Improving the Performance of Partitioning Methods for Crowd Simulations

G. Vigueras, M. Lozano, J. M. Orduña and F. Grimaldo
Departamento de Informática - Universidad de Valencia
Spain
{Guillermo.Vigueras, Juan.Orduna}@uv.es

Abstract

Simulating the realistic behavior of large crowds of autonomous agents is still a challenge for the computer graphics community. In order to handle large crowds, some scalable architectures have been proposed. Nevertheless, the effective use of distributed systems requires the use of partitioning methods that can properly assign different sets of agents to the existing distributed resources.

In this paper, we propose the improvement of the partitioning method for distributed crowd simulations by using irregular shape regions. Concretely, we propose the partition of the virtual world using convex hulls. The performance evaluation results show that the Convex Hull method outperforms the rest of the considered methods in terms of both fitness function values and execution times, regardless of the movement pattern followed by the agents. These results show that the shape of the regions in the partition can improve the performance of the partitioning method, rather than the heuristic method used.

1. Introduction

In recent years, crowd simulations have become an essential tool for many virtual environment applications in education, training, and entertainment [4, 12, 5]. These applications require both rendering visually plausible images of the virtual world and managing the behavior of high number of autonomous agents. These requirements result in a computational cost that highly increases with the numbers of agents in the system. Thus, simulating the realistic behavior of large crowds of autonomous agents is still a challenge for the computer graphics community.

In previous papers, we proposed a scalable architecture for crowd simulations that can manage large crowds of autonomous agents at interactive rates [6, 16]. For illustration purposes, Figure 1 shows a scheme of this architecture. Each computer can act either as a Client Computer (labeled in the figure as Client$_x$) that hosts a subset of agents, or as an Action Server (labeled in the figure as AS$_x$) that contains a part of the virtual world database.

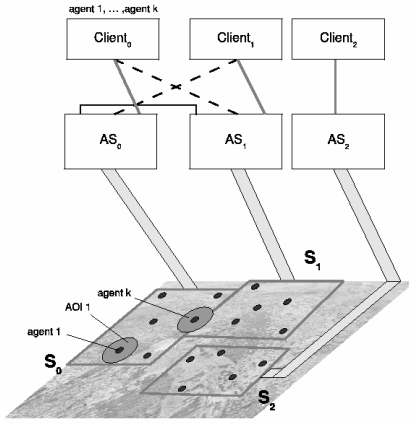

Figure 1. General scheme of the distributed crowd architecture

Nevertheless, the effective use of distributed systems requires to develop efficient partitioning algorithms that find the best distribution of the existing agents to the action servers in the system. This problem has been previously studied for PLAYSTATION3 to display 15000-fishes crowd at 60 frames per second [12]. This work incorporates spatial hashing techniques and it also distributes the load among the PS3-Cell elements. The same social forces model has been also integrated in a PC-Cluster with MPI communications among the processors, although the number of simulated agents is still low (512 agents) and the execution times are far from interactive [17]. Finally, another work describes the use of a multicomputer with 11 processors to simulate a crowd of 10.000 agents at interactive rates [11]. However, they use static agent-processor assignment, and

978-0-7695-3326-1/08 $25.00 © 2008 IEEE
DOI 10.1109/HIS.2008.31

no workload balancing is provided. Also, there are purely graphic approaches [10, 15] that are not concerned with scalability problems because they are not focused on managing the behavior of high number of autonomous agents.

Another proposal investigates several techniques for partitioning a crowded virtual environment into regions that can be managed by separate servers [14]. Region-based partitions, where each part of the distributed database contains the information of a different region of the virtual world, seem to be simplest way of partitioning crowd simulations. However, several problems arise when physically distributing the database. First, in order to maintain the consistency those agents near the borders of each region need to check their actions with the corresponding servers. This requires the exchanging of locking requests among the computers hosting the partition of the database. This constraint adds a significant overhead, and therefore it must be minimized. Additionally, the partition must be properly balanced, in order to avoid the saturation of the distributed system. Otherwise, one or more computers can reach saturation, greatly degrading the performance of the entire system [9].

In a previous work, we proposed a region-based approach for partitioning crowd simulations that improves the performance of the partitioning method by using a genetic search algorithm (GA) [7]. In this paper, we propose the improvement of the partitioning method by using irregular shape regions (convex hulls). We have compared this method with two techniques that use rectangular regions. One of them uses a heuristic search method (GA) and the other one uses an algorithmic method (R-Tree). The performance evaluation results show that the Convex Hull method outperforms the rest of the considered methods in terms of both fitness function values and execution times, regardless of the movement pattern followed by the agents. As a result, this method provides both less locking requests and better balanced partitions than the other methods. These results indicate that the shape of the regions in the partition has a major influence on the performance of the partitioning method, rather than the search method used.

