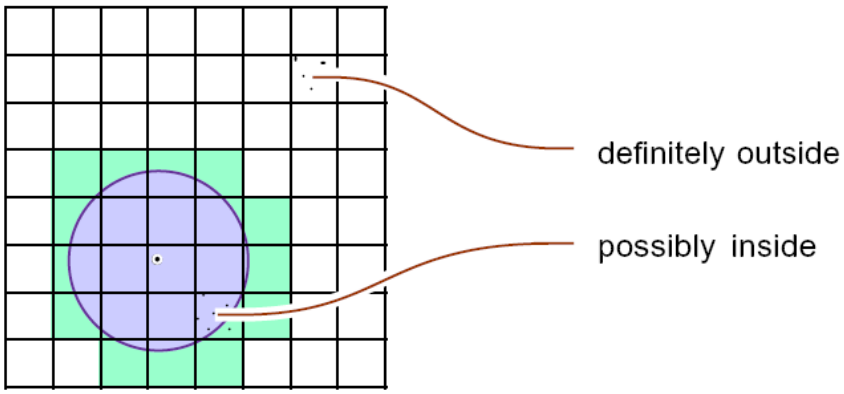


Figure 2.6: The locality query identifies different bins [60]

2.2.2 Intelligent Agents

"Intelligent agents are software agents that perceive their environment and act in that environment in pursuit of their goals." [72] A common theme in many of the military and academic papers is Intelligent Agents (IAs) that implement "smart behaviors" or attempt to mimic the traits of commanders [66]. To produce an impressive behavioral animation, IAs also play an important role to help the animator from specifying details of each individual movement. IAs are also becoming more common in commercial Role Playing Games (RPGs) to implement Non-Player Characters (NPCs).

Artificial intelligence in games used to be a laughing matter, but with the development of Deep Blue finally beating Kasparov, the field is enjoying a renewed sense of importance and prospective success. In most early games, attack algorithms developed for enemies used to be limited to simple 'if-then' statements. But now, through the use of some advanced technologies, such as neural network and genetic algorithms, they become much more intelligent.

An IA may be created using scripting, fuzzy logic, finite state machines, fuzzy state machines and genetic algorithms. These technologies are responsible to form a control

system that results in an agent moving intelligently. In the field of group behaviors, this enhancement is very important. For each agent in the group, it will provide the mechanism to deal with path planning and emotional responses to environments. For the whole group, it will give agents the ability to behave with some specific knowledge, and cooperate with other agents to perform certain specific tasks (such as the ability to hide, ambush and fight). The following problems are challenging to researchers in intelligent agents.

2.2.2.1 Motion Planning

As mentioned above, motion planning is a traditional problem in the area of robotics to plan the movement of robot arms and robot vehicles. The basic motion-planning problem is to compute a collision-free path for a moving body from start to goal positions. In recent years, it has been studied extensively, both in computer animation and game community, and in virtual environments. Computer-generated entities must plan their paths amidst obstacles and other entities to move naturally and intelligently. In this part, we only consider motion planning approaches to computing the simultaneous motion of multiple agents. This spans the works done in computer animation, robotics, games, and crowd simulations.

In the computer animation community, the most popular approach for simulating group behaviors is to use flocking. Reynolds' flocking model used only local rules to describe the behavior of the boids in a group. In [59], Reynolds extended the technique to include autonomous reactive behavior. The idea is that boids steer themselves in such a way that they avoid collisions with other boids and obstacles in the environment. At the same time, they try to align themselves with other boids and try to stay close to other boids. In open areas this leads to rather natural group behavior as that can be observed in bird flocks or fish schools. When a goal is designated to the boids, they will move toward the goal together. However, there is a big drawback of this approach. Without a long-term planning based on global

information, the boids act based on local information can easily get stuck in cluttered environments. Also, lacking the concept of coherence, the combined steering behavior can easily lead the group to split up.

Another widely used technique is grid searching. It is considered to be the most straightforward form of path planning. The environment is divided into a grid that can be searched for a free path using A* like approaches [63]. Using a navigation function, each agent tries to find a path through the grid while avoiding collisions with others. This method can easily lead to unnatural motions, so it needs some optimal techniques. This problem gets worse when some constraints, like the fact that vehicles have a bounded turning radius, must be considered.

The social potential field technique [56] defines potential force fields between members of the group. Desired behavior is then created by defining the correct force fields. However, the same problem as in flocking arises because only local information is considered.

Kamphuis and Overmars [31] developed a novel method for planning the motion of a coherent group. They plan the motion for the group as a whole rather than for individual members. The new method consists of the following steps: First, a deformable rectangle is used to represent the group shape. Second, a path is planned for the deformable shape. Third, the internal motion of the members inside this deformable rectangle is calculated using group potential field techniques. Fourth, the global and local paths are combined to form the final motion of the group. Although the approach guarantees coherence, it lacks completeness. There are situations that existing paths are not found by the approach due to the choice of deformable shape. The approach also generates unnatural behavior when a group enters or leaves a narrow passage.

Probabilistic techniques have received a lot of attention in robot motion planning in recent years. One of the dominating probabilistic techniques is the probabilistic

roadmap approach (PRM) [32]. In such an approach, a random roadmap is built in the C-space in a preprocessing phase and to try to answer planning queries at a later time as quickly as possible. As mentioned above, there are two basic approaches. Centralized approaches treat different robots in a group together as one large robotic system with many degree of freedom [64, 71]. Unfortunately, if we assume each robot has two degrees of freedom: its position on a floor surface, so the total robotic system has 2n degrees of freedom when there are n robots. The complexity of the approach is rather intensive to the size of the robot crowd. Also, centralized approaches require that the number and type of units are known beforehand which is not a realistic assumption for some applications. Compared with centralized approaches, distributed approaches first compute the individual motions of the robots and then try to coordinate these motions over time. It enables the planning of motion for a larger number of units [40]. These approaches are faster but can lead to deadlock because of no central coordination. They can also fail when the number of robots grows largely and result in incoherent motion.

Despite the success of the PRM, work has mostly concentrated on static environments. There are only a few contributions related to changing environments. In [35], the randomized motion planners explore the *configuration*$\times$*time* space for obstacles moving along known trajectories. In [29], a roadmap planner is designed to operate in dynamically changing environments. The planner allows a partial but fast dynamic update of the roadmap to answer path queries as fast as possible.

Recently, Bayazit, Lien and Amato [7] have explored the benefits of combining flocking techniques with roadmap-based path planning methods (see Figure 2.7). The units use the roadmap created by PRM to guide their motion toward the goal while they use flocking to act as a group and avoid local collisions. This indeed leads to better global navigation and also supports more sophisticated group behaviors than the single flocking method.

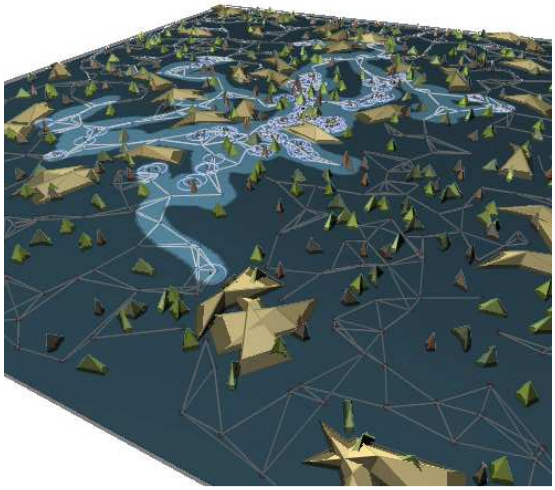


Figure 2.7: Covering behavior implemented bye Bayazit [7]

Crowd simulation investigates the movement of large numbers of persons in a virtual environment. This research area has received vast amounts of attention over the last few years, such as [50, 79]. Unlike computer animation and robotics, the area has a different goal. The global idea behind crowd simulation is to have virtual crowd behave in a natural way, interacting with each other, based on some social rules. The emergent behavior of the units is then studied. Related to crowd simulation is pedestrian and traffic flow simulation. Helbing uses forces related to physics to simulate the behavior of pedestrians and traffic [27](see Figure 2.8). These are related to the social potential fields mentioned above, and have similar drawback for our setting. Often formations are a good way of maintaining coherence. Balch and Hybinette used social potentials to generate scalable formations for mobile robots [4]. They devise a number of forces (potentials) to steer the robots, for example an obstacle avoidance and a formation keeping force. Formations are also very important in games. For example, troops should move in formation over a battlefield. Pottinger describes how these paths can be generated and implemented in two articles [53, 54].

Though a lot of researches have been done on motion planning, many problems still remain unsolved. Some of the challenges and possible directions are listed in [51].

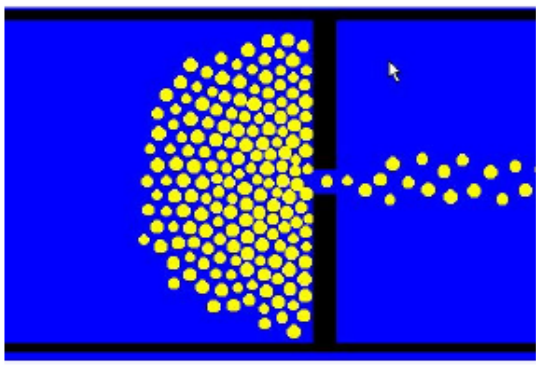


Figure 2.8: Simulation image of Helbing's model [27]

In chapter 4, we will discuss the challenges in detail because of the dynamic changes in the environment.

2.2.2.2 Emotional Intelligence

It is difficult to create believable emotions in animation since emotion is an internal feeling. But behavior can be simulated. Behavior is defined to be "the response of an individual, group, or species to its environment." It is behavior that conveys emotion to a viewer. In a group, each agent's emotion changes in response to its perception of the environment. Based on the emotion, the agent moves in a particular manner which conveys that emotion [8]. For example, if a person runs through a flock of sheep, they will move in order to avoid the person. This reaction is driven by fear, a kind of emotional stimulus, and is communicated among the sheep. Picard claims that "in order to have a really intelligent agent we need to focus on emotion as well" [52].

Over the last years, there have been increased interests in computational model of emotions. Minsky [47] was one of the first to emphasize the importance of emotional intelligence. After him, Picard focused on recognizing emotions and the influences [52]. Based on Izard's Four Types of Emotion Elicitors [28], a model was proposed to synthesize emotions and some of their influences on behavior and learning [82]. These models have been integrated into architectures for control of several agents, in a variety of environments. Emotions play an essential role in rational decision making,

perception, learning, and many other cognitive functions. If we want computers to be genuinely intelligent, to adapt to us, and to interact naturally with us, they will need the ability to recognize and express emotions, to have emotions and have what has become to be called emotional intelligence [22].

This topic within the computer games field could have several meanings. Affecting the emotions of users, expressing emotions in NPC's or modeling emotions and their affects on NPC's behavior. It is the last two of these options, that especially in combination, could be a powerful tool for increasing the levels of user's belief in the NPC's intelligence.

The FEAR engine is "a language independent open-source project providing portable support for the creation of genuine AI within realistic simulated worlds. It stands for Flexible Embodied Animation aRchitecture. It attempts to bring the best of AI to games, and the best of games environments to AI." [73]. In [14], Alex Champandard, who is a key member of the FEAR development team, discusses AI in games as a whole, but with specific reference to the FEAR engine. In the Chapter of Under the Influence of *AI Game Development* [14], Champandard discusses the modeling of emotions in NPCs, using the FEAR engine as an example. The modeling of emotions can be done in many ways. In "Creating emotions as Finite States" [14], he suggests a completely connected Finite State Machine, with transitions between states meaning that "two complementary emotions may not be active at the same time ", for example, *Surprise* and *Anticipation*. Because the graph is fixed, any flexibility in the system comes from a system of design-settable values for each state governing *Precision*, *Power*, *Delay* and *Accuracy*. For any given emotional state these four values are adjusted between 0 and 1. In turn they adjust how the NPC acts.

Whilst this modeling of emotions to affect the behavior of an NPC is academically very interesting, it is perfectly possible that an NPC might well be modeling emotions and actively using them to modify how it is behaving, but the user may not interpret

what they see as emotions. The addition of some simple facial expressions might well significantly help to increase user's interest.

In game industry, the huge advance Valve Software made was including a notion of self-preservation as well as "emotional states" for each enemy, i.e. scared, aggressive, confident, etc. For instance, if significantly outnumbered, the creature might panic or retreat to find back-up [68].

In the Synthetic Characters Group at the MIT Media Lab, headed by Professor Bruce Blumberg, a pack of virtual wolves is used to understand the nature of intelligent behavior. In [77], research in social learning is discussed. There is an essential mechanism by which the wolves form social relationships. When each wolf performs any action, it does so in an emotional state. It has its own emotion and recognizes all of the other wolves. The emotion at a given time is dependant not only on the external stimuli, the internal registers of the wolves but also on the influences of the wolves they are interacting with. Finally, the emotional association influences its current emotional state when he encounters a former wolf again. This simple mechanism allows the wolves to remember each other and pick up where they left off.

The emotions that can be expressed are based on the Pleasure- Arousal-Dominance dimensional model presented by Mehrabiean and Russell [45]. The emotional state of a creature is based on a point in 3D space where the dimensions are Arousal, Dominance and Pleasure (see Figure 2.9).

An emotional system is always the highest level in the framework of behavioral architecture. Virtual agents sense the environment and then the emotional system trigger the possible behaviors. At last, a synthesis mechanism is used to reconcile conflicting behaviors.

2.2.2.3 Synthetic Characters

Struggling to exhibit intelligence, characters in recent games have progressed to include some limited forms of situation-based reasoning, communication, and coopera-

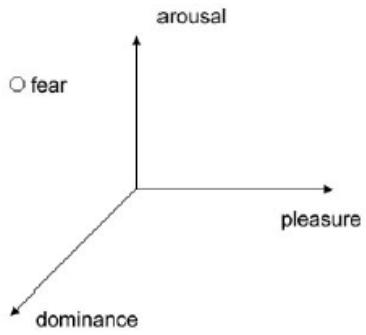


Figure 2.9: The 3D emotional model [45]

tion. "The continued improvement in the intelligence of synthetic characters should lead to significant improvements in game play as well as new gaming experiences" [38].

