A Comparative Analysis of Spatial Partitioning Methods for Large-scale, Real-time Crowd Simulation

Bo Li
University of Canterbury
Christchurch, New Zealand
bli62@uclive.ac.nz

Ramakrishnan Mukundan
University of Canterbury
Christchurch, New Zealand
mukundan@canterbury.ac.nz

ABSTRACT
Acceleration algorithms involving spatial partitioning methods are extensively used in crowd simulation for real-time collision avoidance. Memory and update costs become increasingly important as the crowd size becomes large. The paper presents a detailed analysis of the effectiveness of spatial subdivision data structures, specifically for large-scale crowd simulation. The results demonstrate that a regular grid data structure combined with an extended oriented bounding volume for crowd members can facilitate efficient updates necessary for real-time performance.

Keywords
Crowd simulation, crowd animation, partitioning algorithms, collision detection, subdivision data structures, bounding volumes.

1. INTRODUCTION
Crowd simulation and animation is an active area of research that finds several applications in computer graphics, analysis and design of urban environments and the development of emergency evacuation strategies [SOH11], [WXZ+11]. In recent years, real-time crowd simulation applications have gained importance in virtual training systems, such as combat operations training [QC09]. Real-time simulation of large-scale crowd movement requires effective spatial partitioning methods that can provide fast updates.

Space partitioning techniques [Sam06] are widely used for broad phase collision detection between objects and also for collision avoidance with obstacles. The more general problem of crowd detection however addresses various other aspects such as narrow-phase collision detection using intersection tests between pairs of geometrical primitives, and collision response algorithms. Several space partitioning data structures such as quadtrees [KLZ08], *k*-d trees [GCL+10] and regular grids [BQ10] are commonly used to reduce the number of comparisons between objects. Bounding interval hierarchies [WK06] are recently introduced data structures that have been found useful in applications such as ray-tracing. A detailed comparative analysis of these data structures in terms of their effectiveness and suitability for large-scale crowd simulation will be useful for the development of real-time applications. This paper presents some of the important results obtained through our research. For convenience, a single member of a crowd is referred to as either a "character" or an "agent." The motion of the crowd is assumed to be confined to a two-dimensional "ground" plane. Thus, even though characters in a crowd and obstacles may have three-dimensional representations, we need consider only a two-dimensional motion projected onto the ground for analysing problems such as collision detection, obstacle avoidance and path planning.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

The paper is organised as follows. The next section describes space partitioning data structures considered in our research: Grid, Quadtree, *k*-d tree, and Bounding Interval Hierarchy (BIH). Section 3 presents our crowd simulation model and discusses the implementation aspects of the above four data structures. Results of the comparative analysis are reported in Section 4. Section 5 summarises the main contributions of the paper and outlines future work.

2. SPATIAL PARTITIONING
In this section, four different data structures that are suitable for crowd simulation are discussed, looking

at specific advantages and drawbacks of each. Three of the data structures are widely used in broad-phase collision detection and the fourth method [WK06] is

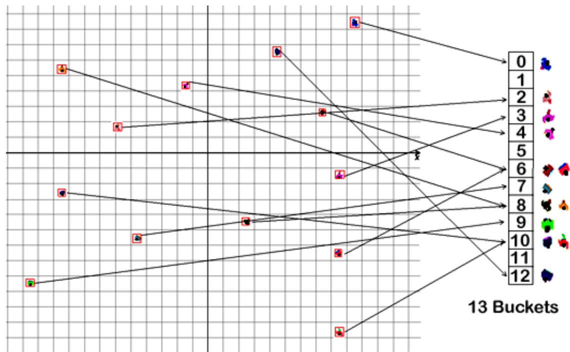

Figure 1. An example of a set of character models in a grid-based partitioning, with a hash table storing object's indices.

used primarily in ray tracing applications. For the purpose of comparison, the brute-force method which compares every pair of character models/agents for collision detection, is also considered in the experimental analysis.