The rest of the paper is organized as follows: Section 2 describes different existing partitioning methods and how they have been implemented for solving the partitioning problem. Section 3 presents the performance evaluation of the considered methods. Finally, Section 4 presents some concluding remarks and future work to be done.

2. Region-Based Partitioning Methods

The partitioning problem consists of finding a near optimal partition of regions (containing all the agents in the system) that minimizes the number of agents near the borders of the regions, and also that properly balances the number of agents in each region. In order to model this problem, we have defined the following fitness function to be minimized [7]:

$$H(P) \;=\; \omega_1 \cdot \alpha(P) + \omega_2 \cdot \beta(P), \quad \omega_1 + \omega_2 = 1 \qquad (1)$$

The first term in this equation measures the number of agents in the resulting partition P whose surroundings (Area Of Interest or AOI [13]) crosses the region boundaries. Each interaction of such agents will require the locking of more than one of the servers, and this overhead must be minimized. Concretely, $\alpha(P)$ is computed as the sum of all the agents whose AOIs intersect two or more regions of the virtual world (that is, the number of agents whose AOIs reside in more than one regions, see ag_k in Figure 1). $\beta(P)$ is computed as the standard deviation of the average number of agents that each region contains. Therefore, $\beta(P)$ measures how balanced the partition P is. Finally, ω_1 and ω_2 are weighting factors between 0 and 1 that can be tuned to change the behavior of the search as needed. Although only the heuristic method uses this function for guiding the search, for comparison purposes we have used $H(P)$ as the global fitness function for measuring the quality of the partitions provided by all the considered methods.

All the methods considered in this paper initially use the k-means algorithm to obtain the initial partition. Once the simulation starts, the partition should be adapted to the current state of the crowd every server cycle. In order to implement all the considered methods, during the simulation each server knows the location of the agents in its region and also the number of agents and the mass center of the region assigned to its neighbor servers. While the heuristic method uses a genetic algorithm (GA) guided by $H(P)$ to search a near-optimal partition of rectangular regions, the other two methods use spatial clustering techniques to provide a near-optimal partition. In the latter cases, the servers periodically assign each of their agents ag_k to the server controlling the region r_i that minimizes the following function:

$$f_{alloc}(ag_k, r_i) = \begin{array}{l} dstMC(ag_k, r_i) + \\ nAgs(r_i) * dstMC(ag_k, r_i) \end{array} \qquad (2)$$

where f_{alloc} is the allocation function, $nAgs(r_i)$ provides the number of agents in region r_i, and $dstMC(ag_k, r_i)$ corresponds to the Euclidean distance from ag_k to the center of mass of the region r_i. Since f_{alloc} should be minimized, the first term in f_{alloc} considers a spatial criterion and the second term balances the server workload. Every time a partition is updated, the corresponding state (center of mass and number of agents) is sent to the neighbor servers.

2.1. R-tree

The R-Tree is one of the most popular dynamic index structure for spatial searching [2]. We have implemented an

algorithmic partitioning method based on the optimization of the area of the enclosing rectangle in each inner node. The most interesting feature of this approach is that it is an efficient structure for managing the partitioning problem, since it let us to handle the crowd motion as insertions in the tree, where the f_{alloc} criteria can be easily introduced.

2.2. A Genetic Algorithm

Genetic Algorithms (GA) consist of a search method based on the concept of evolution by natural selection [8, 3]. GA start from an initial population, made of R *chromosomes*, that evolves following certain rules, until reaching a convergence condition that maximizes a fitness function. Each iteration of the algorithm consists of generating a new population from the existing one by recombining or even mutating chromosomes. In the GA proposed for solving this problem [7], a chromosome consists of an integer array that contains k rectangles, where k is the number of regions in the partition. Hence, a chromosome defines a partition of the virtual world in k regions. Thus, the searching problem can be faced using simple operators to combine the rectangles in order to find the best partition [7].

2.3. Convex Hull

This approach is oriented to handle partitions as the convex hull of the points that represent the agents locations in a particular region. In our implementation, the target is to reduce the area controlled by each server, in such a way that the number of locking requests is reduced. Concretely, we have implemented the Quickhull algorithm (QHull) [1] in our system. As stated above, each server periodically updates its region according to the f_{alloc} function. Once the agents have been inserted, the convex hull can be recomputed, so the center of mass and the number of agents can be also updated.