The actual behavior of artificial entities is especially important in the context of simulations designed for training personnel or evaluating tactical situations. In such simulations, the behavior of agents will have an important effect on the final outcome. Unrealistic agent behavior, e.g., in the form of very limited or even extremely advanced intelligence, can result in poor correspondence to real-life situations. Agent behavior in a sophisticated simulated environment can be very complex and will involve many collaborating or adversary entities. Intelligence can be employed at very different levels. A simple example is a team of agents that has to go from one position to another trying to minimize travel time and keep out of trouble. A more complex example of intelligent behavior can include the decision to cancel the mission of a group of entities and relocating them as a backup for another group. An even more complex situation would involve several adversarial teams, each trying to achieve different goals.

Modern tactical simulators often require the representation of complex autonomous behaviors within a realistic setting. The idea is to ask questions like "what would happen if ..." for a team of agents, in the context of a real environment and potential events. This challenge is the focus of the work addressed in [20] where the authors

consider how broadly defined goals can be used by teams of simulated autonomous entities to achieve the goals. They consider a combination of individual and social control schemes as a way of representing complex behaviors without providing precise prescriptive directions.

In [20], the conceptual issues which arise in this key area of simulation are discussed, and some design principles and a practical implementation are presented. They propose a novel approach to automatically control the motion of synthetic agents in pursuit of broad goals, by combining reinforcement learning, social potential fields and imitation.

MASSIVE [44] is the computer animation and artificial intelligence software developed by Weta Digital to create the award winning graphics for the Lord of the Rings trilogy, particularly the battle sequences, in the series of films directed by New Zealander Peter Jackson. The crowd system allowed every animated character in an animated army to respond individually to its surroundings. If such a character encountered an enemy character it was programmed to fight in a specific style until it either killed the enemy and went looking for another, or until it sustained a specific number of hits and fell down "dead" [61].

Monster behavior based on player's actions moment by moment: In Half-Life, monsters might advance only when it makes sense to. They assess how much health the player may have, where the player is heading, how many of their own kind are left in a room, and whether they have enough health themselves to fight. Such conditions and others dictate whether a monster will chase, attack, or retreat. While in other games monsters are basically suicide squads, in Half-Life monsters do not want to die [76].

A major contribution of Laird to the field of game AI is the SOAR architecture [67] which he has used to implement various projects. In SOAR, all long-term knowledge is encoded as production rules, while the current situation (perception, situational

awareness, mission information, and goals) is encoded in declarative data structures comprising its state. The rules in SOAR match against the state and propose operators to apply to the current situation. Primitive operators send commands to the motor system. Complex operators are dynamically converted to subgoals. Then, the subgoals are pursued once again by selecting and applying operators in subgoals. Eventually, the primitive operators finish the primitive actions (see Figure 2.10). Recently, belief and goal maintenance mechanisms are also included in SOAR. These mechanisms improve the ability to develop a complex behavior system quickly. In [88], six requirements for synthetic characters for MOUTBots are described. The bots use the commercial computer game Ureal Tournament (UT) to define, implement and test the requirements. The cognitive component of the synthetic adversaries is implemented in the SOAR architecture.

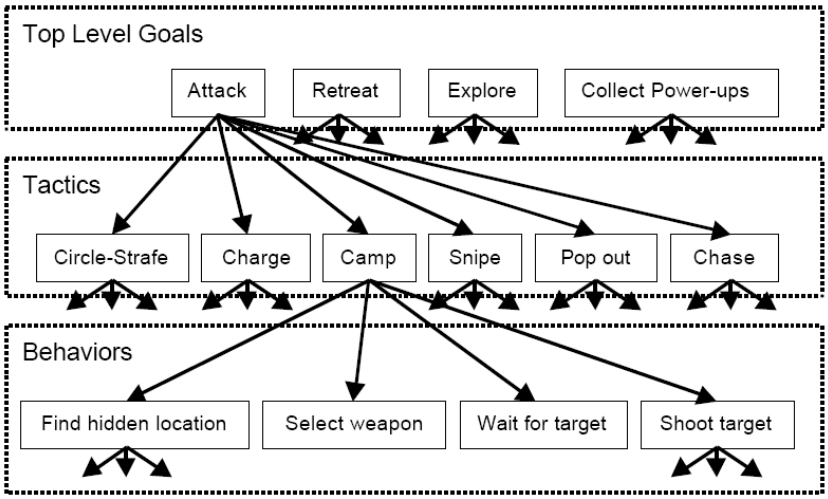


Figure 2.10: A portion of a sample operator hierarchy for an action game such as Quake II [46]

2.3 Summary

This chapter gave a brief review on related work about group behaviors in three different fields. Firstly it introduced the basic concepts of group behavior and the understanding of biologists. Then, some basic works in computer animation, computer games and robotics were discussed. The attempt to explore the benefits of group behavior in different areas was presented in these basic works. They are fundamental to continue the investigation, especially the flocking model of Reynolds. At last, two main challenges were discussed and enhancements in two directions are presented. These works, which consider large system and intelligent agents, have made progress to intelligent group behavior in complex environments. However, methods reviewed above only deal with a part of the issues. They cannot fully resolve many problems when designing realistic and complex group behaviors in virtual environments.

In general, group behavior in complex environments is still an immature research topic and also a great challenge. This report and the future work of this research will focus on the two directions mentioned above. To demonstrate the ideas and mechanisms developed in the research, some sophisticated group behaviors will be simulated in various applications, such as computer games.

Chapter 3

Parallel Simulation of Group Behaviors

3.1 Overview

Group behaviors, e.g. birds flocking, are widely used in virtual reality, computer games, robotics and artificial life. Simulating the group behaviors of a large number of moving objects has attracted the attention of researchers in different fields such as virtual reality [50], computer animation [7, 58] and artificial life [78]. While many methods to simulate group behaviors have been proposed, these methods are usually applied to sequential computing. Since most of these methods have a polynomial complexity, it is difficult to simulate large groups in real-time using these methods. In this chapter, we propose a parallel algorithm to simulate the flocking behavior of a large group. The new partitioning and communication mechanisms in the parallel algorithm make the flocking simulation more efficient. Experimental results show that the proposed parallel algorithm provides good speedup in generating flocking behaviors compared with the sequential simulation.

In flocking simulation, the moving objects in a flock are called "boids" as in Reynolds' [58]. When simulating the bird flock, under the repulses from neighboring members and obstacles, each individual boid will be able to move without collision, and find the correct position in the next time step. The global group behavior emerges from the individual local locomotion plus some global attractions.

Reynolds' behavioral model [58] is effective for simulating a small flock, however, it is very time consuming when the flock has a large number of boids. Lorek and White implemented this model to simulate a bird flock on a Transputer System [43]. To achieve good load balancing, they allocated a fixed number of boids to each available processor. Although the model only focuses on the neighboring attractions and repulses, the detection for neighborhood relationships is a time-consuming process because for any boid, its neighboring boids may reside in any processor during the simulation. In their realization, to simulate N boids with P processors, each processor must receive the information about all boids from other $(P-1)$ processors. So the communication cost increases with the number of processors, and the neighborhood relationship detection process only reduces from $O(N^2)$ to $O(N^2/P)$. Hence its scalability is not good.

Instead of partitioning the boids, we partition the whole virtual space into successive partitions and distribute one partition to one processor [89]. Using this partitioning strategy, each processor only needs to get the information from the processors managing the neighboring partitions, and the time for neighborhood relationship detection also reduces greatly. We implement Reynolds' behavioral model with our new communication and partitioning strategies on a PC-cluster. Different communication and partitioning strategies are investigated, and an efficient method for parallel flocking simulation is proposed.

In Section 3.2, the sequential flocking model is given, and the computation components of the model are analyzed. The main factors in the parallel simulation of the behavioral model are discussed, and a parallel flocking simulation algorithm is proposed in Section 3.3. Experimental results are presented in Section 3.4.

3.2 Sequential Flocking Algorithm

Flocking simulation can be regarded as a special kind of the N-body simulation. The N-body simulation is to simulate the movement of a set of bodies (or particles) under

various types of forces. Reynolds proposed a distributed behavioral model, which was based on simulating the behavior of each body independently.

3.2.1 Flocking Behavioral Model

We mainly focus on the leader-following behavior described by Reynolds [59]. In a flocking group, a designated leader knows the position of the goal. Meanwhile, the followers will follow the leader. Suppose the group is moving in a space without any obstacle, there are three basic rules governing the flocking behavior:

- **Separation**: a repulsive force to avoid collisions with neighboring boids. Based on the relative position of the flock members and ignores their velocity
- **Cohesion**: the attraction of the center of neighboring boids. Based on position
- **Alignment**: the desire to fly in the same direction with others. Based only on velocity and ignores position.

The neighborhood of a boid is characterized by a distance and an angle, see figure 3.1. All boids that are outside this area have no effect on the boid. The three rules are only related to the local information of the group (see Figure 3.2). From each of the three behaviorial urges we obtain a vector v_i that suggests which way to steer the boid. The way v that is actually steered is decided by a weighed average of these three vectors:

$$v = k_1 v_1 + k_2 v_2 + k_3 v_3$$

In a simple model all weights would be equal, but in a more advanced and realistic model the weight k_i could vary depending on the situation of the boid.

With these rules, the simulation can only generate the basic flocking behavior. To realize the leader following behavior in a complex environment, additional rules should be used.

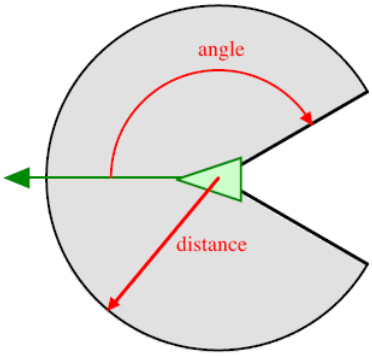


Figure 3.1: The perception area of a boid [59]

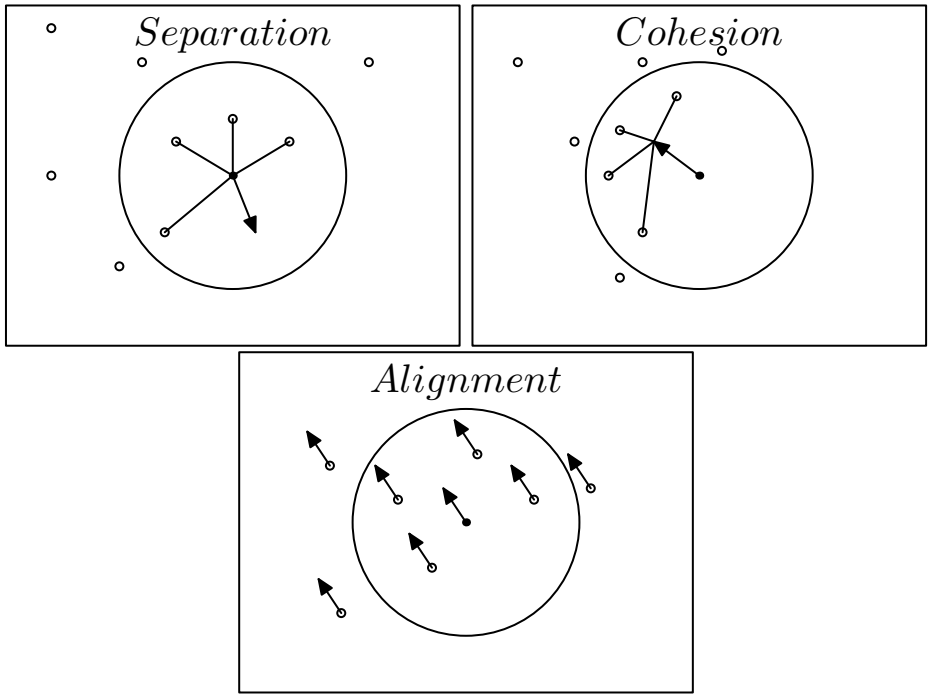


Figure 3.2: Three basic rules of flocking simulation

3.2.2 Environmental Information

When there are some obstacles in the simulated environment, the flock should use two additional forces to avoid collision with obstacles and to explore the complex environment.

- **Obstacle Repulsion**: the repulsion of the obstacles. Each boid will detect the position of the obstacles. If an obstacle is in a boid's vision, the obstacle will repulse it. The smaller the distance is, the stronger the repulsion is.

- **Goal Attraction**: the attraction of the goal on the leader. For the leader-following behavior, it is important to use the goal attraction to guide the leader to achieve the designated target. Following the leader, other boids will tend to approach the position where the leader is currently located, and this attraction is called "leader attraction".

Compared with the three local rules: separation, cohesion and alignment, these two rules need to use the global environmental information (see Figure 3.3).

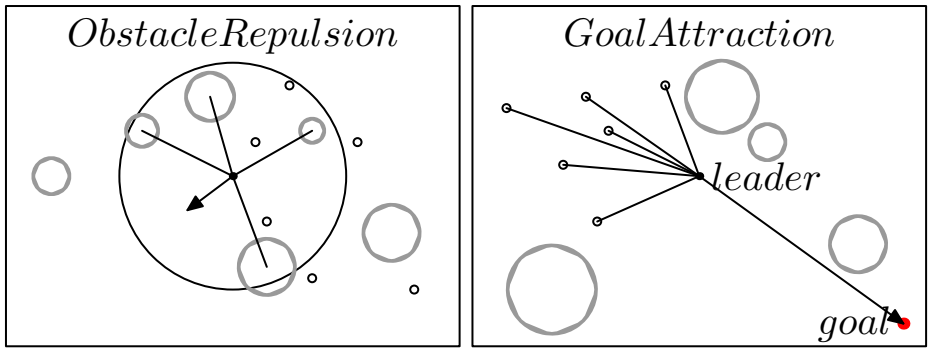


Figure 3.3: Two additional rules of flocking simulation

The boids fly as a cohesive group, and when obstacles appear in their way they spontaneously divide into two subgroups and rejoin again after clearing the obstruction. See figure 3.4 for a series of shots from Reynolds' simulation.