2.1 Regular Grid

A uniform grid [LD08] is a very effective space subdivision scheme. It is fast for collision detection and easy to implement. It partitions the simulation space into small cubic cells with same size. The size of cells is usually defined based on the character's bounding box size, and the character itself is associated with the cell that contains its centre. It is also assumed here that all characters have the same size. Since crowd simulation models will usually consist of large scenes, a regular partitioning of the space into small cells will yield a large number of cells and correspondingly large memory requirement. One straightforward solution is to have spatial hash structures. A spatial hashing [THM+03] divides 2D or 3D space into uniform grids, and then uses a hash function to convert them into 1D hashed table. For example, a point with position $p = (x, y, z)$ is hashed into a hash table of size h by computing its cell index c as follows:

$$c = \left(\left(\left(\frac{x}{d}\right) * u\right) \oplus \left(\left(\frac{y}{d}\right) * v\right) \oplus \left(\left(\frac{z}{d}\right) * w\right)\right) \bmod h \quad (1)$$

Where u, v, w are large prime numbers and d is the cell size. If multiple points are hashed to the same hash cell, chaining is employed to resolve these hash collisions, *i.e.*, the points are stored in a linked list specific to this cell. An example is shown in Fig. 1.

An important advantage of the grid based partitioning is that it is easy to build, and the data structure need not be updated any time during the whole simulation.

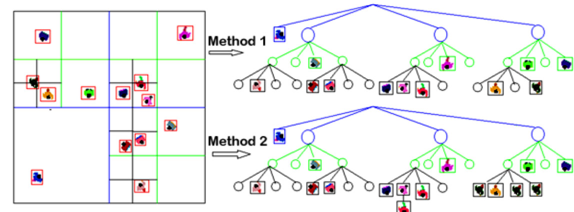

Figure 2. An example of a set of crowd members stored in a Quadtree, showing two storage methods. In the first method, objects are stored only once based on the location of their centroid, while in the second method, an object is stored in all leaf nodes that intersect the object's bounding volume.

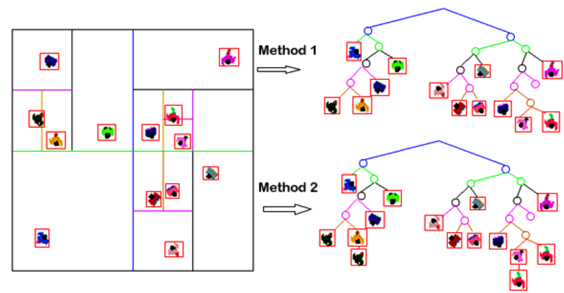

Figure 3. An example of a scene with spatial partitioning using a *k*-d tree, with two storage methods as in the case of a quadtree.

Further, an efficient hash map provides O(1) search time for finding an object.

2.2 Quadtree

Quadtree is a tree-based partitioning method, and was originally proposed in [FB74]. It was used for processing images and two-dimensional range queries at the early stages, and then found applications in several other areas such as ray tracing and collision detection. It is an axis-aligned hierarchical partitioning of a two dimensional space. Each internal node in the quadtree has exactly four children, and each node also has a finite volume associated with it. A two dimensional world generally is fully enclosed in an axis-aligned bounding square, and is subdivided into four smaller squares at each recursive step (Fig. 2).

Quadtree [PPD07], [ST05] also is a popular acceleration data structure in crowd simulation. It is easily constructed by uniformly subdividing regions containing at least one crowd member into four sub-regions. Compared to the grid, a quadtree generally uses much less memory when members of a crowd are not uniformly distributed in a region. The main

drawback of the quadtree structure is that the tree will need to be rebuilt almost every frame for a highly dynamic scene. The search time for a quadtree is also greater than that of a grid.

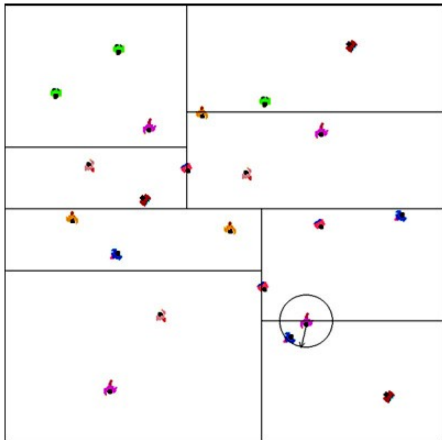

Figure 4. Nearest neighbour search using a *k*-d tree.

2.3 K-d Tree

A *k*-d tree is a binary tree, which divides a *k*-dimensional space hierarchically using a set of axis-aligned splitting planes. It uses a simple construction by dividing a non-empty space and its subspaces using median cut recursively, first using the *x*-coordinate, then the *y*-coordinate, and then again using the *x*-coordinate, and so on (Fig. 3). The depth of the tree is determined such that either the leaf nodes only contain a pre-specified maximum number of objects or the depth of tree has reached a maximum threshold. The algorithm for computing a *k*-d tree can be optimized by sorting the objects first, and then finding the median objects, and the corresponding splitting planes.

