SCALABLE PARALLEL COLLISION DETECTION SIMULATION

Ilan Grinberg
Computer Science Department
Bar-Ilan University
Ramat-Gan
Israel
ilan_grin@hotmail.com

Yair Wiseman
Computer Science Department
Bar-Ilan University
and School of Computer Science & Engineering
The Hebrew University of Jerusalem
wiseman@cs.huji.ac.il

Abstract

Several simulations for parallel collision detection have been suggested during the last years. The algorithms usually greatly depend on the parallel infrastructure and this dependency causes in many times non-scalability performance. The dependency also harms the portability of the simulation. This paper suggests a scalable and portable parallel algorithm for collision detection simulation that fits both clusters and MPI machines.

Keywords: Motion Detection and Estimation, Scalability, Geometry Models, Parallel and Distributed Simulation.

1 Introduction

There are many simulation software tools in the civilian and military markets for many purposes [1], for example Crash Detection Simulation, Vehicle Survival Performance, Sensor Calibration Optimization, Safety Military Experiments and Simulation of Battles. All of these tools are based on three-dimensional shapes that made up of elementary polygons [2]. Such simulation systems are designed to illustrate the real world; hence, they require high accuracy. High accuracy is obtained by using ten thousands to millions of polygons [3]. Handling so many polygons obviously requires enormous computation resources.

The main computation in such simulators focuses in basic functions of computational geometry like:

- Lines intersections
- Line-Polygon intersections
- Polygon-Polygon intersections
- Collision Detection on Gantt Chart
- Projections
- Transformations
- Axis Switches.

and many more.

These functions are part of any standard graphic engine that is used as a framework for any three-dimensional simulator or any computer game [4]. Some of these computations consume many resources and an acute problem can occur if the number of the geometrical elements in a given space is too high.

Methodologies in this area raise several issues like what the optimal way to implement these functions in any computational environment is. Any special hardware like Graphics Accelerator Card, Graphic Card with Dual Processor, Multi-Processor Computer or Computer Cluster can significantly influence the implementation of these functions.

To accommodate the many requirements of the computational geometry functions (e.g. Polygon-Polygon intersections in a space where each polygon has its own velocity and acceleration) there will be a need for:

- Clever algorithms that can reduce the complexity of the function.
- Utilization of as many as possible of processors i.e. parallel or distributed computation.

In this paper, we present a parallel algorithm for Collision Detection Simulation. The suggested algorithm is based on the Locality Principle and the Load Balance Principle. The algorithm is suitable for both complex and simple geometry models with no dependency on the parallel environment and the architecture of the machines.

2 Bounding Volumes

One of the most common methods for efficiently implementing computational geometry functions is constructing a smart simulation model for each geometry shape consists of basic polygons. Figure 1 is an example for such a simulation model.

Such a simulation model for the geometry shapes enables an execution of geometry functions on the simulation model in a much more efficient manner than executing the functions on each polygon that the geometry shape consists of. This simulation model is well known as Spatial Data Structures [5].

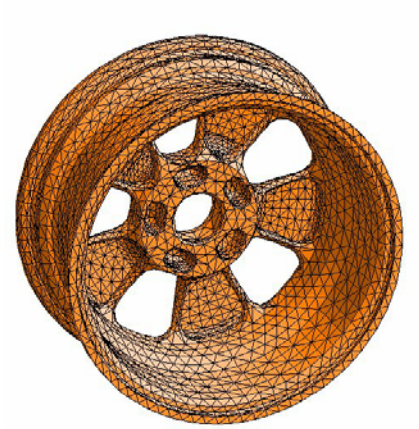

Figure 1: Simulation Model of a geometry shape consists of basic polygons.

Spatial Data Structures are used in two ways. The first way is reducing the number of intersection checks of static and dynamic objects in a given space. For n objects, there will be $\left[\frac{n}{2}\right]$ potential objects that may be intersected. This number is obviously very high and the significantly reducing in the number of intersection checks obtained by Spatial Data Structures is quite important.

The second way is reducing the number of intersection checks of pair of primitive polygons in intersection detection of two complex objects or an intersection of a primitive object and a complex object. In this scheme, the Spatial Data Structures are created in the preprocessing step and remain static, assuming the simulated objects are rigid.

Spatial Data Structures are employed for Space Partitioning [6] and Bounding Volumes [7]. Space Partitioning is a sub-partitioning of a space to convex regions called cells. Each cell maintains a list of objects that it contains. Using this structure, many pairs of objects can be easily sifted out.