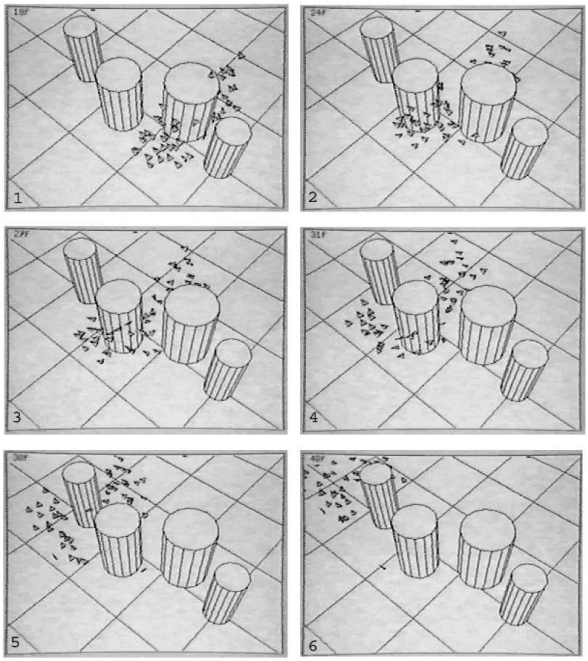


Figure 3.4: A simulated boid flock avoiding cylindrical obstacles [57]

3.2.3 Algorithmic Considerations

The whole process of the simulation is composed of frames. Figure 3.5 shows the algorithm of the simulation in one frame.

```
01. for(each individual boid)
02.     if(the leader achieves the goal)
03.         select the next goal;
04.     else
05.         apply "goal attraction" to the leader;
06.         apply "leader attraction" to followers;
07.     endif
08.     add the force of "separation";
09.     add the force of "alignment";
10.     add the force of "cohesion";
11.     for(each obstacle in the space)
12.         if(in the boid's vision)
13.             add the force of "obstacle repulsion";
14.         endif
15.     endfor
16. endfor
```

Figure 3.5: Sequential simulation algorithm

After this computation, each boid moves to the new position under the integrated force. For smooth movement, the maximum speed of each boid is controlled by a pre-defined value. It is also important to fine-tune the parameters of these rules to generate realistic flocking behaviors.

3.3 Parallel Flocking Simulation

Parallel N-Body simulation of particle systems has been widely investigated based on a class of tree-based methods. Compared with the tree-based methods in the N-Body simulation, the individual-based boid simulation is dependent on both internal and external states in order to interact correctly. It is a kind of emergent behavior, which uses some simple rules based on emergent situation to calculate the individual's

locomotion. Using this model, basic flocking systems have no central control, and the boids are only aware of their local environment. These two features provide a great opportunity to simulate flocking behavior in parallel.

The force calculation part of the sequential algorithm is very time-consuming when the size of the flock is large. If we apply the physics formula accurately as in the N-body problem, the naive sequential simulation takes O(N^2) computations each frame. To reduce the computational complexity, the "vision" of the boid is used to restrict the interactions among boids, i.e., a boid only needs to communicate with its neighboring boids that are within its vision. However, an exhaustive search is still needed to find the neighboring boids.

In this section, we propose the parallel flocking simulation based on Reynolds' flocking model. We only consider the simulation within a two-dimensional simulated space in this paper. When implementing the previous flocking model using parallel algorithms, two main aspects, including communication and partition, should be considered.

3.3.1 Communication

Using data-parallel method, the forces can be computed simultaneously. All processors execute the same statements, but deal with different objects at the same time. To collect the relevant data of the boids on other processors, a processor needs to communicate with other processors. Communication overhead is an important factor of the parallel simulation system. There are two basic communication schemes to exchange the data maintained in different processors:

- **All-to-all communication**: The simulated space is partitioned and distributed to N processors, and each processor sends local information to and receives information from all other processors (see Figure 3.6). Within N steps each processor receives the information about all boids.

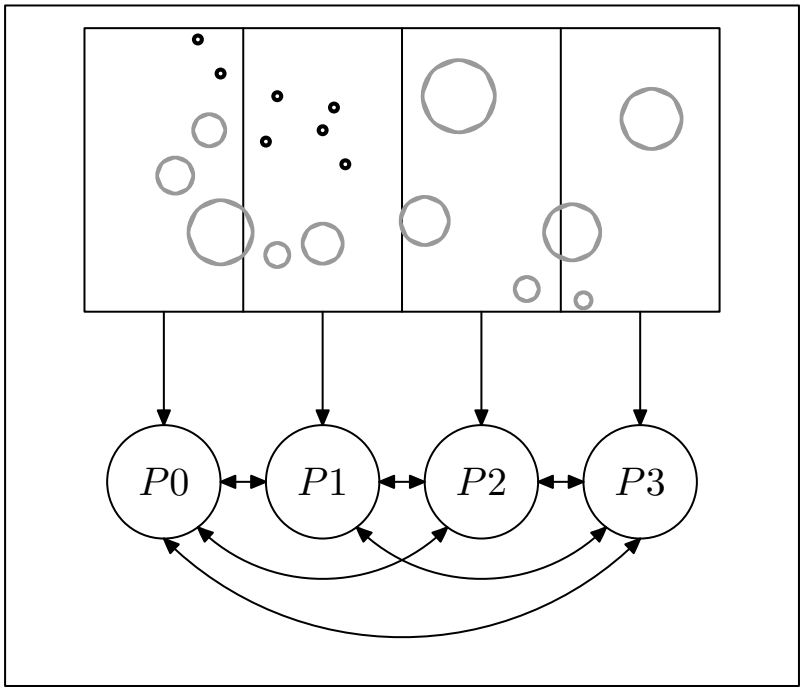


Figure 3.6: All-to-all communication

Under many circumstances, a processor does not know whether other processors need its local information, and does not know the location of the information it needs. So, each processor must communicate with all other processors in order to collect the necessary information. Although this communication scheme is inefficient, it can give the local process a global view. Note that the receive buffers in each processor will increase with the number of group members.

- **Near-neighbor communication**: The simulated space is partitioned and distributed to N processors, and each processor sends local information to and receives information from its logically neighboring processors (see Figure 3.7). Logically neighboring processors are those processors that manage neighboring spaces in the simulated environment.

In a static environment, we suppose that the obstacles are fixed. Based on the space-dividing method which will be discussed later, a processor maintains a partition of the simulated space, together with the boids that are within the partition. At

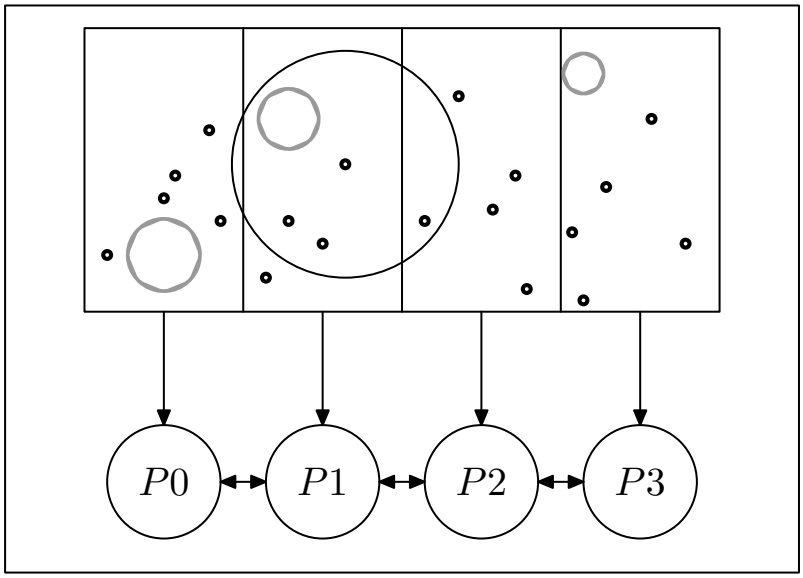


Figure 3.7: Near-neighbor Communication

the beginning of the simulation, all-to-all communication is used to get the global environment information.

Besides the two additional forces based on static environment information, the three local forces from the boids are all related with the dynamic flock. It is efficient to use the near-neighbor communication to get the useful information from the logically neighboring processors. The effect of this mechanism depends on the size of "vision". If the vision exceeds the space managed by the logically neighboring processors, the information on other processors may also be used. In this thesis, we assume that the range of the vision of a boid is small as compared with the size of a partition. Thus, we adopt the near-neighbor communication mechanism for information collection to enhance the simulation performance.

3.3.2 Partition

In a parallel simulation, partitioning mechanism is another critical factor to the system performance. Barnes and Huts proposed to divide the space rather than distribute particles to processors. We use column wise block-striped decomposition to partition the simulated space, then distribute the neighboring spaces to logically neighboring

processors. After that, each processor will maintain one partition, together with the corresponding boids in that partition.

There are two factors that need to be considered. First, the distribution of boids among the processors should be considered while partitioning the space. Second, whether the space partitioning is static or dynamic should also be considered.

- **Even Distribution**: To achieve better performance, each processor in a PC-cluster should take charge of approximately equal computational load so that each processor will use roughly the same time to finish the force calculation in each frame. It means each processor simulates a fixed subset of the group containing the same number of birds. Thus, the synchronization overhead is minimized.

  The simulation environment contains the flock and obstacles. It is hard to make them both evenly distributed among the processors. For example, if the space is divided to make obstacles distributed evenly among the processors, extremely uneven distribution of boids among the processors may occur.

  For a static environment, all-to-all communication will be executed once to form the static global view of environment in each processor. The computational load of detecting the neighboring obstacles and interacting with them primarily depends on the number of the boids in the processor. Therefore, the even distribution of boids is more important to the performance of the whole simulation system.

- **Dynamic Load Balancing**: Suppose that the environment is static. In the sequential model, the calculation of the three local forces from the boids will be more time-consuming with the increase of the group size. If each processor manages a fixed partition of the flock, it can maintain the even distribution of the boids. However, it requires all-to-all communication to collect the necessary

information in each frame, since one moving boid may have the neighboring boids in any processor (see Figure 3.8).

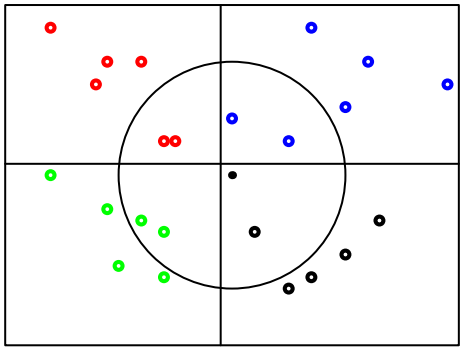


Figure 3.8: Neighboring boids in all processors for fixed partition of boids

Based on the space-partitioned method, the simulation only uses near-neighbor communication to collect the necessary information about boids. However, load imbalance may occur frequently due to the migration of the boids among the processors. Some moving boids may leave the space managed by the processor that they currently reside in, and get into another space managed by another processor.

Due to the movement of the boids, the initial even distribution among the processors may be broken as the simulation progresses. To maintain even distribution, dynamic partitioning mechanism is used for load balancing. When load imbalance among the processors exceeds a pre-specified threshold, the load balancing algorithm needs to re-partition the simulated space and re-distribute the successive spaces among the processors. In Figure 3.9, the dashed lines are borders of the former partition, and the real lines are border of the latter partition after load balancing. After the load balancing, each processor will manage a new space, together with new boids and new border information.

3.3.3 Parallel Flocking Algorithm

We develop the parallel flocking algorithms using MPI [55] to investigate the performance of our proposed mechanisms.

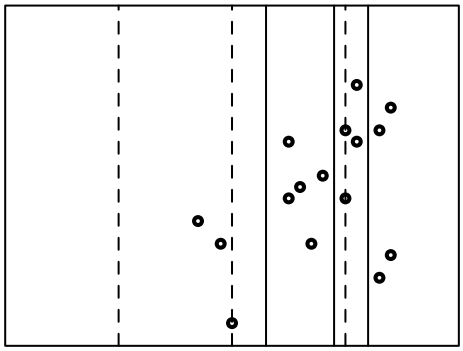


Figure 3.9: Dynamic partition of the simulated space

At first, we initialize the boids, obstacles and processor settings, and designate the leader. Then we partition the simulated space to make the boids be evenly distributed to the processors, and get the boundary information. After one all-to-all communication, each processor receives the information about all obstacles, and keeps a snapshot about the environment.

Figure 3.10 shows the skeleton of our parallel simulation algorithm.

```
01. initialize the simulated environment;
02. partition and distribute the space;
03. collect information about obstacles using all-to-all communication;
04. collect information about bois using near-neighbor communication;
05.   for(each boid)
06.       apply the goal attraction;
07.       compute the forces from boids;
08.       compute the forces from obstacles;
09.   endfor
10. update the position of boids;
11. update the distribution among processors;
12.   if(imbalance exceeds the threshold)
13.       invoke the load balancing method;
14.   endif
15. goto step 4;
```

Figure 3.10: Parallel simulation algorithm

The algorithm uses near-neighbor communication to collect the information about neighboring boids, then uses the similar procedure from the sequential simulation

algorithm to calculate the new position of each boid. After updating the positions, it re-distributes the boids among the processors based on the current boundaries. After that, it evaluates the degree of imbalance. If the load imbalance among the processors exceeds the threshold, the load balancing mechanism is invoked. If the imbalance is still acceptable, it jumps to the beginning of the interaction, and produces the next frame. In this algorithm, we suppose the environment is static. After load balancing, the environmental information is still the same, so we just go to step 4 rather than step 3, because each processor has kept a copy of the whole environmental information after step 3.

The 2-dimensional simulated space is divided into P column wise strips. Each strip is mapped onto one processor. If the number of boids residing in one processor is more than the threshold, load balancing is invoked. It re-calculates and updates the boundary information, re-partitions and re-distributes the virtual space into the processors, and also re-distributes the boids among the processors based on the new boundaries. Because the calculation of boundary information is a time-consuming task, the load balancing should not be invoked frequently. We will evaluate the influence of load balancing under different conditions in the next section.

The major differences between our parallel algorithm and Lorek and White's algorithm [43] are shown in Table 3.1. In Lorek and White's implementation, each processor deals with a fixed number of birds to realize load balancing. However, with p processors, the all-to-all communication mechanism requires $P - 1$ steps to collect the data from other processors. In our work, each processor deals with one partition of the simulated environment, and the partitions are dynamic for load balancing.

3.4 Experimental Results

In this section, we evaluate the performance of our parallel simulation algorithm for different number of boids on different number of processors, and compare the performance of our algorithm with Lorek and White's implementation. We also analyze

Table 3.1: Comparison of the two parallel algorithms

main characteristics	algorithm	
	Lorek and White's	Ours
all-to-all comm.	√	–
near-neighbor comm.	–	√
keep even distribution	√	–
load balancing	–	√

the performance under different environment settings to investigate the factors which affect the system performance.

