Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 918497, 15 pages
doi:10.1155/2012/918497

Research Article

Parallel Motion Simulation of Large-Scale Real-Time Crowd in a Hierarchical Environmental Model

Xin Wang,[1] Jianhua Zhang,[2] and Massimo Scalia[3]

[1] *College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China*
[2] *TAMS Group, Department of Informatics, University of Hamburg, Vogt-Koelln-Straße 30, 22527 Hamburg, Germany*
[3] *Department of Mathematics, Sapienza University of Rome, Piazzale Aldo Moro 2, 00185 Rome, Italy*

Correspondence should be addressed to Xin Wang, xinw@zjut.edu.cn

Received 17 February 2012; Accepted 28 March 2012

Academic Editor: Carlo Cattani

This paper presents a parallel real-time crowd simulation method based on a hierarchical environmental model. A dynamical model of the complex environment should be constructed to simulate the state transition and propagation of individual motions. By modeling of a virtual environment where virtual crowds reside, we employ different parallel methods on a topological layer, a path layer and a perceptual layer. We propose a parallel motion path matching method based on the path layer and a parallel crowd simulation method based on the perceptual layer. The large-scale real-time crowd simulation becomes possible with these methods. Numerical experiments are carried out to demonstrate the methods and results.

1. Introduction

Real-time crowd simulation is one of the important research directions [1, 2] in computer games, movies, and virtual reality. In the last decade, Shao et al. [3] proposed a multilevel model for the virtual crowd simulation in the face of visual effects, perception, routing, interactive and other issues directly and efficient support. Virtual environment modeling consists of two parts, that is, geometric environment modeling and nongeometric environment modeling [4]. Geometric environment model corresponds to the geometric layer of the hierarchical environmental model, and nongeometric environment model corresponds

to the topology layer, the path layer, and the perception layer. There have existed some parallel methods [5], which can fast compute a scene path map based on the topology layer. However, they lacked parallel methods to accelerate real-time crowd simulation based on the path layer and the perception layer.

Given a motion path, matching human movements with it is a common problem in the field of character animation. Usually artist manually tuned the motion data. However, when lots of human movements need to be matched with lots of paths, only using manual method will be impractical and cannot meet the real-time requirements. Based on our analysis, we find that a parallel matching algorithm is suitable for motion path matching. Because motion matching between paths is independent of each other, the path segments among every two points within the calculation of the discrete sampling are independent. Based on the above investigations, this paper employs a parallel motion path matching algorithm based on the path layer.

Researchers often use the agent-based simulation model in crowd simulation. We found that single agent status updates only need to consider it within a limited range state of the world. Based on this observation, we designed a parallel crowd simulation method, employing a parallel update strategy and dividing the scene into bins (each bin is a square area; agents are distributed among all bins); we then determined the scope of each agent. For the nonadjacent bins, agents can employ parallel update strategy, which is divided into four batches of parallel updates. Experimental results show that this strategy can greatly increase update efficiency.

In the real world, each person makes appropriate movement decisions according to the state of the environment around him [6, 7]. For example, when a car approaches, people will maintain their status or change their walking direction according to the car's driving direction, speed, and distance. The agent model proposed by Reynolds [8] can simulate these situations. The current research employed a similar but relatively simple agent model. Our agent is controlled by three forces: [9] avoiding the force of obstacles; avoiding collision with other agent forces; following path force [10, 11]. The three forces are in order of decreasing priority.

This paper presents parallel real-time crowd simulation algorithms based on a hierarchy environmental model and fully taps the multilevel environmental model in the parallelism, making the simulation of large-scale real-time crowds possible.

2. Related Work

Given a motion path, matching human movements with it is a common problem in the field of character animation. In the game industry, a relatively simple and efficient way is to loop the motion clip while the character moves along a motion path. This method makes the movements of characters appear mechanical, monotonous, and unrealistic in terms of imitating body movements. To achieve realistic effects, some scholars have proposed complex motion synthesis methods [12–14]. The general approach first uses motion data, which are captured to construct a graph-like data structure. In searching this data structure for the right movement sequences to match input path, these sequences meet specific constraints that stitch up human posture, finally forming the results. When matching motion data with motion path, there exist some difficulties [15] as follows: (1) in the construction of the motion graph, if the graph is large and complex, determining what motion can be synthesized from the graph is difficult, and the range of new movements is unknown [16]; (2) the structure

of the motion graph is complex, which takes a lot of time to search on the graph each time; hence, this structure cannot improve system performance [17]; (3) for the presence of path constraints, there are some low-level connection relationships in the motion graph, so there will be some unnecessary local shaking when synthesizing motion [12]; (4) some constraints need to be considered (e.g., obstacles); however, such methods are difficult to estimate.