Bounding Volume is a split of an object set into consistent subsets and computing for each one of the subsets tight bounding volume so when the intersections of the subsets are checked, these subsets will be straightforwardly sifted out by finding which bounding volumes are not overlapping.

We require the Bounding Volume to have several features:

- The Bounding Volume should be in a close proximity to the model in order to reduce the cases when an intersection with a Bounding Volume is detected but no other parts of the model are actually intersected.
- Detecting an overlap of two Bounding Volumes should be very fast i.e. this detection should be much faster than a detection of two overlapping models.
- The implemented Bounding Volumes should consume a small memory space. This memory consumption should be much smaller than the model itself consumes.
- The computation cost of the Bounding Volumes should be inexpensive, particularly if the application needs a frequent computation of the Bounding Volumes.

Some researches have been conducted on tactics of representing Bounding Volumes like Bounding Spheres [8], K-DOPs - Discrete orientation polytopes [9], OBB - Oriented Bounding Boxes [10], AABB - Axis Aligned Bounding Boxes [11] and Hierarchical Spherical Distance Fields [12].

We prefer the most common one, the AABB tactic. This tactic represents the bounding volume as minimum and maximum values of a geometric model over each one of the axes. In this way, a bounding volume will be created. It should be noted that the AABB representation consumes more memory space than the "Bounding Sphere" tactic; however, intersections of two bounding volumes can be calculated faster.

Constructing a bounding volume in the AABB technique is quite fast [5]. The algorithm runs through each one of the basic elements contained in the bounding volume and projects it on each of the axes. The following step is finding the minimum/maximum values for each axis.

3 Bounding Volume Hierarchies

Bounding volume hierarchies [9,13] are a tree data structure, whose leaves are constructed from the basic elements of the geometry. Each node's leaves are placed inside the bounding box it represents. Sibling nodes can be overlapped by their representing bounding volumes.

The advantages of using bounding box hierarchies are:

- Fast query for intersection testing
- Linear memory space in the number of elements constructing the geometry.

The major drawback of using this technique is the long execution time that the algorithm needs in order to construct the representing tree of the geometry and the updates that the algorithm requires when using non-rigid objects. This is the reason for the common use of bounding volume hierarchies with rigid geometries - its

representing tree is generated only once as a pre-processing step.

The test of collisions between two geometric models is done recursively for each two nodes that are taken from each of the geometries trees, starting with the roots. The intersection test handles the following cases:

1. If the representing bounding volumes of the nodes intersect, "False" will be returned.
2. If both of the nodes are leaves, the primitive elements contained in these leaves will be checked for an intersection.
3. If one of the nodes is a leaf and the other is not, the leaf will be checked for an intersection with one of the other node's children.
4. If both of the nodes are not leaves, the node having the smaller volume representing a bounding box will be checked for an intersection with one of the other node's children.

The overall collision detection cost of two geometric models, which are represented with bounding volume hierarchies can be formulated by the following equation [5]:

$$T_{total} = N_b \cdot C_b + N_p \cdot C_p$$

Where,

- T_{total} – The total time to test an intersection between the two models.
- N_b – Number of bounding volumes pairs that are tested for an intersection.
- C_b – The cost of an intersection test between bounding volume pairs.
- N_p – Number of primitive polygons pairs that are tested for an intersection
- C_p – The cost of an intersection test between pairs of primitive polygons

The parameters that are affected by the bounding volume type are N_b, N_p, and C_b. A tight-fitting bounding volume type, such as OBB, will produce a low N_b and a low N_p, but will produce a relatively high C_b, whereas AABB would have produced more tests to perform, but the value of C_b would have been lower.

4 Related Works

The potential of sequential algorithms is somehow limited and the parallelizing becomes an essential enhancement if the algorithm has to run quickly. Some works have been done to put into practice parallel collision detection. We survey the approaches of these works and explain what has led us to our approach.

In [14] the authors employ AABB to represent the geometry models that are used for collision detection. The algorithm constructs for each model a hierarchy of three levels of Bounding Volumes. The principle is not to construct a huge tree containing leaves with only one Bounding Volume of a single polygon. However, complex geometry shapes are likely to have leaves with many primitive polygons. Because of such leaves, the execution time is much larger than a full hierarchy of Bounding Volumes due to the many checks of intersections of polygon pairs.

The algorithm of [14] is dedicated for SMP machines and cannot work on computer clusters. In SMP machines the RAM is in the vicinity of all the processors; hence, the locality principle is not kept. If the algorithm is used on a computer cluster and the geometry shapes are complex, a bottleneck will be occur when the geometry shapes are loaded in the beginning of any computation unit.