The experiments were done on a Linux PC-cluster. It has 27 processors connected with Myrinet and a total of 8.0GB memory. We use 16 processors of the cluster. The average speed of the processors is about 450MHz. MPI is used to realize the communication among the cluster processors.

We compute the time needed to produce 1000 frames in different configurations. The radius of the boid's vision is set to 20 cm. The maximum speed of each boid is set to 2 cm per frame.

The simulated environment is a 800cm×600cm space, with round obstacles of different sizes. We first investigate the influence of the number of obstacles on the performance of the algorithm. Two typical scenarios are used in the simulation. The simple scenario is composed of only 24 obstacles, and the complex scenario is composed of 240 obstacles. Figure 3.11 shows the simulation results of the simple and the complex scenario where the number of boids is a medium size of 128. We can see that the influence of the number of obstacles on the performance reduces greatly with the increase of the number of processors. Therefore, in the remaining experiments, we only use the simple scenario. Various configurations are used in the experiments: the number of processors (P) is set to 1, 2, 4, 8 and 16 respectively, the number of boids (N) is set to 32, 64, 128, 256 and 512 respectively.

Figure 3.12 shows that the execution time reduces significantly when more processors are used, especially when the flock is composed of a large number of boids.

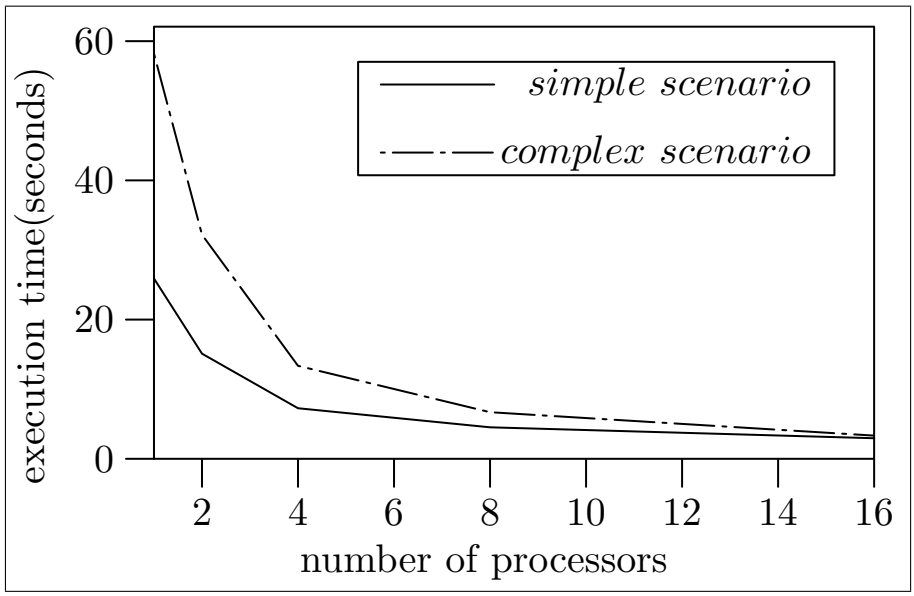


Figure 3.11: Performance of varying environment

For example, when the group contains 500 boids, the execution time reduces most greatly with the increase of processors. When the number of boids is small, the value of parallel simulation becomes small as well. Under this situation, the communication overhead and some other overhead for distribution among processors becomes eminent. So only when the number of boids is large, the parallel simulation using more processors shows its advantage. In this figure, the configuration with $P = 1$ means that one processor simulates the whole flock, so it is equivalent to a sequential algorithm. Compared with the sequential model, obviously we obtain a better performance through parallel simulation.

It is noticed that the performance of the flocking simulation is not good when the number of boids is small. The reason is that the communication overhead becomes a significant part of the simulation when the computational load is light.

Figure 3.13 shows that our algorithm performs better when the number of processors (P) is bigger. In our algorithm, the overall computational load on each processor becomes small as P increases. Compared with Lorek and White's algorithm, the spatial partitioning mechanism in our algorithm helps to limit the number of neighboring

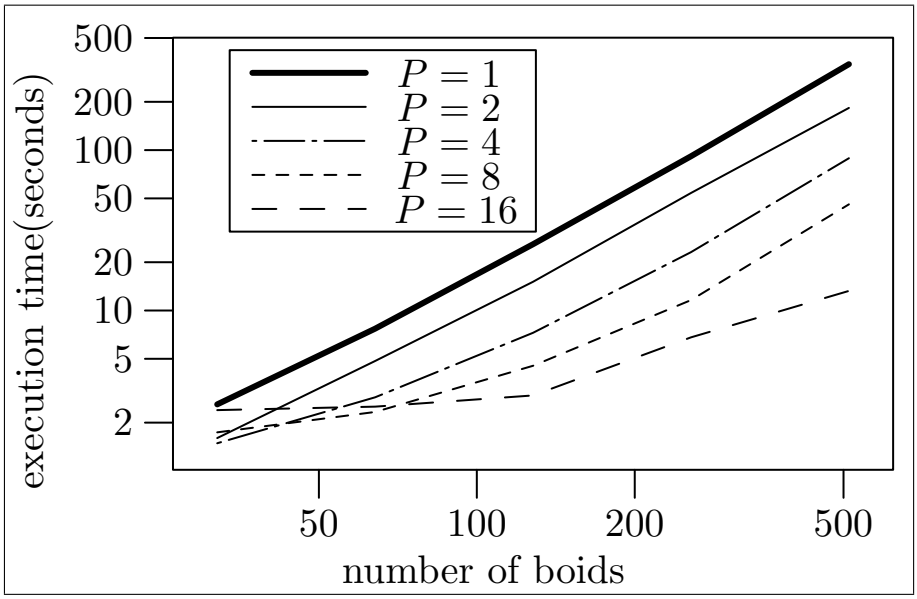


Figure 3.12: Performance of varying numbers of boids and various processor configurations

boids especially when P is larger than 4.

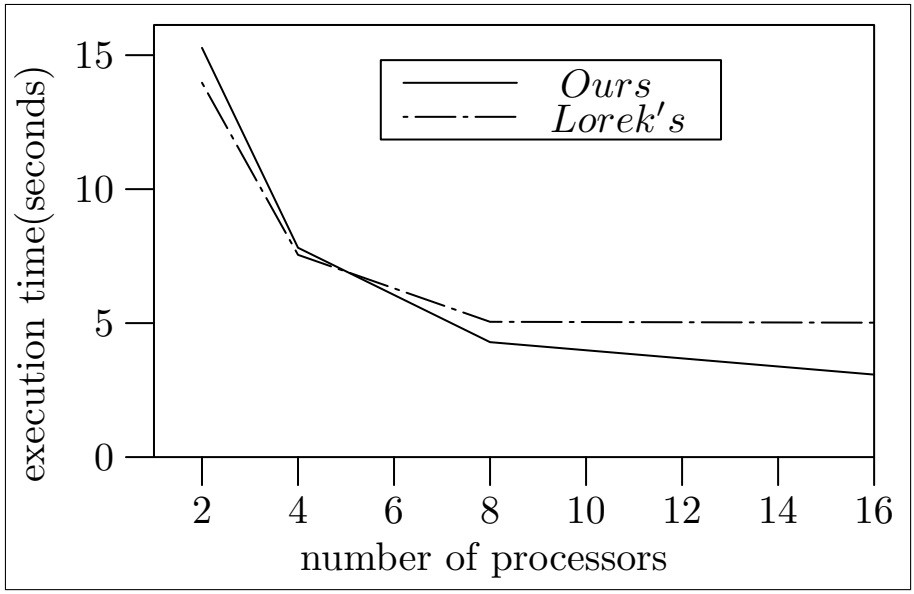


Figure 3.13: Comparison between Lorek and White's and our algorithm($N = 128$)

Figure 3.14 shows that load balancing is effective to enhance the performance. At the beginning of the simulation, we set the initial positions of boids to a small corner in the simulated environment, and set the goal at another opposite corner.

Because of the movement of boids, the method without load balancing will result in severe uneven distribution because one processor may afford the whole group while the others are left unused. The simple load balancing strategy we used performs well to reduce the load imbalance.

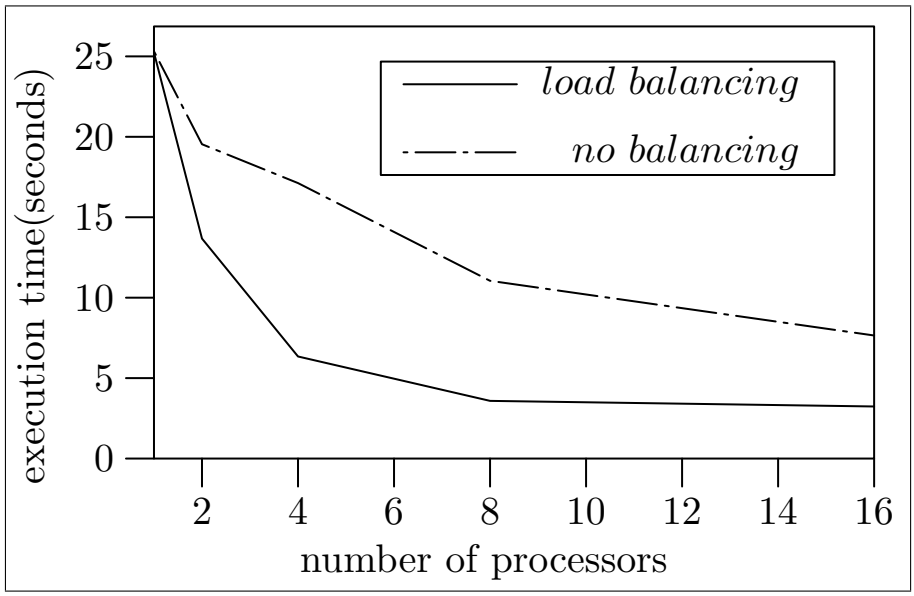


Figure 3.14: Performance of algorithms with and without load balancing

If load balancing is frequently invoked, it will lead to inefficiency. In this case, we will compute the percentage of boids in each processor. The threshold is defined as twice as the average percentage of boids by the number of processors. So we need to define a suitable threshold to keep a good balance. Table 3.2 shows the analysis of the algorithm with load balancing. The total cost of the simulation includes the main calculation cost, the communication cost and other overheads. Here the load-balancing cost is also incorporated into "other overheads".

From table 3.2, we can see that the communication time increases with the number of processors, and gradually becomes a significant part of the total time. With the increase of the number of processors, "other overheads" including the time for moving boids among processors, the time for collecting the boundary information and the time for load balancing become bigger. We can also see that the number of times

Table 3.2: Analysis of the algorithm with load balancing

Different parts	The number of processors (P)				
	1	2	4	8	16
Total time	25.24	14.34	7.75	4.29	3.36
Commun. time	0	0.05	0.28	0.54	0.90
Other overheads	0.09	0.17	0.53	0.93	1.25
Number of times load-balancing is invoked	0	13	19	12	19

that the load-balancing is invoked is not related to the number of processors. In fact, because the task to move the boids among processors needs to communicate with more processors in each frame when P is bigger, it becomes a more significant part of "other overheads" as P increases.

3.5 Summary

In this chapter, we proposed a parallel algorithm for the flocking behavior simulation. With our parallel algorithm, the simulated environment was divided into successive partitions, and one partition was allocated to one processor. In the simulation, each processor only needs to communicate with its logically neighboring processors for collecting necessary information. Load balancing was used to help distributing boids evenly among processors. We implemented the proposed mechanisms on a PC-cluster. Experimental results show that the proposed mechanisms result in better performances as compared with Lorek and White's method and the sequential implementation when the size of the group becomes larger.

The current implementation provides a good framework for future work. In our future work, we will investigate better load-balancing mechanisms and dynamic partitioning mechanisms. More sophisticated group behavior models will be investigated in parallel computing environments. Based on Reynolds' flocking model, the next chapter presents motion planning methods to realize more intelligent group behaviors in complex and dynamic environments.

Chapter 4

Motion Planning in Complex Environments

4.1 Overview

Motion planning can be defined as finding a path for an object from a start to a goal configuration without colliding with obstacles in the environment. Motion planning was traditionally studied in the area of robotics in order to plan the motion of robot arms and robot vehicles. In recent years, it has been investigated in many other fields such as computer games, virtual environments and computer-aided design. For example, in computer games and virtual environments, computer-controlled entities need to move around in natural ways. They must plan their routes amidst obstacles and other moving entities. Motion-planning techniques that originate from robotics have been adapted and applied in such environments. Due to the computational complexity of motion planning, most existing motion planning methods are impractical for complex environments since they are computationally infeasible except for some simple cases, for example, when a robot has very few degrees of freedom (DOF) or a group has a small number of members. To address motion planning problems for group behaviors in complex environments, randomized or probabilistic motion planning methods are investigated based on random heuristic searching method. These methods have gained significant attentions in recent years.

Random search approaches like the probabilistic roadmap method(PRM) planners introduced in [33] pre-compute a roadmap of valid paths reflecting the connectivity of the free-obstacle-configuration space, which allows to process multiple path queries as fast as possible. Some centralized techniques to implement PRM have been proposed for multiple entities. Because they treat the whole entities as a large system, the system has a large DOF when there are a large number of entities. A serious drawback of the random search approaches is the high variance in both computation time and quality of the solution. Few work based on PRM framework are focusing on dynamic environments. The adaptation of PRM planners to environments with both static and moving obstacles has been limited so far.

Challenging problems about motion planning in dynamic environments are discussed in Section 4.2. In Section 4.3, the basic PRM approach is analyzed, then a self-adaptive PRM is proposed to adapt to the dynamic environment. A hierarchical motion planning architecture is proposed in 4.4. Experimental results of the self-adaptive PRM and the hierarchical motion planning method are presented based on two different environments respectively in Section 4.5.

4.2 Problem Definition

In games and other virtual environments, computer-controlled entities need to move around in natural ways. They must plan their routes amidst obstacles and other moving entities. Motion-planning techniques that originate from robotics have been adapted and effectively applied in such environments.

In its simplest form, the motion planning problem requires a collision-free path to be computed for a moving body between start and goal positions. Motion planning was traditionally studied in the area of robotics in order to plan the motion of robot arms and robot vehicles. In recent years these techniques have been increasingly used in virtual environments and games. In such applications, computer-controlled entities move around and consequently their motion must be planned.