Lau and Kuffner [18] described a similar method for this paper. The method organizes motion data into a finite state machine (FSM); each state represents a high-level semantical movement, such as walking and running. If two states can be connected together, there will be one connection in the state machine. Motion state machine is used to find the character's motion data when synthesizing motion, which can overcome the difficulties mentioned above. For multiple paths, this paper divided long path into short path segments. The length of these short path segments is almost equal to the motion segments in a motion state machine. Thus, when performing matching work, the efficiency of matching can be improved. This paper also used parallel computing strategy to greatly reduce the time required for each planning step. In addition, when using motion state machine to match, a benefit arises; that is, while the current matching segment crosses paths with other motion path segments, a collision situation is possible; if such a situation happens, there will be no displacement motion data (idle) for the role to effectively avoid the collision. On the other hand, when more human movements are required to match more paths, solely using such methods cannot meet the real-time requirements.

The agent-based crowd simulation model is accorded the characteristics of crowd movement in the real world and has a very flexible control strategy (e.g., controlling each parameter of each agent). Therefore, agent-based crowd simulation model is widely used. However, for large-scale real-time crowd simulation, each agent needs to update its own status according to its surrounding environment, and each agent is a dynamic obstacle to other agents. Computing cost increases rapidly as the number of agent increases [19] in the face of time complexity $O(n^2)$. Thus, when n is large, the time required for each frame will significantly increase, making the real-time status update of each agent impossible. Hence, we need to find a new method to do large-scale crowd simulation.

3. Overview

The system consists of two parts: (1) hierarchy environment model and (2) multiple parallel algorithms based on the hierarchy (see Figure 1).

The hierarchy environmental model [20], as shown in Figure 1 (right panel), consists of two parts: geometric and non-geometric environment model. The non-geometric environment model includes the topology layer, path layer, and perception layer.

Based on the multi-level environment, the topology layer, path layer, and perception layer employ different parallel algorithms to speed up real-time crowd simulation, respectively, as shown in Figure 1 (left panel). We employed the existing Voronoi diagram algorithm [4] based on topology layer to segment the scene quickly and get the path layer map. We also used parallel motion path matching based on path layer to quickly match human motion for path, as well as parallel real-time crowd simulation based on perception layer to fully tap the crowd behavior in parallelism.

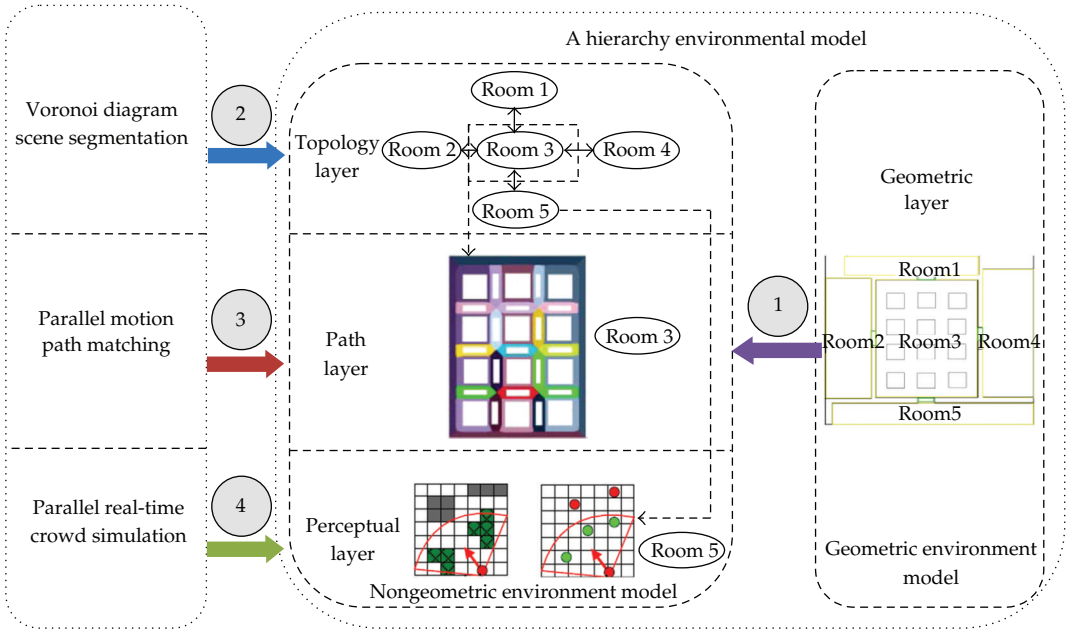

Figure 1: System overview.

4. Parallel Motion Path Matching Based on Path Layer

This section explains parallel motion path matching based on path layer. The section contains three parts: (1) motion state machine, (2) motion path parameterization, and (3) extraction of motion segments.

4.1. Motion State Machine

The main differences between the motion state machine proposed by Kovar et al. [12] and the motion state machine constructed in this paper are as follows: (1) trajectory arc length of the node of motion segments used by each state is fixed and equal in this paper's motion state machine; (2) the length of most of the motion segments in this paper is longer than that in Kovar et al. [12]. For example, there are 100 frame movements about walking used in this paper, but the length of the movements in Da Silva et al. [21] is only 25 frames; (3) the motion state machine in this paper employed group-based hierarchical status node, which can ensure that "walk left" and "run left" are on the same level, and all directional run and walk movements comprise a position move node on the up level. Three facts exist in the motion synthesis process, which lead to the above differences: (1) this paper used sectional matching; (2) the movement range of the motion path to be matched is large; (3) the connection relationships among states are more complex.