The algorithm of [15] suggests a parallel version for Space Partitioning Based Collision Detection. The algorithm is scalable and keeps the locality principle by making any voxel a separate process. However, this algorithm does not employ Bounding Volumes Hierarchy. Rather, it employs Bounding Volumes in a constant size that has been set in advance. This feature will drastically harm the performance of the algorithm if the geometry shapes have non-homogenous density in the polygon prefix.

In addition, some polygons have empty Bounding Volumes that cause unequal balance on the nodes in the cluster. In order to balance the load the algorithm utilizes the parallel infrastructure. Actually, this indicates that no effort is taken to balance the load. If such a case occurs, the parallel infrastructure will be supposed to resolve the unequal balance. This tactic creates an overhead - the solution for unbalance nodes is a migration of processes from one node to another. These migrations may harm the performance of the algorithm.

The cost of constructing the voxels' data structures in [15] is quite low; hence, rigid objects that require frequent updates of the data structure can benefit this feature.

The splitting of the space into a large number of voxels enables a node that hosts many voxels to efficiently manage voxels that need the processor and voxels that need the communication line, by the operating system. Obviously, this will not be correct in the beginning of a new simulation when there is no data for any voxel and all the voxels call one node at the same time and create a bottleneck.

In [16] several versions of parallel algorithms for collision detection are presented. The algorithms keep the locality principle and keep the load balance of the nodes. The algorithms are aimed at collision detections of animations (roughly some dozens of frames per seconds) for simple geometry shapes (less than 4000 primitive shapes). The

algorithms are also aimed at just SMP architectures with shared memory.

The load balancing in [16] is static and is done in the beginning before the intersection check step. In addition, the algorithm assumes that the processors are homogenous. Static load balancing reduces the communication overhead; hence is more suitable for real-time simulation. The authors also suggest several techniques to reduce this overhead by using a common queue for several processors.

There are also works that are only aimed at grid environments like [17]. This work suggests a simulation that cannot work properly on SMP machines.

In this paper, we suggest a dynamic load balancing. This may have a slight higher overhead, but it will be able to handle heterogeneous processors and will be able to manage better computer clusters.

5 Scalable Collision Detection

The new parallel simulation that is suggested in this paper includes several advantages over the known parallel collision detection simulations. The main idea of the suggested simulation is keeping the scalability principle while not abandoning the locality principle and the load balancing of the system.

We can use one of the known algorithms for Bounding Volumes hierarchy for checks of an intersection of two models or a collision. Let us define the smallest "work unit" as one operation (like a collision detection) on a complex geometry model or one operation between two complex geometry models. Indeed, a finer split into smaller unit could have been done like the author of [14] suggest; however, the cost of execution of one "work unit" that we suggest will be still very small, even if the geometry model is very complex. Experiments show that a splitting of geometry models into too many smallest units can produce too much overhead.

Let us define "processing unit" as one process that gets some parts of the collision detection procedure and returns the results to the master process. Any process in the parallel system can migrate from one processor to another processor in the same SMP or migrate from one node to another node in the same cluster, if this is the policy of the parallel infrastructure.

The algorithm uses the Vector Space technique [18] to find similarity of scenarios ("work units") and machines ("processing units") in a similar way of queries in document sets in the Information Retrieval field.

Let us assume that we have two geometry models consist of basic polygons, n different scenarios where the models are placed in various places and various orientations and there is a collision in each scenario.

The work is defined as finding $\sum_{1}^{n} k_i$ intersection points of two objects in the n different scenarios where k_i is the number of the intersection points in scenario i. In such a case, the finding of one single collision will be denoted as one "work unit".

5.1 The Simulation Algorithm

Let us denote np as the maximal processors in our machine.

- Create np children that will be the "processing units".
- Create a queue of "processing units" in an arbitrary order.
- Construct the Bounding Volumes hierarchy of the two geometry models by one of the known models that have been cited above. The data structure can be saved along with the geometry information so there will be no need to reconstruct the hierarchy many times. The Bounding Volumes hierarchy trees represent the geometry models and any leaf in any tree contains one basic polygon. The indices are put in nodes of no more than level d in each tree from left to right as can be seen in Figure 2.
- Create a list of scenarios containing for each scenario, the scenario index and the Bounding Volume vector. i. e. for each scenario ("work unit"), the Bounding Volumes (the black nodes in Figure 2) that are a part of the current check will be put in the list.
- Let us denote the Bounding Volume vector as BB. The value of a BBi that is not intersected in the given scenario will be 0. The value of a BBj that is intersected in the given scenario will be the number of the primitive polygons that the Bounding Volume vector bounds. An example for such a data structure can be seen in Table 1 - The upper table is the Bounding Volume vectors that are intersected in given scenarios for the geometry model that is depicted in Figure 2. The lower table depicts Bounding Volume vectors for the same scenarios but for the geometry colliding with the first model.