The nearest neighbour search algorithm using a *k*-d tree effectively finds an agent's neighbours. Given a *k*-d tree with *N* nodes, at least O(log *N*) inspections are needed on an average, because any nearest neighbour search requires traversal to at least one leaf of the tree. Generally, during a nearest neighbour search, only a few leaf nodes need to be inspected. Fig. 4 shows an example where only two nodes have been visited, the agent with blue background colour is the nearest neighbour for the purple agent.

2.4 Bounding Interval Hierarchy

Bounding interval hierarchies (BIHs) have been recently introduced and used for real-time ray tracing [WK06]. It is found to be faster than *k*-d trees and easy to implement. BIH is similar to bounding volume hierarchies and *k*-d trees, and has the advantages of both approaches.

BIH provides very fast construction times and efficient traversal. It uses two parallel partitioning planes for each node. For a given node, the plane perpendicular to and passing through the midpoint of the longest axis of the node's Axis Aligned Bounding Box (AABB) is first chosen as the splitting plane. Assume that this axis is in the *x*-direction, and the position of the splitting plane is x_0. The AABBs of the objects within the node's volume are then sorted along this axis. The objects whose AABBs have all

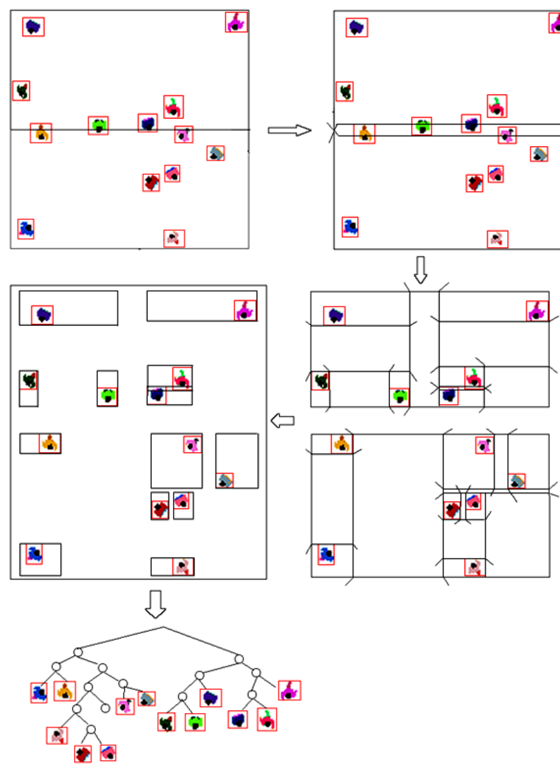

Figure 5. Partitioning of objects into left and right branches using two parallel partitioning planes.

x-coordinates less than or equal to x_0 are assigned to the left child. AABBs that are entirely on the right of the splitting plane are assigned to the right child. Objects whose AABBs intersect the splitting plane are classified as belonging to the left or right child depending on which side of the splitting plane the AABBs have maximum overlap. The left partitioning plane is then defined using the maximum value of the *x*-coordinates of the AABBs belonging to the left child, and the right plane is defined using the minimum value of the *x*-coordinates of the AABBs belonging to the right (Fig. 5). The process continues by splitting each child node along the longest axis and defining two partitioning planes along that axis. A node containing only a single object is not subdivided further.

The efficiency of the bounding interval hierarchy is due to the advantages inherited from space partitioning structures similar to a *k*-d tree. On the

other hand, bounding interval hierarchies have a fixed pre-allocatable size depending on the number of objects. Another advantage inherited from bounding volume hierarchies is that the volume elements can overlap and thus allow efficient update of the structure for dynamic scenes.

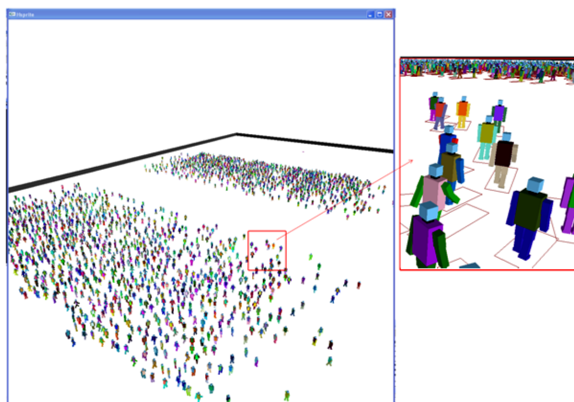

Figure 6. Crowd simulation with 2000 agents, where each agent is bounded by an EOBB of a dynamically changing stride length.