The motion state machine (MSM) used in this paper is shown in Figure 2. The MSM has a total of 12 base states, which have their own names and include 36 motion segments (except start and end states). The motion segments' arc length of "Crawl," "Squat move," and "Stride over" is indeterminate, because these motion segments are mainly used to deal with the path segments about obstacles. These are marked directly by users to explain which

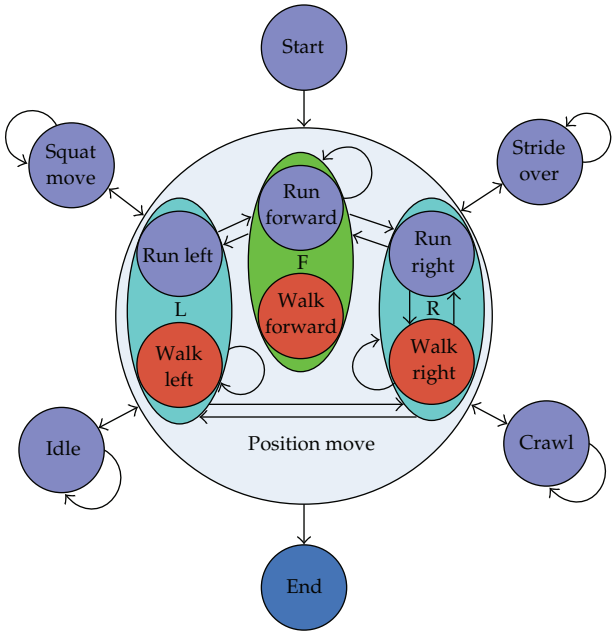

Figure 2: Finite motion state machine.

motion segments are used, so that the system does not need to match in the rear. The function of the "Idle" state's motion segments is to make the virtual characters wait, so that the system can avoid the collision between virtual characters. The states, which include "run forward," "run left," "run right," "walk forward," "walk left," and "walk right," describe the ways in which virtual characters can match according to the specific path situation. Furthermore, the motion path's arc length of motion segments is equal, which makes the matching of motion path segments easy.

There are some directed edges among states in MSM. These directed edges express whether two motion segments can be connected. For example, there is a directed edge between "run right" and "walk right," which means that the motion segments of "run right" can stitch with the motion segments of "walk right." Similarly, the motion segments of "walk left" can stitch with the motion segments of "run left." The motion machine in this paper adopts a group-based hierarchical status node. For example, "walk forward" and "run forward" constitute F, "run left" and "walk left" constitute L, and "run right" and "walk right" constitute R. Finally, L, F, and R constitute locomotion. The child states can inherit the father states' connection. For example, the connection between F and R explains that "walk forward" and "run right" can connect with each other, and the connection between the "crawl" state and l locomotion can lead to the connection between "crawl" and "run left."

This paper also prepares some transitional motion segments. These motion data are used to connect the transitional fragments among motion segments. There are eight frames. The transitional fragments exist at the beginning and the end of motion segments.

4.2. Motion Path Parameterization

In this paper, there is a path planner responsible for generating a parameterized original movement path called "T" and decides the grid size of path planning pace. If the planning space is planed in a lattice structure, which is composed of $m * m$ square, the space between $2d$ point sequence is m, but in the specific application, the space of sample points cannot keep the same with the sample degree of the original path. For example, in this research, the human root node movement size between two frames is s in the MSM. However, the motion clips in the MSM were prepared before motion synthesis and cannot be resampled. The system initially needs to use some fitting methods to parameterize the original path and then resamples the parametric path based on the desired sampling size. The resulting path can be used to match the crowd animation better in the following phase. In detail, our method constructs the cubic polynomial curve $p(u)$ with C_1 continuous based on the original data point. If there is a curve formula called $p(u)$, the system begins with a starting point p_0 and records those points $p_1, p_2, \ldots, p_n$, whose arc length spacing is s and corresponding parameters $u_1, u_2, \ldots, u_n$. The characteristics of cubic polynomial spline curve with C_1 continuous are simple and easy to control, and human motion path is generally not so smooth that curve with C_1 continuous is enough to make the motion path smooth through fitting.

In addition, the algorithm needs the corresponding tangent vector of each sample point. The system can calculate $p(u)$ curve's tangent vector in $p_0, p_1, \ldots, p_n$ through parameters $u_0,\ u_1, \ldots, u_n$. Then, it can represent the motion path after normalizing with the coordinates of parameters point and tangent vector, called T_1:

$$T_1 = \left\{ p_i, \vec{\mathbf{p}}_i \mid i = 0, 1, \ldots, n \right\}. \tag{4.1}$$

The arc length in our motion segments in the MSM is almost equal. The arc length of motion state i is represented by li. If li is n times as long as s, motion path T_1 used in experiment general is n times longer than s. Hence, it can cut up the whole path into many small segments whose length is $n * s$. For example, suppose there are $n + 1$ points from p_j to p_{j+n}. The algorithm can find motion segments called motion_k in the MSM, which is similar to the path curve shape. If there are some obstacles in the path, direct matching cannot be performed. The path is marked by specific segments of motion data through interaction. Then, it can stitch directly in the matching phrase.