As an example, Figure 2 shows a snapshot of the different partitions provided by the considered methods during a simulation. Figure 2 a) and b) show the partitions provided by the R-Tree and the GA methods, respectively. It can be seen that both partitions use rectangular regions, although the overlapping of the regions in the partition provided by the GA method is lower than the overlapping of the regions in the partition provided by the R-Tree method. Figure 2 c) shows the partition provided by the Convex Hull method, and it can be seen that there is no significant overlapping among the regions of this partition.

3. Performance Evaluation

In this section, we present the performance evaluation of the heuristic methods described in the previous section.

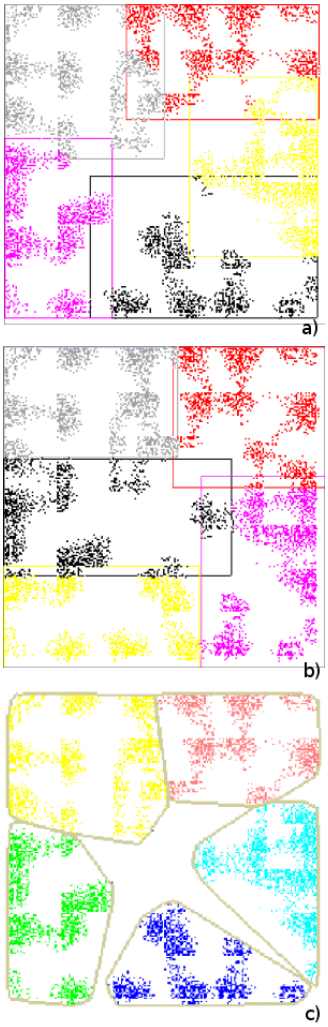

Figure 2. Snapshots of the partitions provided by a) R-Tree b) GA c) QHull methods

We propose the evaluation of the partitioning methods by simulation. Concretely, we have evaluated each method in a crowd simulation composed by 8000 autonomous agents and with five regions. We have used ω_1 and ω_2 values of 0.6 and 0.4, respectively, for all the partitioning methods, and we have applied the partitions provided by each method to the crowd simulation. Concretely, we have extracted the motion patterns of each agent off-line, and we have used this information as an input for the considered methods. The simulations have been performed on a sequential system consisting of an Intel Core Duo processor running at 1600 MHz and 2 GBytes of RAM.

We have evaluated two different crowd scenarios: an evacuation and an urban environment. The evacuation scenario consists of a structured 2-D world where there are se-

veral emergency exits. The autonomous agents must try to escape from the world as soon as possible. For this scenario we have used the same well-known movement patterns considered in our previous work [7]. In order to achieve these movement patterns, we have considered the following 2-D world configurations: *full*, where there are a lot of emergency exits uniformly distributed within the 2-D world (CCP pattern); *perimeter*, where all the emergency exits are uniformly distributed along the four borders of the virtual world (HP-Near); *up*, where there are only a few exits and they are located at the top border of the world (HP-All); and *down*, where there is only one exit located at the bottom border of the world (HP-All with a single hot-point).

The second scenario considers an urban environment where the population size remains constant during the whole simulation. In this way, the complexity of the partitioning problem does not decrease with the simulation time. This scenario contains twenty target locations randomly distributed within the virtual world. Each agent randomly selects one of these targets and approaches it. Once the target has been reached, then the agent randomly selects the next target and repeats the process, until all the targets have been reached. We have denoted this configuration as *urban*.

In order to measure the actual improvement that the different methods can provide to real systems, we have simulated the five configurations shown above. However, due to space limitations we will only show the results for the *down* and the *urban* configurations. The results for the other configurations were very similar to the ones shown here.

Figure 3 shows the fitness function values provided by each partitioning method for the *down* simulation. In this figure,the X-axis represents the simulation time in milliseconds, and the Y-axis represents the fitness function values.

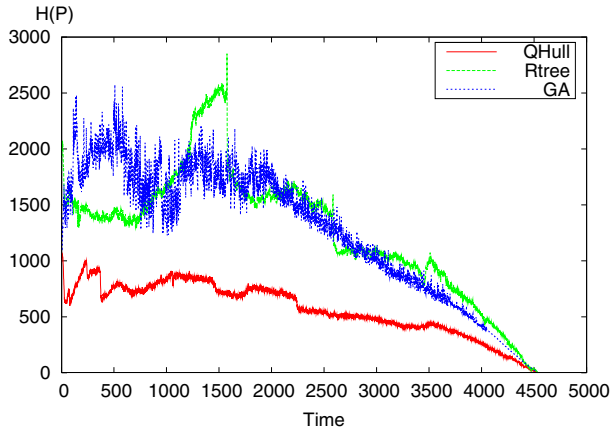

Figure 3. Fitness function values provided by the partitioning methods for the down simulation

Figure 3 shows that the QHull method provides the best fitness function values during the whole simulation. Concretely, the values provided by this method are about 50% (or less) of the values provided by the other two methods in the first two thirds of the simulation. The differences only decrease towards the end of the simulation. The reason is that as the agents exit the virtual world the population size decreases, and so does the complexity of the partitioning problem.