Motion planning for groups in complex environments offers a challenging problem setting because of the following aspects [51]:

- **Multiple entities**: multiple entities often move in the same environment, and must avoid each other and behave as a group
- **Complexity**: scenes are very complex, with up to a hundred or thousand obstacles
- **Dynamic**: scenes tend to be dynamic, i.e. obstacles can appear or disappear (for example when opening a door or when a fire starts)
- **Real time**: motions must be computed in real time for some special applications.

Many methods have been proposed to simulate the group behavior in a complex environment with some static obstacles. An important part of this kind of simulation is the obstacle avoidance method. It is used to prevent the moving objects from colliding with obstacles, allow the group to alter its course when they are trapped in local minima. However, there is little research about the obstacle avoidance method when the obstacles are not static. In [35], the randomized motion planners explore the *configuration*$\times$*time* space for obstacles moving along known trajectories. In [29], a roadmap planner is designed to operate in dynamically changing environments.

In this work, we investigate how to deal with the motion planning problem when the environment contains some dynamic entities, such as moving obstacles and other moving groups. We propose to use a self-adaptive roadmap to guide the group leader, and make other group members follow the leader using a distributed behavioral model [90]. We try to update the nodes included in a queried path to construct a partly correct PRM roadmap, because this roadmap is only used to guide the group to escape from the local minima in dynamic environments. A hierarchical motion

planning mechanism is also proposed to solve the bottleneck of reconnection process when some connections are blocked [91]. Experimental studies are also performed with two different dynamic environments to evaluate the performance of the proposed mechanisms.

4.3 Self-adaptive Roadmap based Motion Planning

In 1987, Reynolds established a distributed behavioral model of the flocking behavior [58]. His model described the behavior of the boids in a group using local rules for each individual boid. He also proposed a "steer-to-avoid" model to avoid obstacles. In this model, a boid actively looks ahead and finds out if it is on a collision course with the obstacles. If the collision is about to occur, the course is changed to make the boid clear of the obstacles [59]. However, there is a drawback of this approach. Without a long-term planning based on global information, the boids act based on local information can easily get stuck in cluttered environments when all the urges have reached a balance.

By combining the PRM approach with Reynolds' flocking mechanisms, we can use the roadmap created by the PRM to plan the global motion of the group leader based on its current location and the state of the environment. At the same time, at a lower level, the flocking behavioral model is used to make the boids move as a group and to avoid collisions with each other.

PRM-based motion planning is a two-phase process. A roadmap is constructed during the pre-processing phase; once the roadmap has been constructed, it can be used later to solve motion planning queries [48]. The roadmap construction proceeds in two stages: (1) random generation of collision-free nodes in free space (see Figure 4.1.a, Node Generation), and (2) connection of these nodes to form a graph, or roadmap, using some method to connect 'nearby' nodes (see Figure 4.1.b, Connection). Here we use the simplest metric of the distance between two nodes to identify

the 'nearby' nodes. Nodes and connections are added to the graph until the roadmap is dense enough. In the query phase, the start and goal configurations are connected to the graph, and a path connecting these connection nodes is obtained from the roadmap using some standard graph searching techniques (see Figure 4.1.c, Query). In this section, we show how the roadmap techniques can be used to achieve the homing behavior.

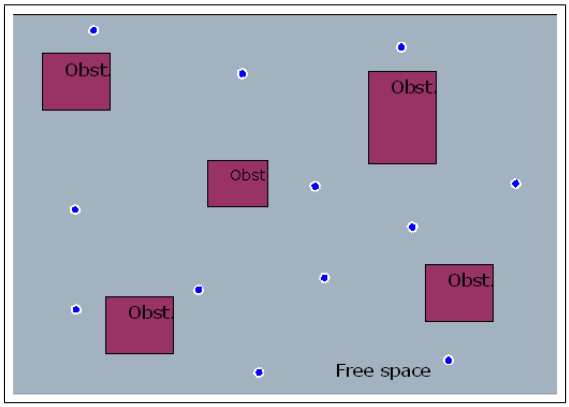
4.3.1 Homing Behavior Simulation Using the PRM Method

The PRM-based path planning method will build a roadmap automatically and find a collision-free path for the boids. The path can give each boid a global view to avoid getting trapped in local minima. The path is discretized to subgoals. As soon as the leader reaches a node on the planned path, it will select the next node on the planned path as the subgoal. By keeping track of the subgoals, the flock can reach the final goal.

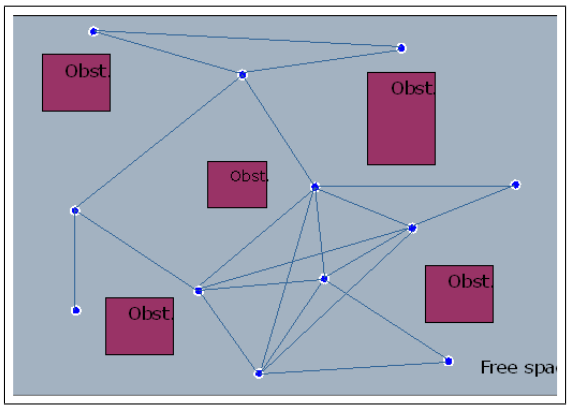
While homing, a designated leader knows the position of the goal. Meanwhile, the followers will pursue the leader. Fig. 4.2 shows the homing behavior algorithm for the leader of a group.

With other interacting forces from neighboring flock members and obstacles, steering toward the subgoal is given the lowest priority, so the group members still move together while moving toward the goal.

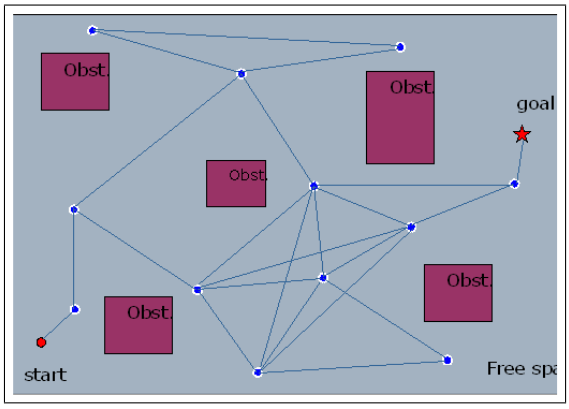
However, the algorithm is based on the planned path in the pre-processing stage. If the environment has some moving obstacles or some other moving groups, the pre-computed path may no longer connect the start and the goal correctly because the connection between two nodes may be broken. In addition, because the node connection task consumes about more than 90% of execution time of the roadmap construction process [2], it is inefficient to re-connect the nodes and search a new path for the dynamic environment. In the next section, we will propose a method to construct the self-adaptive roadmap to solve these problems.



4.1.a: PRM Roadmap - after Node Generation



4.1.b: PRM Roadmap - after Connection



4.1.c: PRM Roadmap - Query

Figure 4.1: A PRM roadmap created in a simulated environment

```
01. if(the goal is within the vision)
02.       stay near the goal;
03. else if(the current subgoal is in view range)
04.          set the next subgoal as the target;
05.       else
06.          steer toward the target;
07.       endif
08. endif
```

Figure 4.2: Homing behavior algorithm for the leader

4.3.2 Self-adaptive Roadmap

We propose a new method by changing the query phase of the PRM-based motion planning. Our new method does not need to give the group a globally correct map of the dynamic environment. We only use the pre-computed roadmap to obtain a path, then evaluate and correct the path during the simulation to help the group perform the homing behavior and escape from the local minima.

Firstly, we generate a roadmap only based on the static obstacles in the environment, without considering the presence of the moving obstacles. Then, during the query phase, a path is obtained for the group leader. In each simulation frame, this path is updated by checking if existing connections are broken by the moving obstacles. Broken connections are labelled as being blocked in the roadmap. If the path does not contain any blocked connection before reaching the goal, the roadmap remains unchanged. If the path contains blocked connections, the RRT (Rapidly-exploring Random Trees) planner [36] can be used to reconnect the nodes of those blocked connections. The goal-biased RRT is well-suited for this kind of short length connection, since the complexity of RRT depends on the length of the solution path. Finally, if the existing roadmap does not allow a path to be found or reconnection is failed, new nodes are inserted to reinforce the roadmap. The reinforced roadmap represents more detailed and correct topology of the changing environment.

The proposed self-adaptive roadmap is efficient because we separate the static obstacles from the dynamic obstacles, and generally the number of static obstacles is larger than the number of dynamic obstacles. The pre-computed roadmap based on static obstacles provides a basic structure of the global environment. After querying this basic roadmap, the obtained path becomes the essential part of the roadmap. We only need to correct the node connections of the obtained path if the moving obstacles break some connections of this essential part. In general, the obtained path contains a small amount of nodes of the whole roadmap, thus our method may greatly reduce the re-connection time.

To generate the homing behavior, besides the adaptive roadmap information, the group leader also needs some intelligence to adapt to the dynamic environment. After we get a path, the group leader will move toward the next subgoal and increase the weight of the current subgoal in each frame. The connected path may become invalid because of the dynamic feature of the environment. If the weight of the current subgoal exceeds the pre-defined threshold, it means that the leader is trapped in a local minimum. Once the local minima trap is detected, the leader will query the roadmap again using its current node as the start node. Other boids will follow the leader to achieve a realistic group behavior in dynamic environments. A well-generated roadmap gives the flock more chances to avoid trapped state.

Compared with the basic PRM-based method, our new method does not need to provide a completely correct roadmap for the flock. We only test and correct a small amount of nodes of the roadmap after the dynamic updates of the roadmap, which saves a lot of computational cost. Fig. 4.3 shows the algorithm of the self-adaptive roadmap method for the group leader. Since we suppose only the leader knows the position of the goal, the direct goal attraction only has impact on the leader. The leader is also guided by the subgoal from the obtained path nodes. If the weight of current subgoal reaches the threshold, it means the leader is trapped in a local

minimum and the connection between the current node and the current subgoal is invalid or unreachable. We label the connection as blocked in the roadmap and query the roadmap again to obtain a new path.

```
01. if(the boid is the leader of the flock)
02.     compute the force of goal attraction;
03.     compute the force of subgoal attraction;
04.     if(the goal is achieved or the weight of
          current subgoal reaches the threshold)
05.       set the current node as the start node;
06.       label the connection to the current subgoal as blocked;
07.       query the roadmap again to obtain a new path;
08.     else if(the current subgoal is achieved)
09.             set the current subgoal as the current node;
10.             set the next path node as the current subgoal;
11.           else
12.             increase the weight of the current subgoal;
13.           endif
14.     endif
15. else
16.     follow the leader;
17. endif
```

Figure 4.3: Self-adaptive roadmap method

Besides the advantages mentioned above, when there are more than one moving groups, the boids in each group can use this model to avoid the "dead lock" state. Using the basic PRM-based method, each group will get a static path after querying the roadmap. If the two pathes for two groups have some points of intersection and the next subgoal of each group is occupied by another group, they will enter a "dead lock" state. However, in this case, with our new method, the weight of current node will eventually exceed the pre-defined threshold after a while, the "dead lock" state will be detected and then be solved automatically by starting a new query.

4.4 Hierarchical Motion Planning

In the self-adaptive PRM, if the obtained path contains blocked connections, the RRT is used to reconnect the nodes of those blocked connections. If this reconnection process is failed, we still need to re-construct a reinforced roadmap. As mentioned before, it is very time-consuming to re-connect all the nodes in the environment. We consider a structured-environment as shown in Fig. 4.4, where the whole environment can be treated as a spatial organization of different spaces. Each space is connected to its neighboring spaces by connection nodes, or "doors". This kind of environment is considered because it is commonly used to model offices or warehouses in the MOUT (Military Operation on Urbanized Terrain) domain. If we connect all the connection nodes, we can identify the connectivity of the spaces at a gross level. To reduce the computational cost of re-connecting all the nodes in the changing environment, we propose a hierarchical motion planning mechanism.

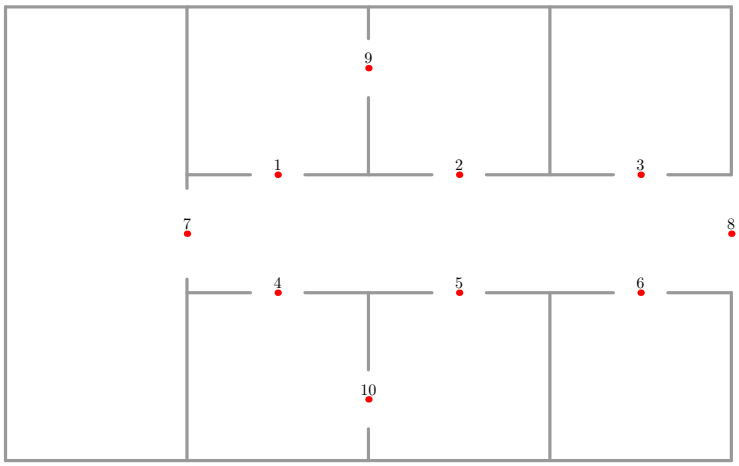


Figure 4.4: The outline of a structured environment

The hierarchical motion planning method contains two levels. The first level is the gross planning for the rough description from one space to another. It can identify which region is closed and unaccessible. At this level, the group leader can get a list of accessible spaces that it will pass through. The second level is the fine planning

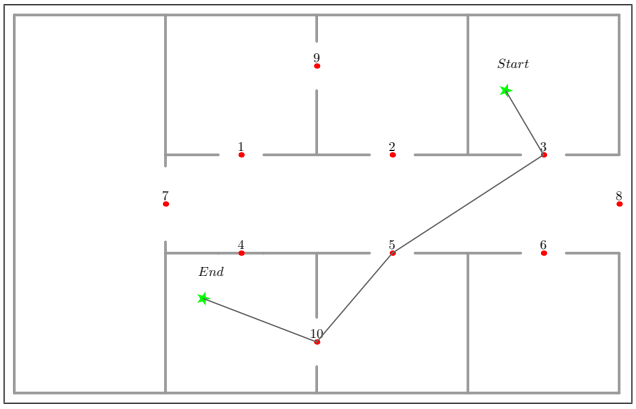
for each listed space. Once the group moves into a specific space, the fine planning mechanism generates a roadmap to help the group move in this space. The first level is dynamic because of the random opening or closing of the doors, and the second level is static. Thus, when the doors are randomly opened or closed, we only need to re-build the first-level roadmap.