There are some crossings of motion paths and obstacles in T_1. Thus, matching directly to T_1 is not enough. An artificial mark on this cross-path, such as "crawl" state, is needed. In the matching, when meeting this path, motion data are directly found under the "crawl" state and are associated with motion path T_2:

$$T_2 = \left\{ p_i, \vec{\mathbf{p}}_i, \left(\text{motion}(j,t) = \text{motion}_k\right)_r \mid i = 0, 1, \ldots, n;\ 0 \le j < n;\ 0 < t \le n - j;\ r = 0, 1 \ldots, u \right\}. \tag{4.2}$$

The path segments composed of continuous t points from p_j have been associated with motion data called motion_k; r means that this mark is the r-section in all $u + 1$ marks. Figure 3 shows the normalized conversion process from curve T to curve T_2.

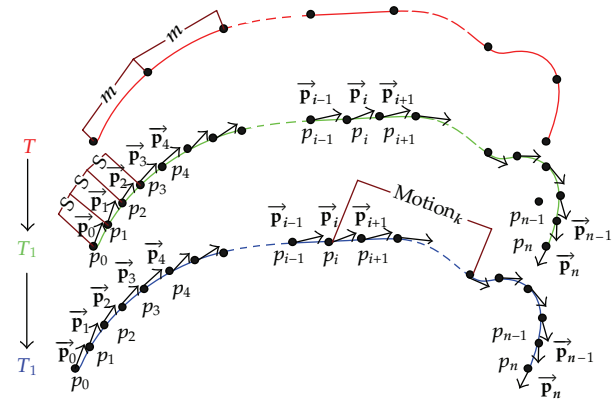

Figure 3: Normalized conversion process from curve T to curve T_2.

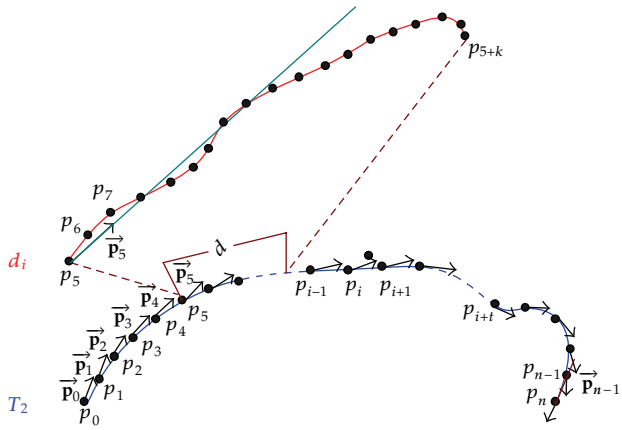

Figure 4: Schematic diagram of T_2 segmentation.

4.3. Extraction of Motion Segments

T_1 changes into T_2 through parametric resampling and marking the specific segment. Setting the path consists of $n + 1$ points, and total arc length is $n * s$. The basic motion segment of the root node's path curves arc length is certain, set as d, such as "walk" and "run." d is also integer times as s, set as k. Thus, T_2 needs to be cut into path segments, whose arc length is d. The set of this path segments is D. Consider

$$D = \{d_i \mid i = 0, 1, \ldots, D_{\text{size}} - 1\}, \tag{4.3}$$

where d_i means the ith motion path segment. D_{size} means the total number of segments after cutting:

$$d_i = \{p_i \mid i = b + 0, b + 1, \ldots, b + k, b = d_i \text{ starting coordinates}\}. \tag{4.4}$$

Figure 4 shows the T_2 segmentation process.

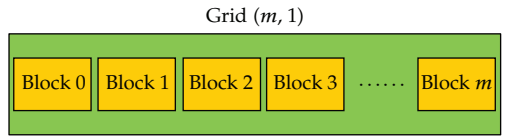

Figure 5: Schematic diagram of Grid $(m, 1)$.

The next work that needs to be done is using the distance formula D to find proper motion segment motion_k for each d_i.

This paper designs a one-dimensional Grid (motionNum, 1) for single curve. motionNum means the number of motion segments in the database, and one Grid has motionNum blocks. Each block is responsible for calculating its representative motion segments. For example, block0 is only responsible for calculating the matching degree between the current line and the 0th motion segment. Each block is one-dimensional. For example, in block (PointNum, 1), PointNum means the number of discrete points currently calculating the curve segment matching. Each thread in the block means the distance between a discrete point in the curve and the corresponding point in the motion segment. For example, thread0 is responsible for calculating the distance of 0th discrete point. The formula is

$$\text{distance}_i = \sqrt{\text{CurvePoint}_i - \text{MotionPoint}_i}. \tag{4.5}$$

Each thread only needs to calculate its own data, and it does not need to interact with other threads. After the final computation, thread0 accumulates matching value, calculating by each thread, and acquires the total matching value between the curve and the motion segment. The computation formula is

$$\text{MatchDegree} = \sum_{i=0}^{n} \text{distance}_i. \tag{4.6}$$

After calculating the matching value with all motion segments, the minimum matching value is obtained through comparison. The final result is

$$\text{Motion} = \left\{ j \mid \text{MatchDegree}_j = \min_{i \in (0,n)} \left(\text{MatchDegree}_i \right) \right\}. \tag{4.7}$$

The matching effect of single curve is shown in Figure 6, and its internal thread organization way is depicted in Figure 5. Before calculating the distance between a discrete point in curve and the corresponding point in motion segment, every point in a motion segment needs to do a rotating translation operation. This step is necessary because motion segment only converts to a curve's local coordinate. The matching value can be obtained accurately. Now, the system selects thread0 to calculate rotation and offset. Meanwhile, the other threads wait for the calculating result of thread. Then, they share the calculating result of thread0.