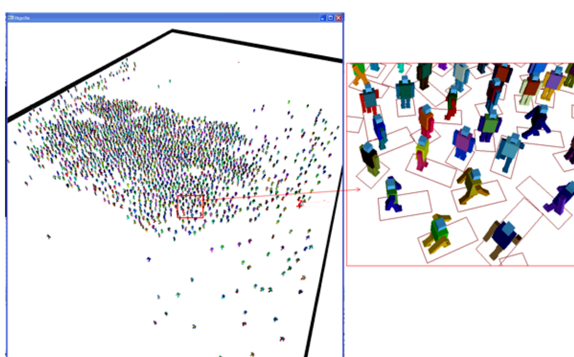

Figure 7. Crowd simulation with 2000 agents, where two groups move towards each other and converge in the middle of the scene.

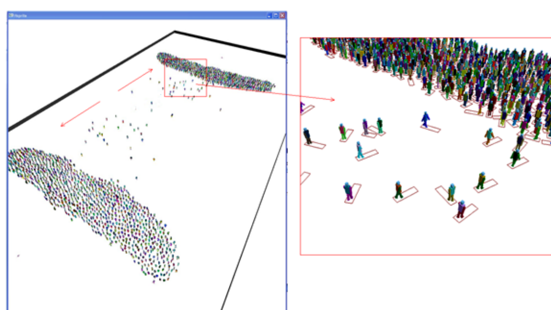

Figure 8: Crowd simulation with 2000 agents, where the whole group moves towards a common destination.

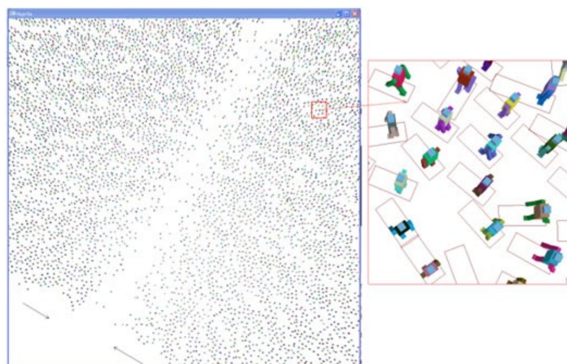

Figure 9. Crowd simulation with 10000 agents, with two groups moving in opposite directions.

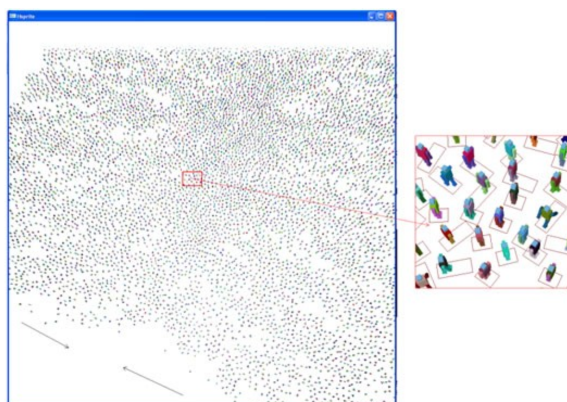

Figure 10. Crowd simulation with 10000 agents, with two large groups merging together in the middle of the scene.

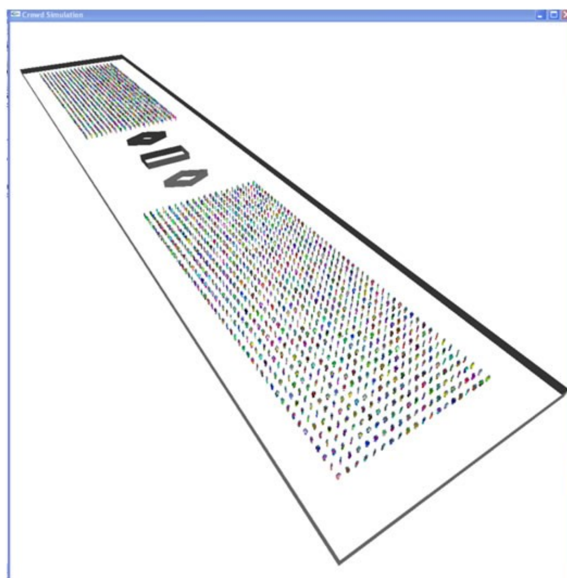

Figure 11. Crowd simulation with 2000 agents with three different obstacles located in middle of the scene.