This approach reduces the computational cost by dividing an environment to several areas. The complexity of the best known A* path-finding algorithm is O($NlogN$), where N denotes the number of nodes in the map. For some very large maps, its computational cost is a critical factor to the performance of simulation. Through the hierarchical motion planning, a rough path is computed first based on several connection nodes. Then the A* algorithm is applied only in a small specific area. It also increases the reality of the simulation and the flexibility to deploy more tactics on the group, since it is closer to human's path planning procedure.

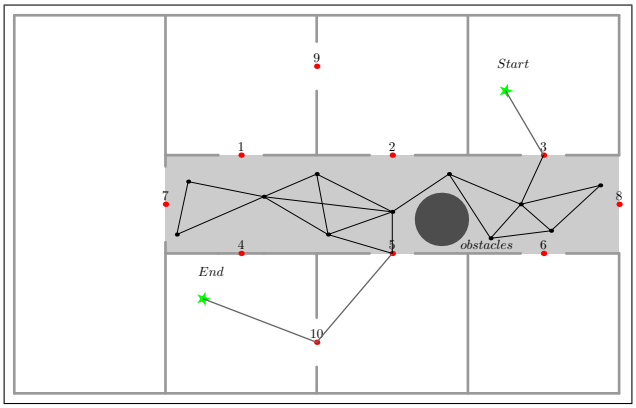
In fig. 4.5.a, we can see that a rough path connecting the start and the end has been found by the gross planning. It consists of door 3, 5, 10 consequently. If the group is now moving from door 3 to door 5, at the second level, the fine planning is only used in the shadowed space as shown in Fig. 4.5.b. If there are some other dynamic obstacles in this shadowed space, our self-adaptive PRM can also be used at the fine planning level as shown in Fig. 4.5.c.

4.5 Experimental Results

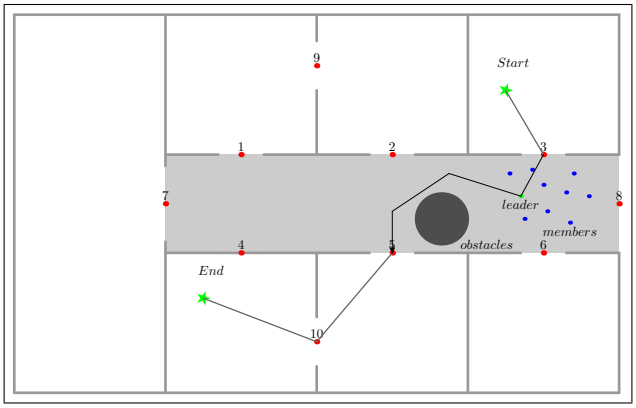
We have implemented two environments to verify the self-adaptive PRM and the hierarchical motion planning method. In the first environment, it is hard to use the hierarchical motion planning because we cannot identify the basic structure of the environment, and thus it is hard to know where to place the connection nodes. In the second environment, the hierarchical motion planning mechanism is applied to enhance the simulation performance.



4.5.a: Rough path connection from the start to the end



4.5.b: Roadmap produced by PRM in the shadowed space



4.5.c: Obtained path in the fine planning level

Figure 4.5: The hierarchical motion planning in the structured environment

4.5.1 A Cluttered Environment

We have developed a simulation environment to evaluate the performance of the self-adaptive algorithm. The flocking model is the same as the model proposed by Reynolds. For the homing behavior, the leader of the group is designated automatically at the beginning and will not change over time. Following the leader, other boids will tend to approach the position where the designated leader is currently located. The current simulation testbed was developed with C and OpenGL. The terrain size is 600 by 400 units and it is filled with 100 obstacles. The radius of the boid's vision is set to 20 units. The maximum speed of each boid is set to 2 units per frame. A group (white group) of 50 boids without motion planning mechanism are implemented to pursue a predefined goal. Another group (red group) of 50 boids will use the PRM to plan the motion in a static scenario, and use the self-adaptive roadmap in a dynamic scenario. At the query phase, A* path finding method [63] is used to get a connected path. We will compare the motion of the two groups and the two PRM-based motion planning mechanisms in static and dynamic scenarios respectively.

All tests were run on a NEC, Pentium 4 2.67 GHz machine with 512 MB RAM running a Gigabyte MAYA RADEON 9000 PRO Graphics card with 64 MB of memory.

In the static scenario, to compare the performance of the two different groups, we reset the goal's position whenever the group leader reaches the goal. The roadmap is built using 300 roadmap nodes and we attempt to connect each node to its 6 nearest neighbors. Fig. 4.6 shows the simulation results of this scenario. We can see that the red group (with motion planning) is guided by a light blue curved line produced by the basic PRM approach, while the white group (without motion planning) has been stuck and cannot escape in a long period, which shows that the obtained path provides a global guide without collision with obstacles for the red group. In all our tests in this static scenario, the group with motion planning performed the homing

behavior well, while the group without motion planning always got stuck at some time.

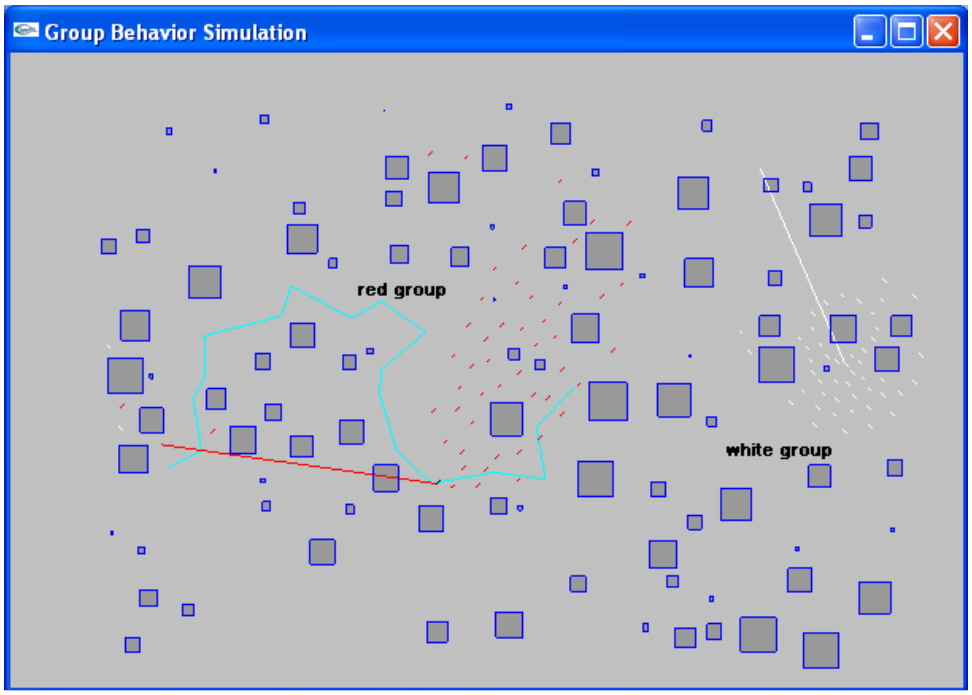


Figure 4.6: Screen shot of the group behavior simulation in a static environment

In the dynamic scenario, we set 20 of the 100 obstacles to move back and forth in the environment. Simulation results show that the self-adaptive PRM approach can be used to guide the group in a dynamic environment properly. The group using the self-adaptive PRM approach can move smoothly and avoid being stuck in local minima, while the group using the basic PRM approach may get stuck if some moving obstacles happen to break the connected path. To further compare the performances of the two approaches, we update the validity of all the nodes and connections of the basic PRM roadmap to solve the blocked connection. Comparative performance results are shown in Table 4.1. The results are based on the production of 500 frames. The results show that our self-adaptive PRM approach is useful and more efficient for group movement simulation in dynamic environments.

As mentioned before, the pre-computed roadmap based on static obstacles provides a basic structure of the global environment. Through the experiments, we

Table 4.1: Performances of the self-adaptive and the basic PRM roadmap method

main characteristics	PRM roadmap method	
	Self-adaptive	Basic
total time(s)	15.6	28.1
no. of reconnections	12	10
reconnection time(s)	5.7	17.2

observed that when more than half of the obstacles are dynamic in the simulated environment, the new method may not perform well. The reason is that if the whole environment changes dramatically, the global information of the environment becomes only temporarily valid and provides little global guidance information for the group leader.

4.5.2 An Open Office Environment

We also implemented a simulation in an open office environment created by the MOVIE (Motion Planning in Virtual Environments) Models [49] to evaluate the hierarchical motion planning method. Figure 4.7 is a screen shot of the environment. Callisto [30] is used in this experiment as a library to do visualization. In this environment, each region is used to simulate an office room. The offices are connected by doors which can be open or closed. This dynamic environment may change greatly because the connectivity of regions can be suddenly broken.

As mentioned before, if the whole environment changes dramatically, the precomputed roadmap becomes only temporarily valid and provides little global guidance information for the group. To compare the performances with and without using the proposed hierarchical motion planning mechanism, we first implement the PRM without using hierarchical motion planning in this environment. The doors are randomly opened or closed every 100 frames, and we try to update the whole roadmap when the status of doors changes. Figure 4.8 shows the roadmap produced by PRM in this environment.

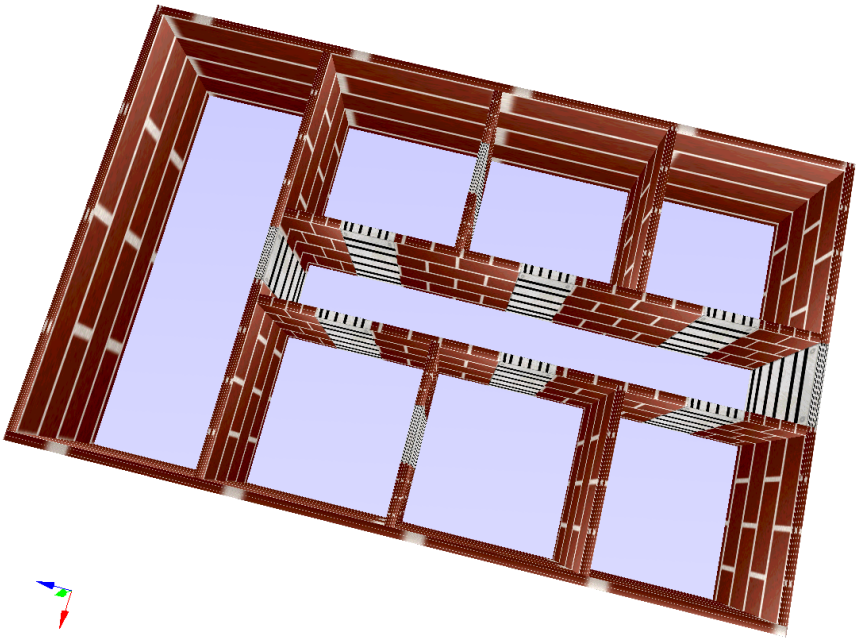


Figure 4.7: A screen shot of the open office environment

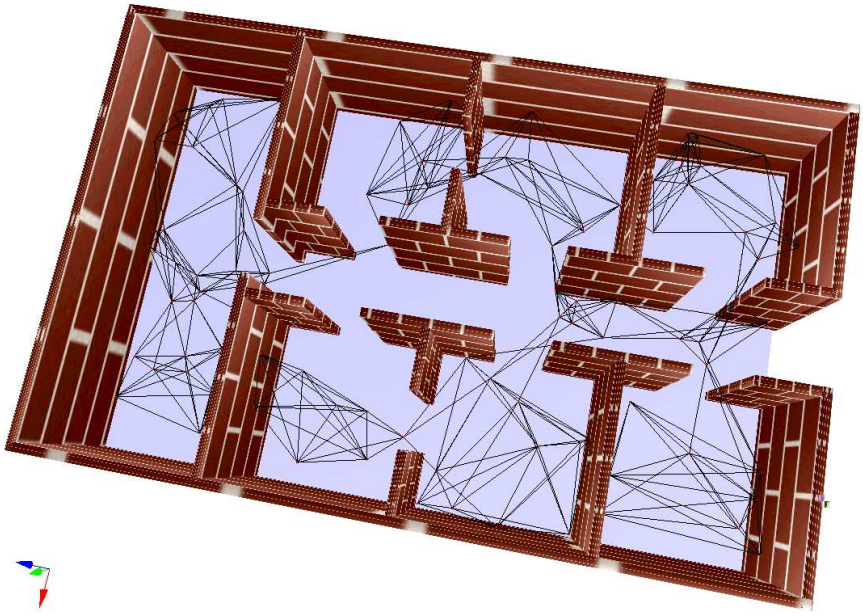


Figure 4.8: The roadmap produced by PRM in the open office environment

By querying this roadmap, the group leader can get a path connecting the start and goal positions. Figure 4.9 shows the simulation result of a group moving in the open office environment. We can see that the group can move intelligently from one office to another guided by this obtained path. However, when doors are randomly closed, the obtained path may no longer connect the start and goal positions correctly. Therefore, we need to re-build the roadmap and query a new path if some doors are closed.

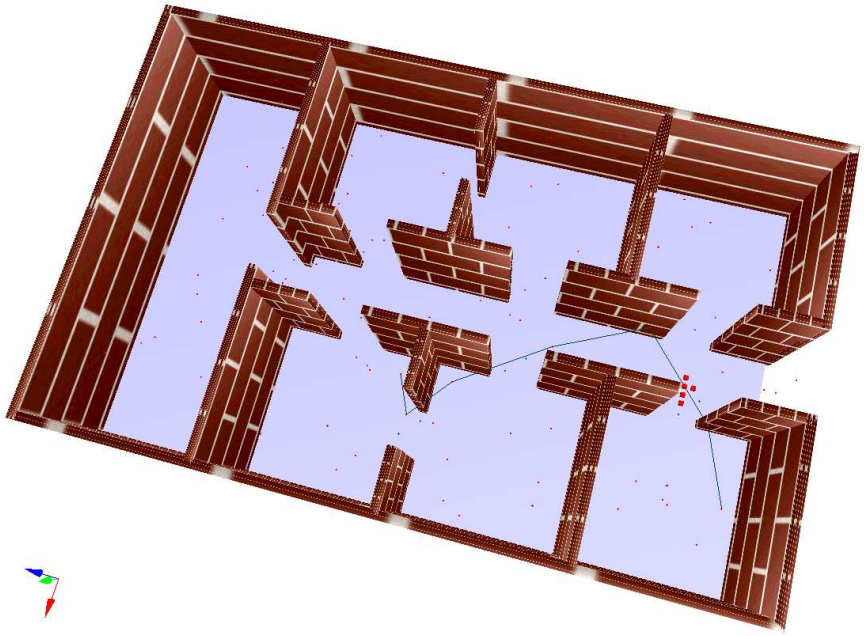


Figure 4.9: The simulation result of a group moving in the open office environment

In Figure 4.10 we can see that five doors are closed. A new roadmap is generated which represents the connectivity of the offices. The group leader need to query this new roadmap to get a new path. During the simulation we found that this process is very slow because the node reconnection task is very time-consuming. Therefore, we use the hierarchical motion planning mechanism to reduce the re-construction time.