We need to design a two-dimensional Grid (motionNum, CurveNum) to extend to multiple curves, as shown in Figure 7. motionNum means the number of motion segments.

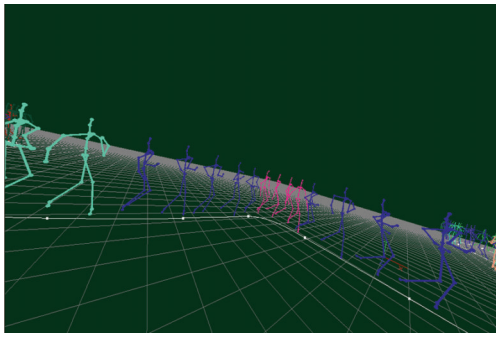

Figure 6: Matching effect of single curve.

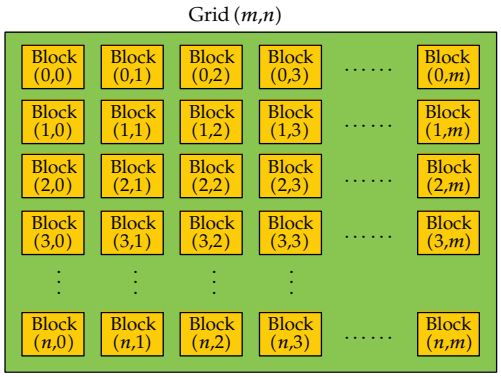

Figure 7: Schematic diagram of Grid (m, n).

CurveNum means the number of curves. Others are similar to the single curve. The matching effect of multiple curves is illustrated in Figure 8.

This method can achieve better expansibility. If there are more motion segments that need to match, then the dimension of Grid only increases.

5. Parallel Real-Time Crowd Simulation Based on Perception Layer

This section explains how to parallel real-time crowd simulation based on perception layer. This section contains three parts: (1) scene segmentation; (2) nearest neighbor query; (3) parallel real-time crowd simulation.

5.1. Scene Segmentation

In agent-based crowd simulation, each agent has its own perceived range. In each update, each agent must query the scope of its perception to obtain the range of obstacles or information of other agents. Then, based on the information to calculate the forces acting on the agents, their speed and direction can be adjusted, and the location can be updated. Neighbor queries generally employ serial query that is executed when the current agent is updated. Then, the next agent begins to update. In this algorithm, using parallel neighbor

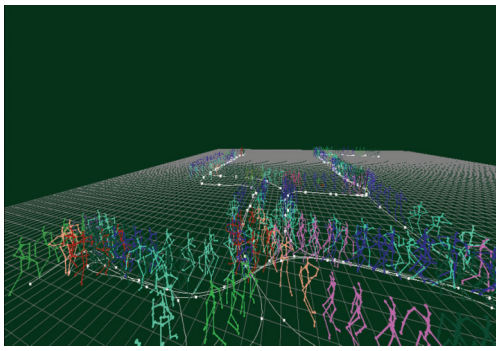

Figure 8: Matching effect of multiple curves.

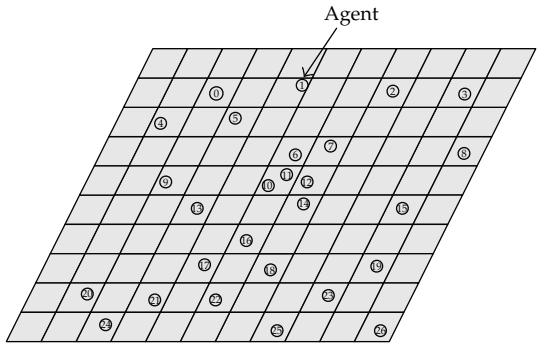

Figure 9: Scene segmentation and agent distribution (circle represents the agent).

query, the main query is the bin, rather than each agent. Through these neighbor queries, the bin can be obtained, in which agents need to consider all the potential neighbor agents. Before the implementation of parallel neighbors' query, the entire scene needs to be evenly segmented to get all the information about the bin.

The segmentation needs to be completed in the initialization process. The size of the entire scene is 1200 × 1200; setting the side length of each bin is 4 for the square, so that the whole scene is to be divided into 300 × 300 bins. The square side length is 4, because the sensing range of agent sets a circular area that is a query radius of 4. The formula is as follows:

$$\text{queryRadius} = \text{predTime} * \text{maxVelocity} * 2. \tag{5.1}$$

The predTime is the forward predictive time, set as 1 s. maxVelocity is the agent's maximum move speed, set as 2 m/s. In the query process, only 8 bins need to be considered, which are around the center bin.

