Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

PARALLEL SIMULATION OF GROUP BEHAVIORS

Bo Zhou
Suiping Zhou

School of Computer Engineering
Nanyang Technological University
Singapore, 639798, SINGAPORE

ABSTRACT

Group behaviors, e.g. birds flocking, are widely used in virtual reality, computer games, robotics and artificial life. 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 this paper, 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.

1 INTRODUCTION

The aggregate motion of a group, e.g. a flock of birds, is common in nature. A flock consists of individual birds, each interacting with its neighboring members. Simulating the group behaviors of a large number of moving objects has attracted the attention of researchers in different fields such as virtual reality (Musse and Thalmann 2001), computer animation (Bayazit, Lien and Amato 2002; Reynolds 1987) and artificial life (Terzopoulos, Tu and Grzeszczuk 1994). In flocking simulation, the moving objects in a flock are called "boids" as in Reynolds (1987). 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 (Reynolds 1987) 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 (Lorek and White 1993). 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. 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. 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.

The remainder of this paper is organized as follows. In Section 2, the related work in behavioral model and its parallel simulation are described. In Section 3, 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 4. Experimental results are presented in Section 5. Finally, in Section 6, concluding remarks are made.

2 RELATED WORK

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. Parallel N-Body simulation of particle systems have been widely investigated based on a class of

tree-based methods. Warren and Salmon (1992) and Liu and Bhatt (1994) have implemented Barnes-Hut's O($NlogN$) method (Barnes and Hut 1986) using the message passing programming paradigm. Some parallel implementations of the fast multipole method (FMM) (Greengard and Rokhlin 1987) have also been investigated in Blelloch and Narlikar (1994).

Reynolds proposed a distributed behavioral model in Reynolds (1987), which is based on simulating the behavior of each member independently. 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.

Lorek and White (1993) implemented Reynolds' model on a Meiko Transputer System. In his implementation, each processor deals with a fixed number of birds. However, with p processors, its communication scheme 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 SEQUENTIAL FLOCKING ALGORITHM

3.1 Flocking Behavioral Model

We mainly focus on the leader-following behavior described in Reynolds (1999). 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
- **Cohesion**: the attraction of the center of neighboring boids
- **Alignment**: the desire to fly in the same direction with others

We can see in Figure 1 that the three rules are only related to the local information of the group. 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.

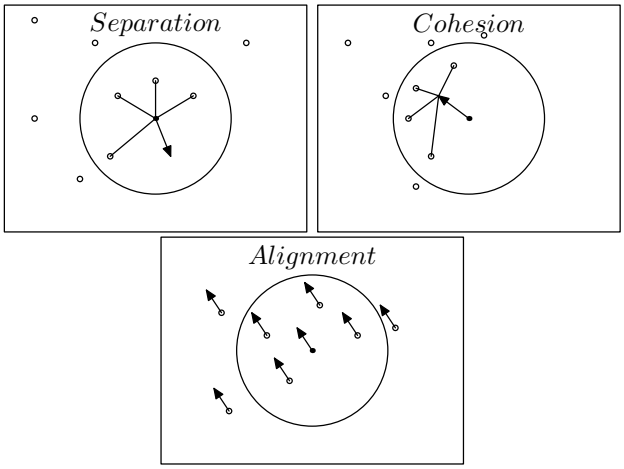

Figure 1: Three Basic Rules of Flocking Simulation

3.2 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. Figure 2 shows the two additional rules.

- **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.

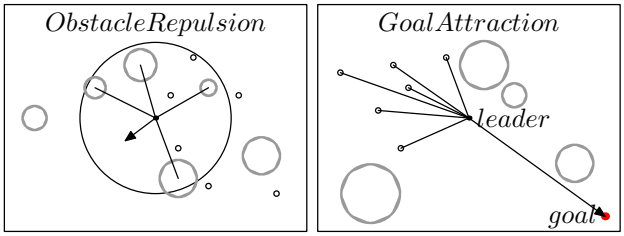

Figure 2: Two Additional Rules of Flocking Simulation

3.3 Algorithmic Considerations

The whole process of the simulation is composed of frames. Figure 3 shows the algorithm of the simulation in one frame.