 ISBN 978-80-86943-75-6

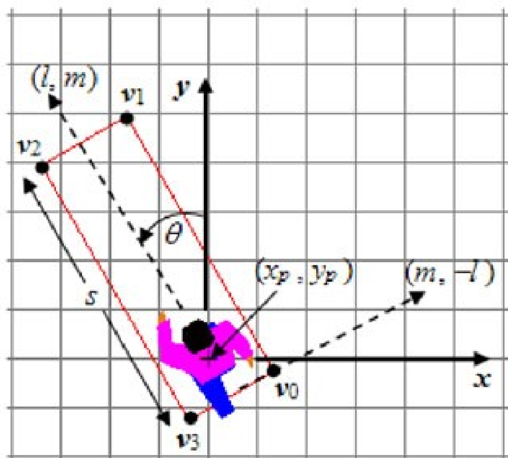

Figure 12. Definition of the extended OBB. [ML12]

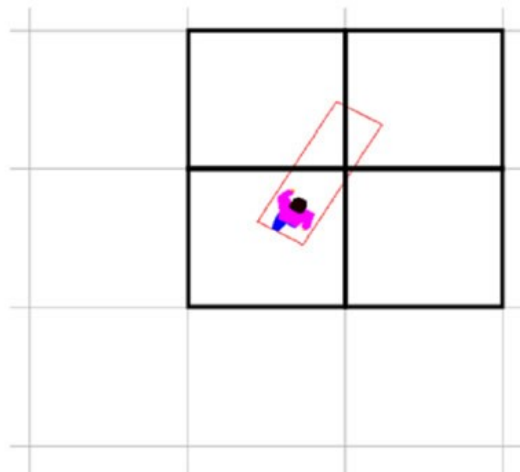

Figure 13. Spatial partitioning using a regular grid where the grid size equals the maximum stride length.

3 CROWD SIMULATION MODEL AND IMPLEMENATION

For the purpose of experimental analysis, we constructed a set of complex 3D scenes (such as Fig. 6-10), each with a large number of crowd members. A typical scenario consisting of two large groups of people moving in opposite directions are shown in Figs. 6-8, with Fig. 6 showing the start configuration of the simulation. The groups meet in the middle of the scene later in the simulation (Fig. 7) and later reach the destination (Fig. 8). In each figure, a small region is enlarged to clearly show the crowd members and their corresponding extended oriented bounding boxes (EOBB) with different stride lengths. We then increase the complexity of the simulation by adding a few obstacles into the scene (Fig. 11). Such a scene can have increasing levels of complexity based on the behaviour models and path planning algorithms used. Poorly designed algorithms can also make inefficient memory requests. We implemented all four data structures described earlier to evaluate the performance of each case with respect to increasing crowd size.

We also used a new data structure called the extended oriented bounding box (EOBB) (Fig. 12) [ML12] into our simulation system. EOBBs are convenient data structures that can be used for both bounding volume and instantaneous motion representation. Using EOBBs, broad-phase collision detection can be performed using OBB overlap tests. The stride length *s* of an EOBB can be dynamically updated based on the number and positions of character models present in the immediate neighbourhood of an object. EOBBs are also found to be useful for obstacle avoidance and computing path deviations [ML12]. EOBBs were used in all our experiments with the four data structures outlined in the previous section.

First, we implemented a uniform grid data structure, where the grid size is equal to a predefined maximum stride length. A hash map is used for storing object locations. In [EL07], the authors described a simple and fast way to implement hash functions for minimising the time taken for collision detection. Based on their research, we used a XOR hash function to generate the hash table using Eq. (1), with the values of *73856093* for u and *19349663* for v. While designing the hash table, we need to consider the trade-off between number of cells and memory usage. If we reduce the number of cells, the probability of objects being assigned to the same cell of the hash grid increases. Based on the analysis in [EL07], our hash table has a size h equal to the number of the objects in the crowd simulation.

After the hash spatial data structure is created, the grid neighbour searching method is used for finding all potential colliding agents. The position (x_p, y_p) of a character can be directly used to compute the hash table index as well as the indices of the neighbouring grids. The direction (l, m) is used to select a maximum of three neighbouring cells out of a total of 8 (Fig. 13). If the world coordinate extents of the scene space are given by (x_{min}, y_{min}), (x_{max}, y_{max}), and if the maximum stride length used for discretization is s, then the function is given by:

$$k = \left(\frac{x_p - x_{min}}{s}\right) + M \times \left(\frac{y_p - y_{min}}{s}\right) \qquad (2)$$

where $M = (x_{max} - x_{min})/s$.