As shown in Figure 9, the scene is evenly divided into a number of bins, the scene of the agents located in each bin. There are multiple agents in a bin; however, a bin can also have no agent.

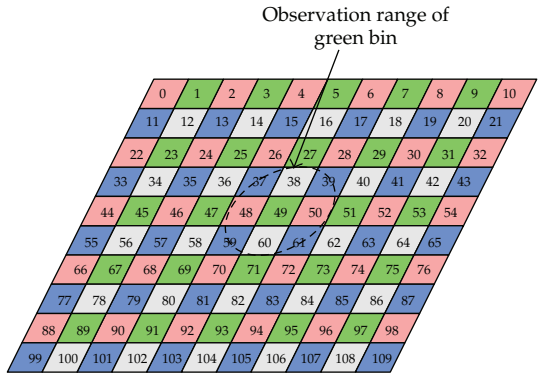

Figure 10: Observation range of bin (dashed circle).

5.2. *Nearest Neighbor Query*

After scene segmentation and statistical distribution among the agents in each bin, the nearest neighbor query is employed. In the scene segmentation step, each Agent's observation range is a circular area of radius 4, and the bin side is also 4. Thus, the algorithm only needs to check the bin and approximately 8 bins. Figure 10 shows that the number of green bins is 49 and the observation range is the dashed circle. Hence, the bin numbers that need to be checked are 37, 38, 39, 48, 49, 50, 59, 60, and 61. As the neighbor queries between the bins only read data with no write operation, this paper chooses the parallel query; the query results are saved in the corresponding data structures.

5.3. *Parallel Real-Time Crowd Simulation*

After completing the nearest neighbor querying, crowd velocity, and position updating, the next Agent begins to update. At the step of parallel nearest neighbor query, the algorithm has queried all the neighbors of the bin. However, as the Agents of the adjacent bin would interact, this step of the update operation cannot be completely parallel. For example, for bin 48 and bin 49, if the two parallels update, there would be a read/write conflict. The reason is that when the Agents of bin 48 are updating, the Agents of bin 49 should be considered; hence, the reading data may be outdated Agent data of bin 49, and vice versa. Therefore, this paper chooses partial parallel update strategy; steps are described below in detail.

Referring to the behavior rules of pedestrians in the real world and traditional Agent model, the Agent model used in this paper is acted upon by three kinds of forces, namely, (1) avoid the force of obstacles; (2) avoid collision with other Agent forces; (3) follow the path of force. Figure 11 illustrates each kind of force.

Three kinds of forces decline in priority order. Avoiding the behavior of obstacles is the highest priority; that is, when Agent k detects it will crash into obstacle d, it also may have collided with Agent g, and then it would only consider avoiding obstacle d. Considering that the obstacle is stationary, if the Agent does not adjust its speed, it will be hit and move. Even if Agent k does not make an adjustment, Agent g itself can adjust the speed for Agent k. Avoiding force is calculated as follows. Agent forward predictive distance with the observed radius builds up a rectangular area; in the region of collision, we can find the nearest obstacle. As shown in Figure 11 for obstacle d, avoiding force (Force k) for the current Agent is pull

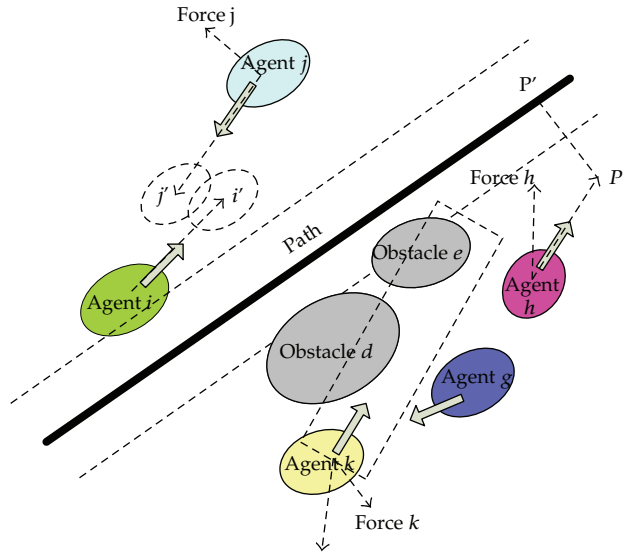

Figure 11: Schematic diagram of agent's three kinds of forces.

force and the obstacle repulsive force direction. Another situation is when Agent j moves to prediction j' position, and Agent i will reach position i at the same time. When the distance between position j and position i is less than the distance between the center of the radius of their observations, a collision will be assumed; thus, they will not be influenced by the force of path-following, but by the Agent force between the impact of avoidance, the avoidance of force as a lateral tension (Force j), to deviate the current running direction. In Figure 11, there is a lane in the path (dotted line parallel with the path and the distance between the path); the current updated Agent will forecast forward for some distance to predict the location of P projected onto the path P' point. If the distance between P' and P is greater than the lane, then the Agent's travel direction deviates from the path; otherwise, there is no deviation. As shown in Figure 11, Agent h would be acted upon by the force of path following (Force h), for the projection point P' of the direction of the Agent center of the connection to slowly move closer to the path direction.