Figure 4 shows the execution times required by each partitioning method for the *down* simulation. This figure shows on the X-axis the simulation time (in milliseconds), and it shows on the Y-axis the execution times (in milliseconds) required by each method for computing the provided partition in each cycle of the simulation. Figure 4 shows on the one hand that the QHull method requires the shortest execution times. This method requires around one third of the execution time required by the GA method, and around one fifth of the time required by the R-Tree method. The ratio between the different execution times remains constant during the simulation. On the other hand, Figure 4 shows that the execution times required for all the methods decrease as the simulation proceeds, since the agents exit the virtual world.

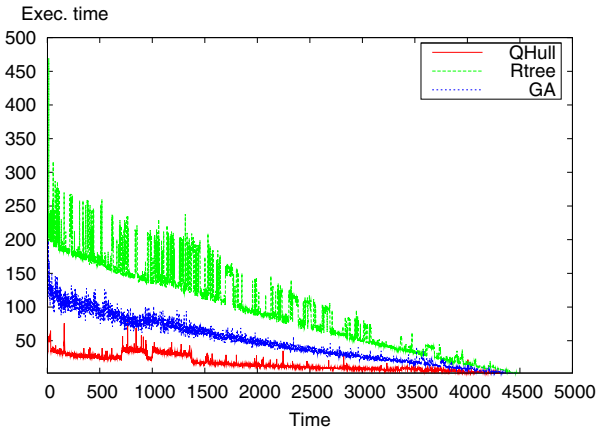

Figure 4. Execution times required by the partitioning methods for the down simulation

In order to show the performance of the partitioning methods when the population size remains constant during the whole simulation, Figure 5 shows the fitness function values provided by each partitioning method for the *urban* simulation. This figure shows that in this case all the plots have similar shapes, and after a stabilizing period (about 200 milliseconds) all the plots show a flat slope. Figure 5 clearly shows that again the QHull method provides the best fitness function values, being around 50% lower (better) than the values provided by the GA method and around one third of

the values provided by the R-Tree method.

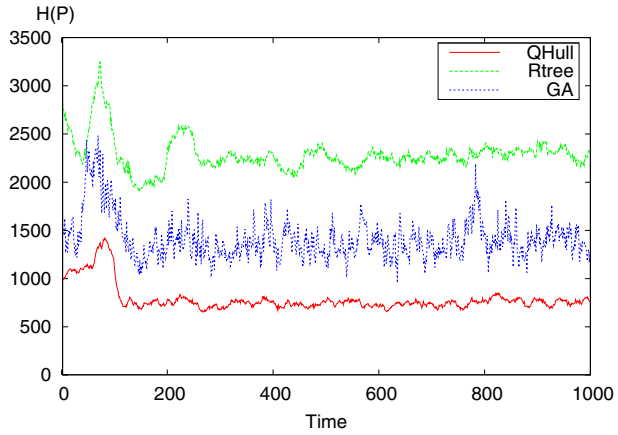

Figure 5. Fitness function values provided by the partitioning methods for the urban simulation

Figure 6 shows the execution times required by each partitioning method for the *urban* simulation. This figure also shows great differences in the execution times requires by each method. Again, the QHull method requires execution times that are around one third of the times required by the GA method and around one fifth of the R-Tree method.

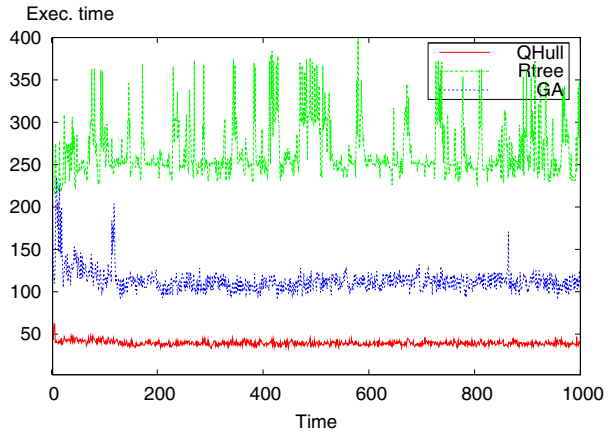

Figure 6. Execution times required by the partitioning methods for the urban simulation