The neighbouring cells are selected based on the signs of the components of the current direction vector (l, m). For example, the cell vertically above index k given by the index $k+M$ is chosen if $m \geq 0$.

 ISBN 978-80-86943-75-6

The cell diagonally above the current cell is given by $k+M+1$ is selected if both l and m are positive. Only

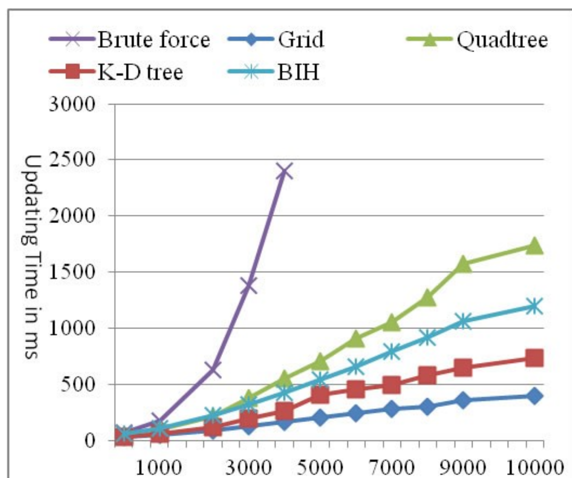

Figure 14. Updating time vs. number of agents (Grid, *k*-d tree, quadtree, and BIH)

four identified cells (including the current cell) are used for the broad-phase collision detection for each agent.

Second, we implemented a quadtree shown in Fig. 2, and the simulation scene is subdivided into four smaller squares at each recursive step. A leaf node contains only agents. Each agent is stored only once in a leaf node. The centroid of each agent is used to determine which side of the split plane the agent lies. A leaf node size is always larger than the maximum stride length of an agent. For the neighbour search, we first traverse the tree and find the leaf node which contains the agent, then we use OBB-AABB overlap test to find if the agent is fully contained by leaf node. If so, we add all agents in this leaf node to the agent's neighbour. Otherwise, we check which boundary of leaf node is intersected with EOBB of the agent, and then traverse the tree to find the leaf nodes which connect with those boundaries, finally add those agents in the leaf nodes to the agent's neighbours.

Third, a *k*-d tree is implemented into our system, as shown in Fig. 3. In a highly dynamic scene, the *k*-d tree will need to be updated almost every frame. By choosing splitting planes properly using median points we can always aim to get a nearly balanced tree. We first choose the longest axis, and use the quick sort algorithm to sort the object coordinates along this axis, and the middle point is chosen as a splitting plane. The goal of this approach is to create subgroups which contain nearly the same amount of objects. Then the recursive spatial partitioning is continued until the depth of the tree has reached a pre-specified number, or if the number of the objects in the leaf nodes is less than a given threshold. A point on the splitting plane is always stored in left node of the tree. The nearest neighbour search algorithm is used to minimise the number of comparisons in the collision detection phase.

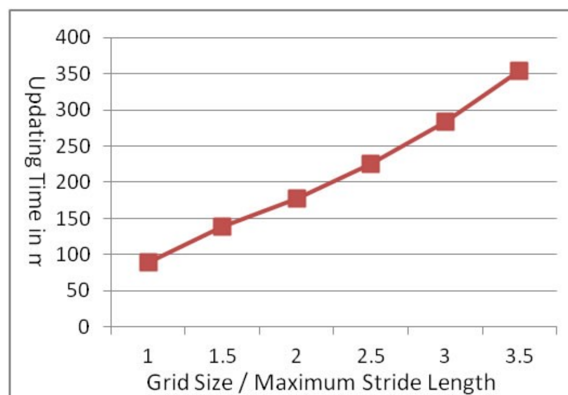

Figure 15. Updating time vs. cell size for the grid structure.