This paper makes parallel updates on the Agent rates four times. As shown in Figure 10, the bin has a total of four colors. Each time parallel updates the bin that has the same color. When all colors have been updated, a general update is completed. Agents within the same bin have an impact on each other, but they cannot be updated in parallel; hence, the serial update can only be used. In the implementation process, each block represents a bin, following the Agent model above; meanwhile, according to the priority of various types of forces, the research also carries out the appropriate adjustments.

6. Experimental Results

In general, the experiment platform uses a computer of 4 core (Q9950 CPU at 2.83 GHz and 4 GB of RAM), equipped with a NVIDIA GTX 280 graphics card, which has 30 multiprocessors. Each multiprocessor has 8 cores; the total number of cores is 240. The CPU-based algorithm achieves serial algorithms, and the GPU-based algorithm achieves the proposed parallel algorithms in this paper.

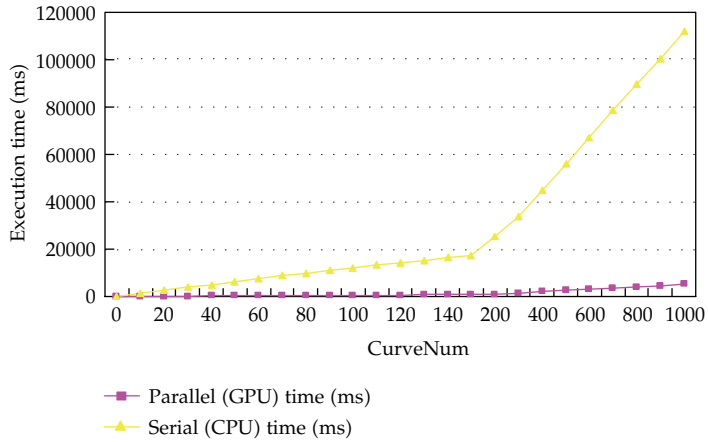

Figure 12: Comparison between CPU-based and GPU-based algorithms.

Table 1: Comparison between CPU and GPU execution time.

	10	100	200	500	1000
CPU execution time (ms)	1214.9124	11999.31	25312.8457	56183.2070	112101.531
GPU execution time (ms)	70.2647	549.388	1049.8966	2572.0297	5188.6655

Figure 12 clearly shows that in the path matching based on path layer, with the increase in the number of curves that are required for matching, the CPU's execution time increases significantly, whereas the GPU's execution time barely increases.

Table 1 lists the matching time based on the path matching on the path layer with the CPU matching algorithm and GPU matching algorithm under some curve lines. The data in the table indicate that GPU parallel algorithm would increase by about 20 times than the CPU algorithm in execution speed.

Figure 13 shows the frame rate comparison between GPU parallel algorithm and CPU serial algorithm in real-time crowd simulation based on the perceived layer.

As can be seen from Figure 13 GPU-based parallel computing speed is significantly higher than the CPU-based parallel computing speed in crowd simulation. With the increase in the number of Agents, CPU-based FPS drops very quickly. When the number reaches more than 5,000, real-time simulation results are not achieved. Although the GPU program number is above 10000 of the Agent, FPS can reach 24 or more, fully meeting the real-time requirement. When the number is 1000, the FPS of the GPU is 60.

In addition, as the curves indicate, the FPS of GPU-based algorithms is significantly higher than that of the CPU-based algorithms, but it still has not reached 10 or more times. The main reason is the speed of Agent updating; there are a lot of conditional executions, such as statements reducing the parallel computing efficiency of the GPU-based algorithm.

7. Discussion

Through the proposed parallel algorithms in this paper, we achieve a completely parallel real-time crowd simulation based on a hierarchy environmental model, making topology

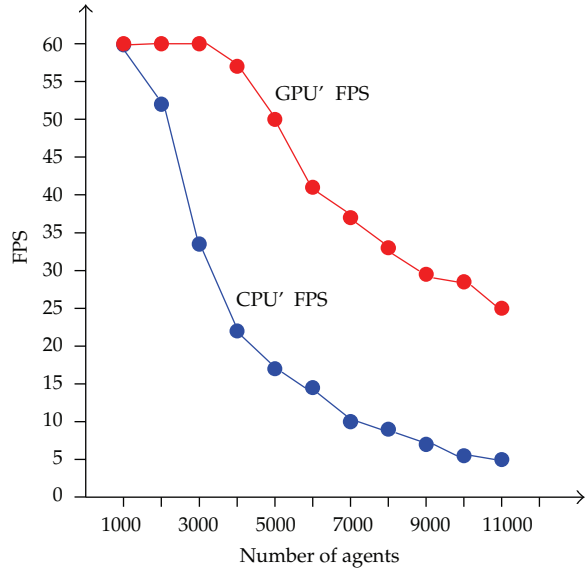

Figure 13: Performance comparison between CPU and GPU in crowd simulation.

layer, path layer, and perception layer have corresponding parallel algorithms to speed up the calculation.