With the hierarchical motion planning mechanism, when the doors are randomly closed, we only need to re-build the first-level roadmap. In the first level, besides nodes

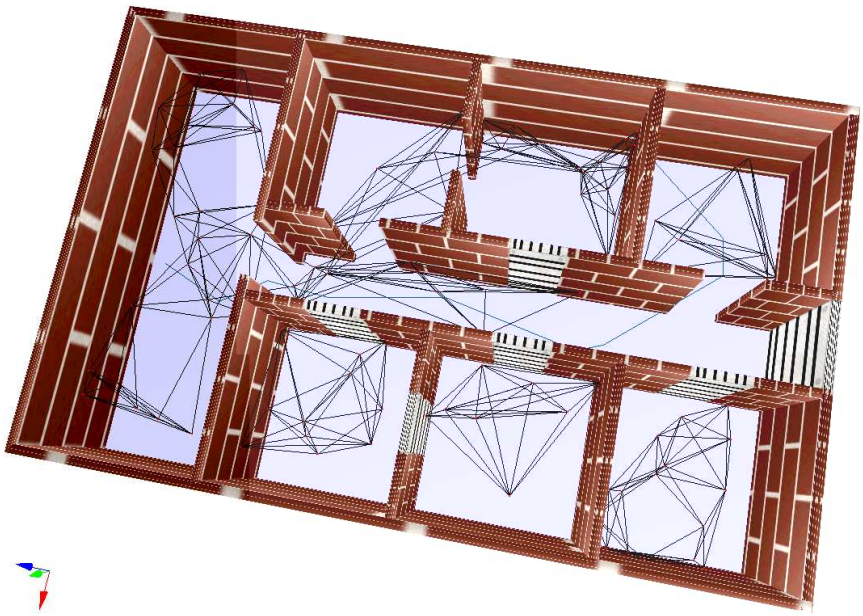
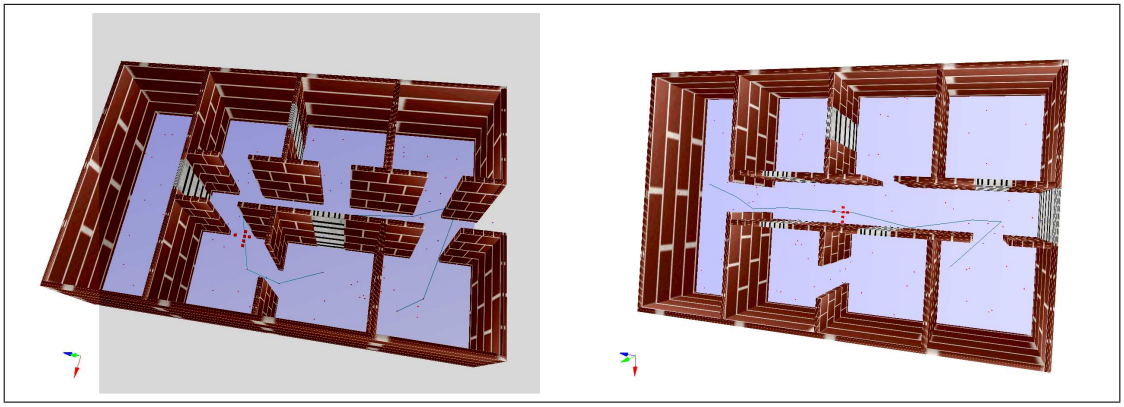


Figure 4.10: The roadmap is re-built when doors are randomly closed

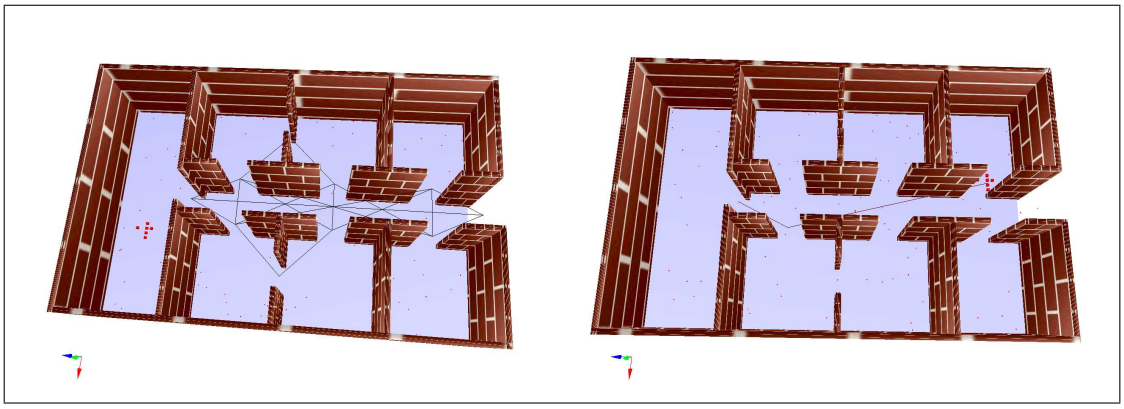
in the middle of doors, we also sample a node in the center of the environment to connect the nodes. Then we connect these nodes to generate the first-level roadmap. If one door is closed, delete the corresponding node and related connections. This task is very fast because it only depends on the number of connection nodes which is much smaller than the total number of nodes as in the previous case.

Figure 4.11.a depicts the results of the re-building and re-querying process because of the random closing of the doors. Without the hierarchical motion planning, these processes involve much more nodes compared with the case of using the hierarchical motion planning mechanism. An example of the first-level PRM is shown in Figure 4.11.b. We can see that the group leader is tasked to reach the goal guided by the rough path obtained from the first-level roadmap.

Comparative performance results are shown in Table 4.2. The results are based on the production of 500 frames. We can see that the hierarchical method greatly enhances the performance of the whole simulation. The average reconnection time



4.11.a: Without hierarchical motion planning: re-build and re-query the whole roadmap



4.11.b: With hierarchical motion planning: the first-level roadmap(left) and the obtained path(right)

Figure 4.11: The simulation results of the hierarchical motion planning

without hierarchical motion planning is about 2 seconds, which produces an obvious pause period during the simulation.

Table 4.2: Performances with and without using hierarchical motion planning

main characteristics	hierarchical motion planning	
	with	without
connection nodes	11	100
no. of reconnections	5	5
reconnection and re-query time(s)	0.8	9.6

The experimental results have demonstrated the effectiveness of the hierarchical motion planning mechanism in dealing with group movement simulation in dynamic environments.

4.6 Summary

Motion planning mechanisms are critical to create intelligent group behaviors in complex environments. How to plan the motion for a group of agents is challenging when the environment is complex or even dynamic. This chapter proposed the challenges of motion planning for group behaviors in complex environments. First, the PRM approach was implemented in this chapter. Then, to adapt to the changing environment, a new variant PRM was proposed to construct a roadmap which is partially correct, but was updated quickly for describing the dynamic feature of the environment. To solve the bottleneck of the re-connection process, a hierarchical motion planning architecture was also proposed to reduce the computational cost.

In this chapter, we also showed the simulation work of motion planning algorithms for group behaviors. The basic PRM was used to guide the group with a long-term consideration. Then, the new self-adaptive roadmap was implemented in a dynamic environment. The new method performed well if the environmental change was not too fast. To make performance comparison, another group without any motion planning was also introduced in the simulation. Conclusion can be reached

from the experimental results that motion planning mechanism is important for a group to search the target intelligently and the improved self-adaptive roadmap is a useful extension to help groups in dynamic environments.

Then, we proposed a structured open office environment in which the connectivity of the offices can be broken by the randomly closed doors. In this environment, we used the hierarchical motion planning method to enhance the performance of the simulation process. From the experimental results we conclude that the hierarchical method can reduce the computational cost when the environment changes and solve the bottleneck of the re-connection process. Without the pause period caused by re-connection of all nodes, the hierarchial motion planning mechanism can also make the simulation of the group movement smooth.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Group behaviors in nature have inspired researchers in computer animation, computer games, robotics and virtual environment, etc. A survey has been conducted to identify the problems encountered and potential research issues in theses fields. Relevant techniques have been studied in Chapter 2. Although many methods have been proposed to produce different kinds of group behaviors, these conventional techniques are not desirable when addressing some new challenges, e.g., large groups in grand scenes and intelligent agents in the group. This project works on developing and implementing new techniques for producing larger groups quickly and more intelligent group behaviors in dynamic environments.

Having studied the most popular distributed behavioral model proposed by Reynolds, we focus on how to improve the performance of Reynolds' flocking model in two directions: large groups and intelligent groups.

Having analyzed the computational complexity of the sequential algorithm for simulating large groups, a parallel simulation algorithm of the flocking model with new partition and communication mechanisms has been proposed and systematically described. The skeleton of our parallel algorithm has been presented and explained. With emphasis on reducing the communication overhead, the novel communication and partition mechanisms have been proposed to improve the performance of the

parallel simulation. Based on this parallel algorithm, a flocking simulation has been implemented and tested in a parallel computing environment. To test the performance of our algorithm, it has been compared with another parallel algorithm proposed by Lorek and White.

Then, we have investigated the motion planning problems for producing more intelligent group behaviors in complex environments. Having analyzed challenging problems about motion planning in complex environments, the PRM approach has been presented and implemented to help the group leader plan its motion. To adapt to the dynamic environment with moving obstacles, the self-adaptive PRM has been proposed to construct a partially correct but fast updating roadmap. Animation sequences have been generated to verify the performance in a cluttered environment and desirable results have been achieved. In order to prove the generic nature of the group in handling different dynamic features, a new simulation with the hierarchical motion planning method has also been implemented in an open office environment.

5.2 Contributions

New techniques have been successfully developed for simulating larger and more intelligent group behaviors in dynamic environments. The contributions of the project are summarized as follows:

- A new parallel simulation algorithm of the flocking model is developed to generate large group behaviors more quickly in a parallel computing environment. Simulation results show that, compared to Lorek and White's parallel method, our method with new proposed communication and partition mechanisms results in better performances, and compared to the sequential implementation, our method performs better when the size of the group is large.

- Based on the space-partitioned method, the simulation only uses near-neighbor communication to collect the necessary information. It reduces the communication overhead among processors largely.
- Load balancing mechanism is also considered to reduce the uneven distribution when partitioning the group members. The simple load balancing strategy we used performs well to reduce the load imbalance occurring in the simulation.

- A Self-adaptive PRM is proposed to construct a roadmap which is partially correct, but is updated quickly for describing the dynamic feature of the environment. Simulations have been done to evaluate the performance of this method and to compare with another group without any motion planning strategy. We can conclude from the simulation results that motion planning is important for a group to behave intelligently and our improved roadmap is helpful motion-planning in dynamic environments.

- A hierarchical motion planning method is proposed to enhance the performance of the simulation process in a structured space. It breaks the bottleneck of the re-connection process, and makes the movement of the group more smooth.

5.3 Future Work

Our new techniques address some new challenges in group behavior simulations. However, there are still more aspects that we would like to explore further in the future; particularly in the areas listed below:

- efficient motion planning in high dimensional spaces,
- characterization and communication of the global information about the dynamic environment, and
- intelligent group behavior with tactical cooperation among group members.

5.3.1 Motion Planning in High Dimensional Spaces

Motion planning in high dimensional spaces is a time-consuming and challenging task. Both the basic framework and the procedural-based scheme described in Chapter 3 and 4 respectively are only applicable for movement in 2-D configuration spaces.

Most motion planning methods are not practical since they are computationally infeasible because computation complexity for high dimensional configuration spaces would grow exponentially. For this reason, randomized or probabilistic roadmap methods (PRMs) have gained much attention for motion planning problems involving high dimensional spaces [33, 1]. PRM methods avoid computing an explicit representation of the configuration space. So they are of great use for applications with many degrees of freedom.

"Nevertheless, there exist many important applications which have not been solved by current methods, or cannot be solved within acceptable time constraints" [2]. In [2], Amato and Dale report on their experience parallelizing PRM motion planning methods. PRM methods are of particular interest to them because they are particularly amenable to parallel implementation. Because they are randomized, no node's generation depends on any other. That is, the roadmap nodes can be produced independently on all processors in parallel. This minimizes the costly overhead of communication among processors.

They show that significant, scalable speedups can be obtained with relatively little developer effort. In particular, they outline general techniques for parallelizing types of computations commonly performed in motion planning algorithms, and identify potential difficulties that might be faced in other efforts to parallelize sequential motion planning methods.

In the future work, we intend to implement some more sophisticated algorithms for generating better quality roadmap nodes in the generation phase and more sophisticated algorithms for searching paths with the local planner in the connection

phase. Then, we will try to combine the previous parallel simulation algorithm with the parallel PRM motion planning mechanism in a single system. We expect to see more benefit from the parallelism of these PRM motion planning techniques.

5.3.2 The Global Information about the Dynamic Environment

For PRM-based techniques, a valid representation of environment is the basis for planning queries. If the objects in the environment are dynamic, the constructed roadmap becomes invalid and we need generate it again.

Dynamic changes in the environment are very common in many path planning applications. A simple solution to motion planning in dynamic environments has been discussed in chapter 4. However, it is only useful for a dynamic environment with some little and simple changes. It only considers the validation and connectivity of roadmap nodes. It is not a robust model for spatial reasoning.

In the game community, "the terrain reasoning can significantly off-load the in-game AI" [69]. The game terrain is also not completely static. Dynamic environment contains movable objects such as doors, bridges, vehicles, elevators, destructible walls, etc. They will change the path trafficability and the lines-of-sight.

Dynamic environment in games does not necessarily invalidate pre-processed terrain information. Whether a distant door is open or closed does not really affect the decision of a squad. However, a nearby door being closed typically does affect the decision.