These results show that the QHull method provides the best fitness function values while requiring the shortest execution times. However, the actual benefits that each method provides to the crowd simulation should be measured. Table 1 shows the performance of the partitioning methods in terms of the average number of locking requests produced in each AS cycle among the computers hosting the database. Each value shown in this table is the average value for all the AS cycles of the simulation time. Also, this table shows the average standard deviation for the average number of agents assigned to each server. (that is, how balanced the provided partitions are).

Down		
Method	Locks	Std. Deviation
GA	1133.76	1471.68
RTree	1903.52	327.39
QHull	723.19	351.49
Urban		
Method	Locks	Std. Deviation
GA	1895.97	694.31
RTree	3156.02	989.74
QHull	1022.98	439.6

Table 1. Actual performance provided by the different methods.

Table 1 shows that the behavior of each method does not depend on the movement pattern of the agents, since similar results are provided for both the urban and the down patterns. Although they are not shown here due to space limitations, these results were also similar for the other evacuation patterns. As Table 1 shows, for all the considered patterns the GA method provided an intermediate number of locking requests and the highest standard deviations (the worst balanced partitions). The R-Tree method provided the highest number of locking requests and the lowest standard deviations (the best balanced partitions). Finally, the QHull method provided the lowest number of locking requests and also the lowest standard deviation. Thus, we can conclude that this method is the most appropriate one for solving the partitioning problem in distributed crowd simulations.

4. Conclusions

In this paper, we have proposed the improvement of the partitioning method for distributed crowd simulations by using irregular shape regions (convex hulls). We have compared this method with both a heuristic method and an algorithmic method that use rectangular regions.

The performance evaluation results show that the Convex Hull method outperforms the rest of the considered methods, in terms of both fitness function values and execution times, regardless of the movement pattern followed by the agents. As a result, this method provides the best performance in terms of both locking requests and workload balancing. These results show that the shape of the regions in the partition can improve the performance of the partitioning method, rather than the search method used.

References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[2] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM.

[3] R. L. Haupt and S. E. Haupt. *Practical Genetic Algorithms*. Ed. Willey, 1997.

[4] P. A. Kruszewski. A game-based cots system for simulating intelligent 3d agents. In *BRIMS '05: Proceedings of the 2005 Behavior Representation in Modelling and Simulation Conference*, 2005.

[5] M. Lozano, P. Morillo, D. Lewis, D. Reiners, and C. Cruz-Neira. A distributed framework for scalable large-scale crowd simulation. In *Virtual Reality, Second International Conference, ICVR 2007, Held as part of HCI International 2007, Beijing, China, July 22-27*, volume 4563 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2007.

[6] M. Lozano, P. Morillo, J. M. Orduña, and V. Cavero. On the design of an efficient architercture for supporting large crowds of autonomous agents. In *Proceedings of IEEE 21th. International Conference on Advanced Information Networking and Applications (AINA'07)*, pages 716–723, May 2007.

[7] M. Lozano, J. M. Orduña, and V. Cavero. A genetic approach for distributing semantic databases of crowd simulations. In *Proceedings of 21th. International Parallel and Distributed Symposium Workshops*. IEEE Computer Society Press, March 2007.

[8] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1994.

[9] P. Morillo, J. M. Orduña, M. Fernández, and J. Duato. Improving the performance of distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):637–649, 2005.

[10] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[11] M. J. Quinn, R. A. Metoyer, and K. Hunter-Zaworski. Parallel implementation of the social forces model. In *In Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, pages 63–74, 2003.

[12] C. Reynolds. Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 113–121, New York, NY, USA, 2006. ACM.

[13] S. Singhal and M. Zyda. *Networked Virtual Environments*. ACM Press, 1999.

[14] A. Steed and R. Abou-Haidar. Partitioning crowded virtual environments. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 7–14, New York, NY, USA, 2003. ACM.

[15] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1160–1168. ACM, 2006.

[16] G. Vigueras, M. Lozano, and J. M. Orduña. A scalable architecture for crowd simulation: Implementing a parallel action server. In *Proceedings of the 37th International Conference on Parallel Processing (ICPP-08)*. IEEE Computer Society Press, 2008.

[17] B. Zhou and S. Zhou. Parallel simulation of group behaviors. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 364–370. Winter Simulation Conference, 2004.