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 predefined value. It is also important to fine-tune the parameters of these rules to generate realistic flocking behaviors.

```
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: Sequential Simulation Algorithm

4 PARALLEL FLOCKING SIMULATION

The force calculation 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 the 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.

4.1 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. Figure 4 shows this kind of 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**: We see in Figure 5 that 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. 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 the beginning of the simulation, all-to-all communication is used to get the global environment information.

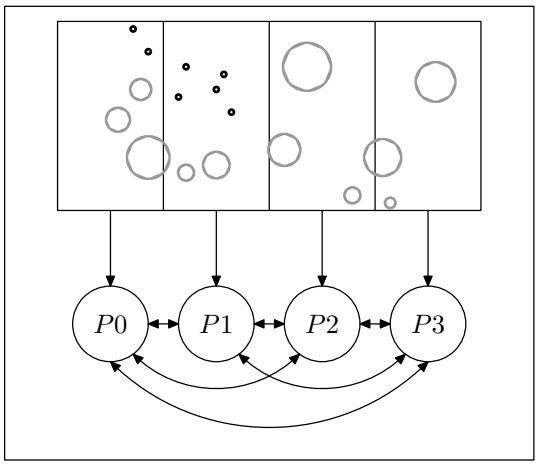

Figure 4: All-to-all Communication

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 paper, 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.

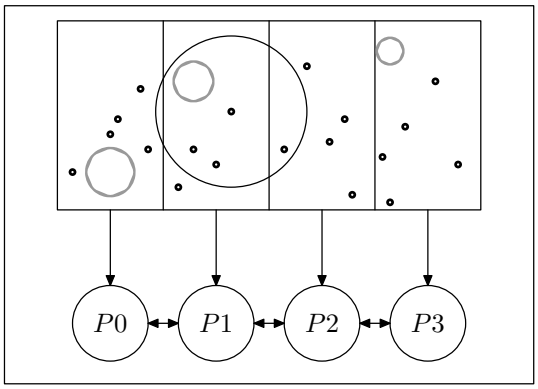

Figure 5: Near-neighbor Communication

4.2 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 need 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. 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 as seen in Figure 6.

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.

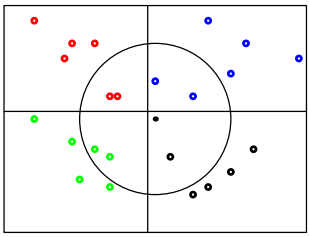

Figure 6: Neighboring Boids from All Processors

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 prespecified threshold, the load balancing algorithm needs to re-partition the simulated space and re-distribute the successive spaces among the processors. Figure 7 shows that, after the load balancing, each processor will manage a new space, together with new boids and new boundary information.

4.3 Parallel Flocking Algorithm

We develop the parallel flocking algorithms using MPI (Quinn 2004) to investigate the performance of our proposed mechanisms.

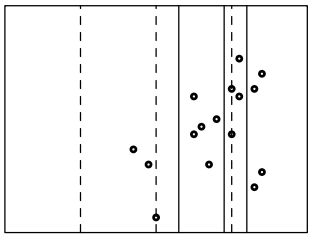

Figure 7: Dynamic Partition of the Simulated Space

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 8 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 8: 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, evaluate the degree of imbalance. If the load imbalance among the processors exceeds the threshold, load balancing mechanism is invoked. If the imbalance is still acceptable, jump to the beginning of the interaction, and produce the next frame.

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 are more than the threshold, load balancing is invoked. It re-calculates and update the boundary information, then 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 (Lorek and White 1993) are shown in Table 1.

Table 1: Comparison of the Two Parallel Algorithms

Main Characteristics	Algorithm	
	Lorek and White's	Ours
All-to-all comm.	$\surd$	$-$
Near-neighbor comm.	$-$	$\surd$
Keep even distribution	$\surd$	$-$
Load balancing	$-$	$\surd$

5 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 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$\times$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 9 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 10 shows that the execution time reduces significantly when more processors are used, especially when the flock is composed of a large number of boids. The configuration with $P = 1$ means that one processor simulates the whole flock, so it is equivalent to a sequential algorithm. Obviously we obtain a better performance through parallel simulation.

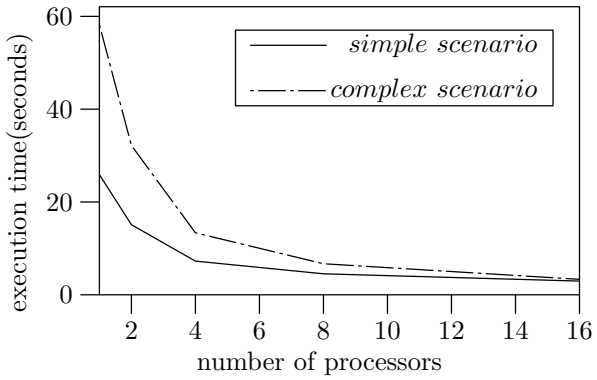

Figure 9: Performance of Varying Environment

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.

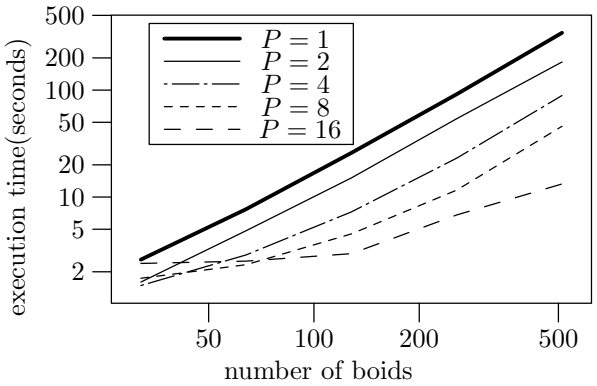

Figure 10: Performance of Various Configurations