The addition of dynamic features to a game terrain presents many challenges. Terrain reasoning in a dynamic environment becomes significantly more difficult and expensive. Characterization and communication of the global information about the dynamic environment must be considered for the longer-term actions of a group.

In the future work, we want to build a central environment manager that will maintain all dynamic environment information. Qualitative spatial representations [16]

work will be investigated to deal with more efficient motion planning. The central manager can slowly update the dynamic information, the agents can also try to patch the terrain description. For example, the changes can be applied on the fly when a door is closed or a bridge is destroyed.

5.3.3 Intelligent Group Behavior with Tactical Cooperation

Building bots that can plan and reason under different situations is important for producing intelligent group behavior with tactical cooperation among group members. There exist many sophisticated reasoning models and planning algorithms today. A prominent one is Soar [67] developed in University of Michigan. Soar Quakebot [37] is the bot created by use of Soar architecture for the Quake game. The bot produced goal-directed behaviors through planning. In addition, based on the planning algorithm the bot can anticipate and predict enemy's behaviors by use of its own internal representation of the enemy's state. Besides, Soar has also been used successfully in simulating computer-generated forces.

Based on the Soar, behaviors generated can be highly goal-directed so as to give audience an illusion of human level intelligence. Especially in task oriented war game simulations, planning and reasoning are essential to produce realistic behaviors.

In the MOUT (Military Operation on Urbanized Terrain) domain, the missions consist mostly of defending specific rooms, with some individual missions. The agent's knowledge is independent of the specific mission and scenario. The same set of rules is used for all scenarios and all agents. For example, a single "mission specification" production rule allows the user to indicate teammates, fire teams, and commanders, the type of mission(defensive/offensive), each MOUTbot's role in the mission, initial areas to defend, places to which to retreat, and the type and direction of expected threats. Moreover, the leader can issue a limited set of commands to change the missions of others [88]. They are the basic requirements for tactical cooperation in the group.

Unreal Tournament is an off-the-shelf extendable 3D game engine that supports networked play. In the future work, we want to build bots based on Unreal engine that can autonomously achieve certain tasks or long-term goals in cooperation, as well as make quick response to urgent situations at the same time.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based PRM for 3D workspaces. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[2] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proceedings of IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 688–694, 1999.

[3] S. Amkraut, M. Girard, and G. Karl. Motion studies for a work in progress entitled eurythmy. In *SIGGRAPH Video Review*, volume 21, 1985. produced at the Computer Graphics Research Group, Ohio State University. Columbus, Ohio.

[4] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *IEEE International Conference on Robotics and Automation (ICRA 00)*, volume 1, 2000.

[5] J. Barnes and P. Hut. A hierarchical o(nlogn) force calculation algorithm. *Nature*, 324:446–449, 1986.

[6] J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. *Intl. Journal of Robots Research*, 10:628–649, 1991.

[7] O. B. Bayazit, J. M. Lien, and N. M. Amato. Better group behaviors in complex environments using global roadmaps. In *Proceedings of the Eighth International Conference on Artificial life*, pages 362–370. MIT Press, 2003.

[8] P. Becheiraz and D. Thalmann. A behavioral animation system for autonomous actors personified by emotions. In *Proceedings of the 1st Workshop on Embodied Conversational Characters*, pages 57–65, 1998.

[9] G. Blelloch and G. Narlikar. A practical comparison of n-body algorithms. In *Dimacs Implementation Challenge Workshop*, volume 30, pages 81–96, 1994.

[10] E. Bouvier, E. Cohen, and L. Najman. From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation. *Journal of Electronic Imaging*, 6(1):94–107, January 1997.

[11] D. C. Brogan and J. K. Hodgins. Group behaviors for systems with significant dynamics. *Autonomous Robots*, 4:137–153, 1997.

[12] D. C. Brogan, R. A. Metoyer, and J. K. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, 15(5), 1998.

[13] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–23, March 1997.

[14] A. Champandred. *AI Game Development*. New Riders, 2003.

[15] C. Delnay. Verchalf-life AI, schedules and task, August 2004. `http://collective.valve-erc.com/index.php?doc=1018030771-90025600` Last Visited: Monday, June 6, 2005.

[16] K. D. Forbus, J. V. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *IEEE Intelligent Systems*, 17(4):25–30, 2002.

[17] K. Fujimura. *Motion planning in dynamic environments*. New York: Springer-Verlag, 1991.

[18] J. Funge, X. Y. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *ACM Computer Graphics Proceedings*, pages 29–38, Los Angeles, CA, August 1999.

[19] Epic Games. Unreal tournament homepage, 2004. `http://www.unrealtournament.com/` Last Visited: Monday, June 6, 2005.

[20] E. Gelenbe, K. Hussain, and V. Kaptan. Simulating the navigation and control of autonomous agents. In *Proceedings of the 7th International Conference on Information Fusion*, pages 183–189, 2004.

[21] V. Gervasi and G. Prencipe. Flocking by a set of autonomous mobile robots. Technical Report TR-01-24, Department of Information, University of Di Pisa, Italy, 2001.

[22] D. Goleman. *Emotional Intelligence*. Bloombsbury, 1996.

[23] J. Greenberg and R. Baron. *Behavior in Organizations: Understanding and Managing the Human Side of Work*. Prentice-Hall, London, 6 edition, 1995.

[24] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[25] M. K. Habib, H. Asama, I. Endo, A. Matsumoto, and Y. Ishida. Simulation environment for an autonomous decentralized multi-agent robotic system. In *Proc. 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1550–1557, Raleigh, NC, July 1992.

[26] A. T. Hayes and P. Dormiani-Tabatabaei. Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *Proc. of the 2002 IEEE Int. Conf. on Robotics and Automation*, pages 3900–3905, Washington DC, USA, May 2002. IEEE Press.

[27] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[28] C. E. Izard. Four systems for emotion activation: Cognitive and noncognitive processes. *Psychological Review*, 100(1):68–90, 1993.

[29] L. Jaillet and T. Simeon. A PRM-based motion planner for dynamically changing environments. Technical report, MOVIE, 2004. `http://www.give.nl/movie/publications/laas-kineo/iros04-dynamic-planner.pdf` Last Visited: Monday, June 6, 2005.

[30] A. Kamphuis. Callisto, 2005. `http://www.cs.uu.nl/people/dennis/callisto/callisto.html` Last Visited: Monday, June 6, 2005.

[31] A. Kamphuis and M. H. Overmars. Motion planning for coherent groups of entities. In *IEEE Int. Conf. on Robotics and Automation*, San Diego, CA, 2004.

[32] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *International Conference on Robotics and Automation (ICRA)*, pages 2138–2139, San Diego, CA, 1994.

[33] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12:566–580, 1996.

[34] K. Kelly. *Out of Control: The New Biology of Machines, Social Systems and the Economic World*, chapter Hive Mind.

[35] R. Kindel, D. Hsu, J. C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *International Conference on Robotics and Automation*, pages 537–543, 2000.

[36] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[37] J. E. Laird. An exploration into computer games and computer generated forces. In *The Eighth Conference on Computer Generated Forces and Behavior Representation*, Orlando, FL, May 2000.

[38] J. E. Laird. Building intelligent synthetic characters for computer games. `http://www.movesinstitute.org/Events/Laird.MOVESTalk.ppt` Last Visited: Monday, June 6, 2005, March 2001.

[39] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[40] S. Leroy, J. P. Laumond, and T. Simeon. Multiple path coordination for mobile robots: A geometric algorithm. In *International Joint Conference on Artificial Intelligence*, pages 1118–1123, 1999.

[41] T. Li and H. Chou. Motion planning for a crowd of robots. In *International Conference on Robotics and Automation (ICRA)*, pages 4215–4221, San Diego, CA, 2003. IEEE Press.

[42] P. Liu and S. N. Bhatt. Experiences with parallel n-body simulation. In *6th Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 122–131, 1994.

[43] H. Lorek and M. White. Parallel bird flocking simulation. In *Parallel Processing for Graphics and Scientific Visualization*.

[44] Massive. Massive software homepage, 2004. `http://www.massivesoftware.com/` Last Visited: Monday, June 6, 2005.

[45] A. Mehrabien and J. Russel. *An Approach to Environmental Psychology*. Cambridge MA MIT Press, 1974.

[46] V. L. Michael and J. E. Laird. Developing an artificial intelligence engine. In *Proceedings of the Game Developers Conference*, pages 577–588, San Jose, CA, March 1999.

[47] M. Minsky. *The society of mind*. Simon & Schuster, New York, 1985.

[48] M. Morales, S. Rodriguez, and N. M. Amato. Improving the connectivity of PRM roadmaps. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 4427–4432, 2003.

[49] MOVIE. Movie models homepage, 2004. `http://www.give.nl/movie/moviemodels` Last Visited: Monday, June 6, 2005.

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

[51] M. H. Overmars. Algorithms for motion and navigation in virtual environments and games. Technical report, MOVIE, 2003. `http://www.give.nl/movie/publications/utrecht/motionPlanningVE.pdf` Last Visited: Monday, June 6, 2005.

[52] R. Picard. *Affective Computing*. The MIT Press, 1997.

[53] D. Pottinger. Coordinated unit movement. *Game Developer*, pages 42–51, January 1999.

[54] D. Pottinger. Implementing coordinated movement. *Game Developer*, pages 48–58, February 1999.

[55] M. J. Quinn. *Parallel programming in C with MPI and OpenMP*. Boston: McGraw-Hill Higher Education, 2004.

[56] J. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 431–459, 1995.

[57] C. W. Reynolds. Boids: Background and update. `http://www.red3d.com/cwr/boids/` Last Visited: Monday, June 6, 2005.

[58] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[59] C. W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, 1999.

[60] C. W. Reynolds. Interaction with groups of autonomous characters. In *Game Developers Conference 2000*, pages 449–460, 2000.

[61] B. Robertson. Innovative computer graphics techniques help weta digital create tolkien's middle-earth. *Computer Graphics World*, December 2001.

[62] J. J. Rousseau and A. Bloom. *Emile: Or, On Education*. Basic Books, 1979.

[63] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[64] G. Sanchez and J. C. Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2112–2119, 2002.

[65] O. Schnapauff and J. Schule. Simulating and visualizing natural flocking behavior. Technical report, Institute of Scientific Computing, TU Braunschweig, 1998. `http://citeseer.ist.psu.edu/schnapauff98simulating.html`.

[66] D. Ezra Sidran. Current methods to create human-level artificial intelligence in computer simulations and wargames. *the Modeling and Simulation Journal*, 2004.

[67] SOAR. University of Michigan SOAR project, 2004. `http://sitemaker.umich.edu/soar` Last Visited: Monday, June 6, 2005.

[68] Valve Software. Counter strike online homepage, 2004. `http://www.counter-strike.net` Last Visited: Monday, June 6, 2005.

[69] W. Sterren. Terrain reasoning for 3D action games. In *Game Developer Conference*, 2001.

[70] K. Sugihara and I. Suzuki. Distributed motion coordination of multiple mobile robot. In *IEEE Intl. Symp. on Intelligent Control*, pages 128–143, Philadelphia, PA, September 1990.

[71] P. Svestka and M. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998.

[72] P. Sweetser. Current AI in games: A review. `http://www.itee.uq.edu.au/~penny/GameAIReview.pdf`, 2002.

[73] FEAR TEAM. Fear project homepage, 2004. `http://fear.sourceforge.net` Last Visited: Monday, June 6, 2005.

[74] D. Terzopoulos. Artificial life for computer graphics. *Communications of the ACM*, 42(8):32–42, August 1999.

[75] D. Terzopoulos, X. Y. Tu, and R. Grzeszczuk. Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Journal of Artificial Life*, 1(4):327–351, 1994.

[76] J. Thierbach. Unofficial half-life faq, November 1997. `http://www.bluesnews.com/faqs/half-life.html` Last Visited: Monday, June 6, 2005.

[77] B. Tomlinson and B. Blumberg. Alphawolf: Social learning, emotion and development in autonomous virtual agents. In *First GSFC/JPL Workshop on Radical Agent Concepts*, Greenbelt, MD, 2002.

[78] X. Y. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behaviour. In *Computer Graphics*, volume 28, pages 43–50, 1994.

[79] B. Ulicny and D. Thalmann. Crowd simulation for interactive virtual environments and vr training systems. In *Eurographics Workshop on Animation and Simulation*, pages 163–170, 2001.

[80] C. Unsal and J. S. Bay. Spatial self-organization in large populations of mobile robots. In *1994 IEEE International Symposium on Intelligent Control*, August 1994.

[81] R. Vandeveer, M. Menefee, and G. Sinclair. Group behavior, 2004. `http://www.tech.purdue.edu/Ols/courses/ols252/slides/chapter8.ppt` Last Visited: Monday, June 6, 2005.

[82] J. Velasquez. Modeling emotions and other motivations in synthetic agents. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, 1997.

[83] M. Warren and J. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Supercomputing' 92*, pages 570–576, 1992.

[84] O. E. Wood. Investigation into flocking using boids technology, 2003. `http://www.coldclimate.co.uk/academic/` Last Visited: Monday, June 6, 2005.

[85] S. Woodcock. Games making interesting use of artificial intelligence techniques. `http://www.gameai.com/games.html` Last Visited: Monday, June 6, 2005.

[86] S. Woodcock. Flocking: A simple technique for simulating group behavior. In In Mark Deloura, editor, *Game Programming Gems*, pages 305–318. Charles River Media, Hingham, MA, 2000.

[87] S. Woodcock. Flocking with teeth: Predators and prey. In In Mark Deloura, editor, *Game Programming Gems 2*, pages 330–336. Charles River Media, Hingham, MA, 2001.

[88] R. E. Wray, J. E. Laird, A. Nuxoll, D. Stokes, and A. Kerfoot. Synthetic adversaries for urban combat training. In *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*, San Jose, CA, July 2004. AAAI Press.

[89] Bo Zhou and Suiping Zhou. Parallel simulation of group behaviors. In *Winter Simulation Conference*, Washington DC, USA, December 2004.

[90] Bo. Zhou and Suiping Zhou. Motion planning for group movement simulation in complex environments. In *CyberGames 2005: International Workshop on Games Research and Development*, Singapore, 28 March 2005.

[91] Bo Zhou and Suiping Zhou. Motion planning for group movement simulation in dynamic environments. *International Journal of Modelling and Simulation*, accepted.