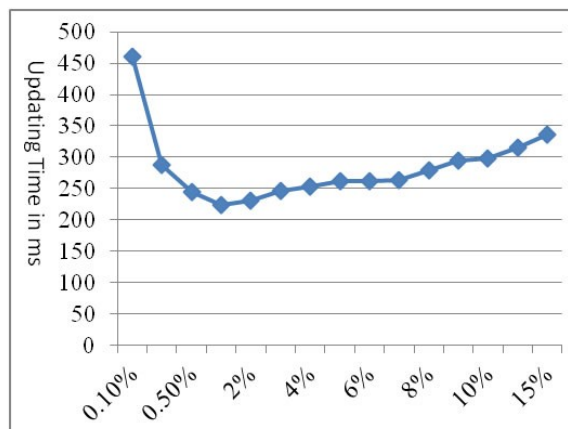

Figure 16. Updating time vs. number of agents in leaf node for a quadtree.

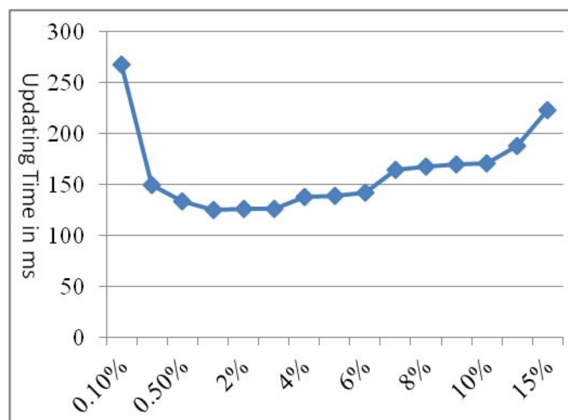

Figure 17. Updating time vs. number of agents in leaf node for a *k*-d tree.

 ISBN 978-80-86943-75-6

Finally, we implemented a Bounding Interval Hierarchy for partitioning a crowd scene. The AABB is calculated for each object, the approximate sorting [WK06] is used to sort the agents, and then we determined the two paralleled splitting planes by median-cut. The next section describes experimental results obtained using the above partitioning methods.

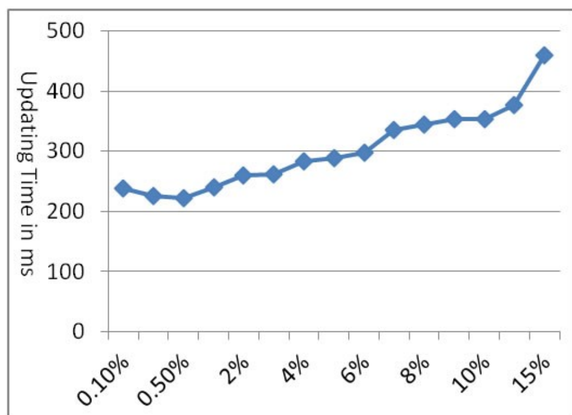

Figure 18. Updating time vs. number of agents in leaf node for a BIH.

4 RESULTS AND EVALUATION

Experimental results shown in this section are generated by simulating different scenarios by increasing crowd size and adding different types of obstacles.

First, we compared the update time for each data structure. The results are depicted in Fig. 14. The graph shows that grid structure gives the best performance. Other types of hierarchical structures required frequent updates because their partitioning algorithms depend on both the position and distribution of crowd members in the constantly changing scene. Even when the number of agents in the scene is 10000, the time taken for updates using the grid structure is less than 500ms. We also noticed that there grid structure does not provide a significant improvement over the brute-force method, when the number of agents is less than 1000.

Cell size is an important parameter in the design of the grid data structure. In our experiment, the size of cell is set up to the maximum stride length of agent first, and then we increased the size of cell, and measured the updating time. The results are shown on Fig. 15. The best performance is obtained when the cell size just fits the maximum stride length of agent.

In Figs. 16, 17, 18, we provide experimental results using quadtree, *k*-d tree, and BIH to find the variation in performance with the maximum number of objects stored in each leaf node. The results show that when the objects in leaf node is 1% of total number of objects provide the best performance for both quadtree and k-d tree. For the BIH, we can subdivide the space until only one object is in the leaf node, and we can still get a good performance, but when the number of objects contained in leaf node is 10, we got the best performance.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a fairly extensive comparative analysis of the performance of spatial subdivision structures in large-scale crowd simulation. The simulation results show that a grid data structure with extended oriented bounding boxes for character models gives the best performance when the number of crowd members is very large. Crowd simulation of 10000 agents in real-time can only be achieved by using such spatial data structures. The grid data structure has the advantage that it doesn't need to be updated, and an efficient hash map implementation can provide fast look-up. The extended oriented bounding box is also found to be very efficient in representing both geometry and instantaneous motion of a character in the crowd.