Figure 11 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 help to limit the number of neighboring boids especially when P is lager than 4.

Figure 12 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. The simple load balancing strategy we used performs well to reduce the load imbalance.

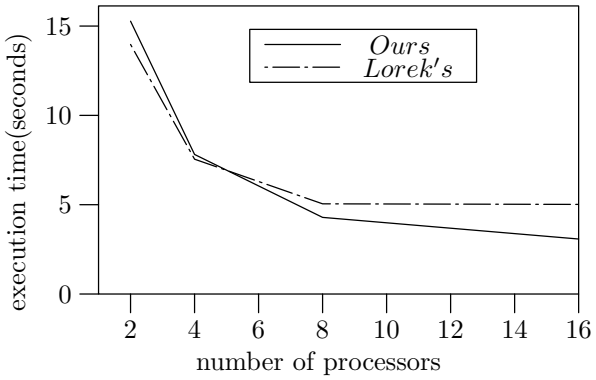

Figure 11: Comparison between Lorek and White's and Our Algorithm ($N = 128$)

If load balancing is frequently invoked, it will lead to inefficiency. So we need to define a suitable threshold to keep a good balance. Table 2 shows the analysis of the algorithm with load balancing.

From Table 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 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.

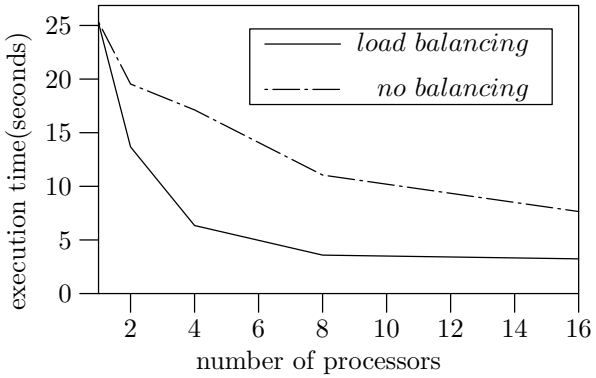

Figure 12: Performance of Algorithms with and without Load Balancing

Table 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

6 CONCLUSIONS

In this paper, we propose a parallel algorithm for the flocking behavior simulation. With our parallel algorithm, the simulated environment is divided into successive partitions, and one partition is allocated to one processor. In the simulation, each processor only needs to communicate with its logically neighboring processors for collecting necessary information. Load balancing is used to help distributing boids evenly among processors. We implement 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 is large.

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. Motion planning method will also be investigated to realize more sophisticated group behaviors in a dynamic environment.

REFERENCES

Blelloch, G., and G. Narlikar. 1994. A practical comparison of N-body algorithms. In *Dimacs Implementation Challenge Workshop* 30: 81–96.

Barnes, J., and P. Hut. 1986. A hierarchical O(NlogN) force-calculation algorithm. *Nature* 324: 446–449.

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

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

Lorek, H., and M. White. 1993. Parallel bird flocking simulation [online]. Available online via <http://citeseer.ist.psu.edu/lorek93parallel.html> [accessed August 3, 2004].

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

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

Quinn, M. J. 2004. *Parallel programming in C with MPI and OpenMP*. 1st ed. Columbus: McGraw-Hill Higher Education

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH' 87*. Reprinted in *Computer Graphics* 21 (4): 25–34.

Reynolds, C. W. 1999. Steering Behaviors For Autonomous Characters. In *Proceedings of Game Developers Conference* 763–782.

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

Warren, M., and J. Salmon. 1992. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing'92* 570–576. IEEE Computer Society.

AUTHOR BIOGRAPHIES

BO ZHOU is currently a Ph.D student in School of Computer Engineering at Nanyang Technological University (Singapore). He received his B.Eng. in Computer Science and Engineering from Huazhong University of Science and Technology (P.R. China). His current research topic is on group behaviors in complex environments. His e-mail address is <zhou0021@ntu.edu.sg>.

SUIPING ZHOU is currently an Assistant Professor in the School of Computer Engineering at Nanyang Technological University (Singapore). Previously, he was a Post-doctoral fellow at Weizmann Institute of Science (Israel). He received his B.Eng., M.Eng. and Ph.D in Electrical Engineering from Beijing University of Aeronautics and Astronautics (P.R. China). His current research interests include: distributed interactive applications, hybrid control systems and group behaviors in complex environments. His e-mail address is <ASSPZhou@ntu.edu.sg>.