This paper describes the detailed process with the path layer-based motion path matching, through the structure of motion state machine, setting up the transfer relationship among movement segments. Specifying the sequence of key points, this paper chooses cubic spine curve fitting to get a continuous path and then samples the sequence of points needed. We use the distance function to calculate the matching degree between the motion segment and the path, then accumulate the value of discrete points, and get the total matching degree of the corresponding segment. Moreover, this paper explains the use of parallel computing algorithms to accelerate and verify the algorithm through data analysis.

This paper introduces parallel computing algorithm based on the perception layer to achieve real-time simulation of large crowds. Using scene segmentation evenly, according to his own location, each Agent is assigned to the appropriate bin where the original calculations must be serialized into parallel computing. Each bin is a separate update unit.

Experimental results suggest that the proposed method in this paper is consistent with the serial method in effect, but efficiency has been greatly improved. Given that the Agent model used in this paper is relatively simple, the focus of future work is how to use more complex models to achieve a more realistic real-time crowd simulation, reduce the occurrence times of logic-based computing in the parallel algorithm framework, and further improve the algorithm's parallelism.

Acknowledgments

This work was supported by Natural Science Foundation of Zhejiang Province (Y1110882, Y1110688, R1110679), Department of Education of Zhejiang Province (Y200907765, Y201122434), and Doctoral Fund of Ministry of Education of China (20113317110001).

References

[1] N. Farenc, S. R. Musse, E. Schweiss et al., "A paradigm for controlling virtual humans in urban environment simulations," *Applied Artificial Intelligence*, vol. 14, no. 1, pp. 69–91, 2000.

[2] F. Tecchia, C. Loscos, R. Conroy et al., "Agent Behavior Simulator (ABS): a platform for urban behavior development," in *Proceedings of Games Technology Conference*, 2001.

[3] M. Shao, X. Wang, and Y. Hou, "Crowd evacuation simulation based on a hierarchy environmental model," in *Proceedings of the IEEE 10th International Conference on Computer-Aided Industrial Design and Conceptual Design: E-Business, Creative Design, Manufacturing (CAID & CD '09)*, pp. 1075–1078, November 2009.

[4] S. Chen, Y. Wang, and C. Cattani, "Key issues in modeling of complex 3D structures from video sequences," *Mathematical Problems in Engineering*, vol. 2012, Article ID 856523, 17 pages, 2012.

[5] M. Denny, "Solving geometric optimization problems using graphics hardware," *Computer Graphics Forum*, vol. 22, no. 3, pp. 441–451, 2003.

[6] S. Chen and Z. Wang, "Acceleration strategies in generalized belief propagation," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 41–48, 2012.

[7] S. Y. Chen, H. Tong, Z. Wang, S. Liu, M. Li, and B. Zhang, "Improved generalized belief propagation for vision processing," *Mathematical Problems in Engineering*, vol. 2011, Article ID 416963, 12 pages, 2011.

[8] C. W. Reynolds, "A distributed behavioral model," in *Proceedings of the ACM Computer Graphics (SIGGRAPH '87)*, M. C. Stone, Ed., pp. 25–34, 1987.

[9] C. W. Reynolds, *Steering Behaviors for Autonomous Characters*, Sony Computer Entertainment America, Boulevard Foster City, Calif, USA, 1999.

[10] M. Li and W. Zhao, "Visiting power laws in cyber-physical networking systems," *Mathematical Problems in Engineering*, vol. 2012, Article ID 302786, 13 pages, 2012.

[11] M. Li and W. Zhao, "Representation of a stochastic traffic bound," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1368–1372, 2010.

[12] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *Proceedings of the ACM Transactions on Graphics (ACM SIGGRAPH '02)*, pp. 473–482, July 2002.

[13] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," in *Proceedings of the ACM Transactions on Graphics (ACM SIGGRAPH '02)*, pp. 483–490, July 2002.

[14] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," in *Proceedings of the ACM Transactions on Graphics (ACM SIGGRAPH '02)*, pp. 491–500, July 2002.

[15] S. Y. Chen, H. Tong, and C. Cattani, "Markov models for image labeling," *Mathematical Problems in Engineering*, vol. 2012, Article ID 814356, 18 pages, 2012.

[16] P. S. A. Reitsma and N. S. Pollard, "Evaluating motion graphs for character navigation," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 89–98, Grenoble, France, 2004.

[17] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," *Graphical Models*, vol. 68, no. 2, pp. 158–174, 2006.

[18] M. Lau and J. J. Kuffner, "Behavior planning for character animation," in *Proceedings of the 5th Eurographics Symposium on Computer Animation (ACM SIGGRAPH '05)*, pp. 271–280, Los Angeles, Calif, USA, July 2005.

[19] S. M. Lavalle, *Planning Algorithms*, Cambridge University Press, Cambridge, Mass, USA, 2006.

[20] S. Chen, J. Zhang, Y. Li, and J. Zhang, "A hierarchical model incorporating segmented regions and pixel descriptors for video background subtraction," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 118–127, 2012.

[21] A. R. Da Silva, W. S. Lages, and L. Chaimowicz, "Improving boids algorithm in GPU using estimated self occlusion," in *Proceedings of SBGames'08: Computing Track, Computers in Entertainment (CIE)*, pp. 41–46, 2008.