The paper has presented an overview of four commonly used spatial subdivision methods, and analysis using update time with respect to variations in crowd size, grid cell size, and the maximum number of objects in the leaf nodes of the tree structures for quadtree, *k*-d tree and the bounding interval hierarchy.

Future work in this area is directed towards combining collision avoidance with path/motion planning algorithms incorporating various types of behaviour models. Effective mechanisms for improving the performance of hash mapping for the grid structure will also be explored. A direct extension of the work presented in the paper would be the performance analysis of acceleration algorithms when crowd motion is not confined to a two dimensional plane. Such methods would then heavily rely on multi-dimensional data structures [Sam06] for minimizing comparisons.

When the crowd size increases in scale from large to massive, the performance of acceleration methods becomes crucial. Several models for the simulation and rendering of massive crowds have now been attempted on the GPU [JPZ+09], [PJZ+08], [PJZ+10]. GPU implementations of spatial structures have been successfully tried using just neighbours of agents. Structures similar to the extended oriented bounding boxes could also be explored further, and implemented on parallel architectures.

 ISBN 978-80-86943-75-6

6 REFERENCES

[BQ10] F. Bu and C. Qin. Research on the mass events based on grid-agent. Proc. of Youth Conference on Information Computing and Telecommunications, pp. 130–133, 2010.

[EL07] M. Eitz and G. Lixu. Hierarchical spatial hashing for real-time collision detection. Proc. of IEEE International Conference on Shape Modeling and Applications, pp. 61–70, 2007.

[FB74] R. A. Finkel and J. L. Bentley. Quadtrees: a data structure for retrieval on composite keys. Journal of Acta Informatica, pp. 1–9, 1974.

[GCL+10] S. J. Guy, S. Curtis, M.C. Lin, D. Manocha. PLEdestrians: a least-effort approach to crowd simulation. Proc. of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 119–128, 2010.

[JPZ+09] M. Joselli, E.B. Passos, M. Zamith et. al. A neighbourhood grid data structure for massive 3D crowd simulation on GPU, Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on. IEEE, pp. 121-131, 2009.

[KLZ08] W. L. Koh, L. Lin, and S. Zhou. Modelling and simulation of pedestrian behaviours,. Proc. of 22nd Workshop on Principles of Advanced and Distributed Simulation, pp. 32–50, 2008.

[LD08] A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. Computer Graphics Forum, Proc. of the 19th Eurographics Symposium on Rendering, pp. 1235–1244, 2008.

[ML12] R. Mukundan and B. Li. Crowd simulation: Extended oriented bounding boxes for geometry and motion representation. Proc. of the 27th Conference on Image and Vision Computing New Zealand, pp. 121–125, 2012.

[PJZ+08] E.B. Passos, M. Joselli, M. Zamith, et. al., Supermassive crowd simulation on GPU based on emergent behavior. Proc. of the VII Brazilian Symposium on Computer Games and Digital Entertainment, pp. 70-75, 2008.

[PJZ+10] E.B. Passos, M. Joselli, M. Zamith, et. al. A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU, Computers in Entertainment, Vol. 7, No. 4, Article 60, pp. 1-15, 2009.

[PPD07] S. Paris, J. Pettré, and S. Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. Proc. of Computer Graphics Forum, pp. 665–674, 2007.

[QC09] H. Qiu and L. Chen. Real-time virtual military simulation system. Proc. of 1st International Conference on Information Science and Engineering, pp. 1391–1394, 2009.

[Sam06] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann Publishers, New York, 2006.

[SOH11] S. Sharma, S. Otunba, and J. Han. Crowd simulation in emergency aircraft evacuation using virtual reality. Proceedings of the 16th International Conference on Computer Games, pp. 12–17, 2011.

[ST05] W. Shao and D. Terzopoulos. Autonomous pedestrians. Proc. of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 19–28, 2005.

[THM+03] M. Teschner, B. Heidelberger, M. Muller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable models. Proc. of the Vision, Modeling, and Visualization Conference, pp. 19–21, 2003.

[WK06] C. Wächter and A. Keller. Instant ray tracing: The bounding interval hierarchy. Proc. of the 17th Eurographics conference on Rendering Techniques, pp. 139–149, 2006.

[WXZ+11] X. Wei, M. Xiong, X. Zhang, and D. Chen. A hybrid simulation of large crowd evacuation. Proc. of 17th International Conference on Parallel and Distributed Systems, pp. 971–975, 2011.

 ISBN 978-80-86943-75-6