scenario	BB1	BB2	BB3	BB4	BB5	BB6
1	0	0	1	0	1	4
2	0	11	1	1	1	0
3	1	11	1	1	0	4
4	0	0	1	0	0	0
5	0	11	1	1	1	0
6	…	…	…	…	…	…

scenario	BB1	BB2	BB3	BB4
1	2	0	14	6
2	2	1	14	6
3	0	0	14	6
4	0	1	0	0
5	0	1	0	6
6	…	…	…	…

Table 1. Example of a "work unit" list

- Create a list of "processing units". Each "processing unit" will contain a vector in the same length as the vectors in the "work unit" list. In the beginning, these vectors are zeroed.

- Allocate q scenarios for each "processing unit" in this way:
  - For each "work unit" in the processing queue, the q free scenarios that are most similar to the "work unit" vector will be selected. The most similar scenarios can be chosen by the well-known VS tactic [19] i.e. a scalar multiplication of the scenario vector and the "processing unit" vector.
  - These q scenarios will be allocated for these q "processing units" and will be removed from the "work unit" list.
  - The "processing unit" vector will be update by a switch of 0 to 1 for any Bounding Volume that was added by the new q scenarios.

- Any "processing unit" finds the collision points of the two geometry models for any scenario that was allocated for this specific "processing unit". If the "processing unit" needs more information on the primitive polygons and it does not have the information, the "processing unit" will call the parent process and will get this information from it. The "processing unit" will cache this information for a possible future use.

- When a "processing unit" finishes its jobs, the "processing unit" will call the parent process and will return the results about the intersections that were found in each scenario. The parent process will add the "processing unit" to the free "processing unit" queue.

- This procedure will be repeated until the "work unit" list is empty.

Figure 2 depicts an example of a node indexing getting to level 4 in the tree. Each of the filled squares leaves represent one single polygon. The white nodes represent internal nodes whereas the black nodes are the nodes that will be indexed and will be put in the geometry vector. The number within the node indicate the number of the polygons within the volume that the node bounds. The number beside the node is the index of the node in the vector.

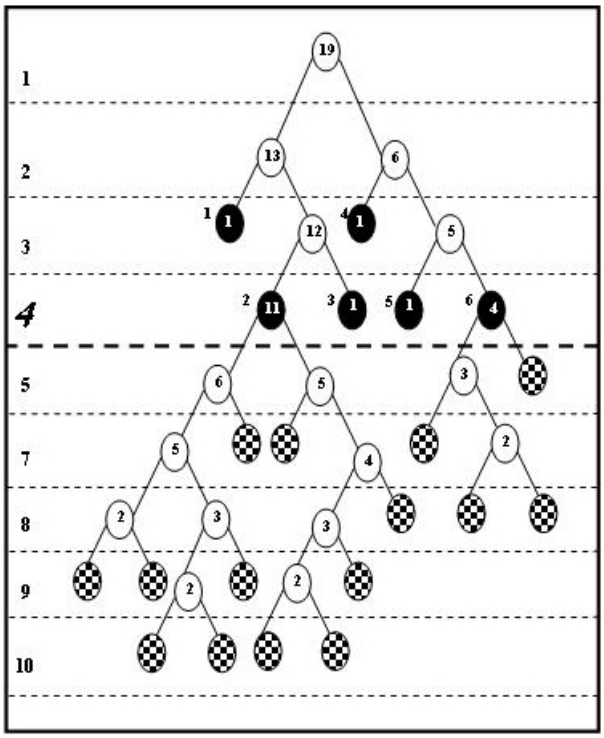

Figure 2. Example of a node indexing

5.2 Simulation Analysis

It can be seen that the "work unit" splitting mechanism enables the simulation to keep the locality principle, because the splitting of a job that contains information on parts of the geometry model is similar to the information that the "processing unit" has worked on it. In this way, the overhead of transferring the geometry models to all the machines in the cluster is prevented. If the geometry models are very complex (This is very common in many simulation tools) and the communication line speed is the common 1Gb/sec, the execution time can be improved significantly.

The simulation keeps the load balancing in the "processing units" by managing a dynamic queue and

allocation of a small work portion in each iteration for each "processing unit". In this way, an optimal computation time will be obtained even if the simulation is executed on a complex parallel infrastructure.

The simulation allocates the needed memory for each work portion in each "processing unit" and in that way the simulation saves unnecessary memory allocations in other machines in the cluster.

The simulation is generic and it can be independently implemented on any Operating System, Middleware, Hardware or Framework. The simulation is fully portable and can be used in any environment.

These is no harm for the simulation performance in any geometry models. The simulation can handle flawlessly geometry of different sizes or shapes.

6 Conclusions

Given complex geometry models, the simulation can detect an intersection in an efficient execution time. The suggested simulation is another level to the parallel infrastructure and can be easily installed on any parallel architecture. The suggested simulation does not require special resources or extensive needs; hence, many parallel machines can easily adapt it. Moreover, there is no obstruction to implement the suggested simulation on either clusters or SMP machines and the suggested simulation overcomes the initial overhead stemmed from the test of a parallel collision between complicated geometries on a computer cluster.

7 References

[1] P. Jiménez, F. Thomas, and C. Torras, "3d collision detection: A survey", *Computers and Graphics*, Vol. 25(2), pp. 269-285, 2001.

[2] S. Brown, S. Attaway, S. Plimpton, and B. Hendrickson, "Parallel strategies for crash and impact simulations" *Computer Methods in Applied Mechanics and Engineering*, Vol. 184, pp. 375-390, 2000.

[3] B. Curless and M. Levoy. "A volumetric method for building complex models from range images", In *Proceedings of ACM Siggraph '96*, pp. 303-312, 1996.

[4] Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M., "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments", In *Proceedings of the 1995 Symposium on interactive 3D Graphics* (Monterey, California, United States, April 09 - 12, 1995). SI3D '95. ACM Press, New York, NY, pp. 189-end, 1995.

[5] G. van der Bergen . "Collision Detection in interactive 3D Environments" Spatial Data Structures, pp. 171-217, 2004.

[6] T. M. Ghanem, R. Shah, M. F. Mokbel, W. G. Aref, and J. S. Vitter. ``Bulk Operations for Space-Partitioning Trees," *Proceedings of the 20th Annual IEEE International Conference on Data Engineering (ICDE '04)*, Boston, March-April 2004.

[7] B. Trumbore, *"Rectangular Bounding Volumes for Popular Primitives",* in Graphics Gems III, edited by D. Kirk, Academic Press, pp. 295-300, 1992.

[8] D.L. James, D.K. Pai, BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models, ACM Transactions on Graphics (TOG), Vol. 23(3), pp. 391-396, 2004.

[9] James T. Klosowski, Martin Held, Joseph S.B. Mitchell, Henry Sowizral, Karel Zikan, "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs", *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21-36, Jan-Mar, 1998.

[10] Gottschalk, S. A. "*Collision Queries Using Oriented Bounding Boxes*". Doctoral Thesis University of North Carolina at Chapel Hill, 2000.

[11] G. van den Bergen, Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools* Vol. 2(4), pp. 1-13, Jan. 1998.

[12] C. Funfzig, T. Ullrich, D.W. Fellner, "Hierarchical spherical distance fields for collision detection", IEEE Computer Graphics and Applications, Vol. 26(1), pp. 64-74, Jan.-Feb. 2006.

[13] Tan T., Chong, K. and Low K., "Computing Bounding Volume Hierarchies Using Model Simplification", In *Proceedings of the 1999 Symposium on interactive 3D Graphics*, Atlanta, Georgia, USA, April 26 - 29, pp.63-69, 1999.

[14] M. Figueiredo and T. Fernando. "An Efficient Parallel Collision Detection Algorithm for Virtual Prototype Environments". ICPADS'04. 2004.

[15] O. Lawlor and L. Kale. A Voxel-Based Parallel Collision Detection Algorithm. Proceedings of the 16th international conference on Supercomputing. 2002.

[16] U. Assarsson and P. Stenstr. A Case Study of Load Distribution in Parallel View Frustum Culling and Collision Detection. Lecture Notes in Computer Science Vol. 2150, pp. 663 - end, 2001.

[17] O. Lawlor. "A grid-based parallel collision detection algorithm", Master's thesis, University of Illinois at Urbana-Champaign, March 2001.

[18] Salton, G., Wong, A., and Yang, C. S.. A vector space model for automatic indexing. *Commun. ACM* vol. 18(11), pp. 613-620, Nov. 1975.

[19] G. van der Bergen, "Collision Detection in interactive 3D Environments", Support Mapping, pp. 130-139, 2004.