EG UK Theory and Practice of Computer Graphics (2006)
M. McDerby, L. Lever (Editors)

Collision Detection for Deformable Objects using Octrees

[1]F. A. Madera, [2]A. M. Day and [3]S. D. Laycock

University of East Anglia, School of Computing Sciences
Norwich NR4 7TJ, UK

[1]f.madera@uea.ac.uk, [2]amd@cmp.uea.ac.uk, [3]sdl@cmp.uea.ac.uk

Abstract

We present an algorithm for collision detection between multiple deformable objects translating in a large environment. We use Spatial Partitioning to subdivide the scene and a Bounding Volume Hierarchy to decompose the objects, using octrees in both cases. The algorithm is divided in two parts, the Broad and Narrow Phases, with objects that can be rigid or deformable. In the Broad Phase, an octree is used to partition the scene and cull away the object's Bounding Volumes that are distant. In the Narrow Phase, a hierarchical decomposition of Axis Aligned Bounding Boxes or spheres is employed to reduce the number of primitives in the pairwise comparisons. In summary this work is a general-purpose collision detection technique for performing real time collision detection of deformable bodies in interactive 3D applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

To deal with objects in a dynamic environment, it is necessary to define the type of interactions that address the simulation. Common applications that require collision detection are computer games, surgical simulation, cloth modelling, motion planning, and CAD. These applications often require both rigid and deformable objects.

Interaction among components in computer graphics programs requires efficient and robust collision detection algorithms that enable the program to reach its objectives in an easy and flexible way. In dynamic environments for animating objects, the movements of the objects make the computation expensive due to the fact that the algorithm must check the collision process at every time step. Efficient and fast algorithms are needed in real time systems, such as surgical training and virtual sculpting, where users interactively modify deformable objects (DO).

Most of the current collision detection algorithms have focused on a particular approach to get the exact requirements. The algorithm proposed here is made as general as possible, without assuming a specific physical simulation or interaction among components. The main tool is the octree, a spatial data structure used to subdivide the scene in the Broad Phase and to decompose the object in the precomputation stage to be used in the following Narrow Phase.

The program runs with n rigid or deformable objects that are able to move randomly, bounded by Axis Aligned Bounding Boxes (AABB) or spheres, in a 3D scene. The algorithm has two phases, the Broad Phase that uses an octree to partition the scene. Its function is to cull away objects until a pair is close together, then determine the collision between the Bounding Volumes (BV). The Narrow Phase starts when two or more of the BVs collide, making comparisons between the octrees that represent an object's hierarchical decomposition.

Distance computation and interaction among BVs are not made until the octree is subdivided in small regions which means that objects are in close proximity. Our application is similar to Smith et. al. [SKTK95] because we deal with multiple moving objects in a 3D environment using bounding box and spatial subdivision techniques. However [SKTK95] does not consider deformable objects as we do. The remainder of the paper is organised as follows: previous work is described in the next section, in section 3

© The Eurographics Association 2006.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

an overview of the algorithm is presented. Then follows a description of the three main modules of the program, the Physical Simulation of motion, the Broad Phase, and the Narrow Phase. Section 7 describes some experiments used to test our algorithm, and finally some possible future work and conclusions are described.

2. Previous Work

Surveys about collision detection are found in [LG98, TKH*05, JTT01], where it is stated that the most common methods used in collision detection for DO are Bounding Volume Hierarchies (BVH), Stochastic Methods, Distance Fields, Spatial Subdivision and Image-Space Techniques.

In this work we focus on Spatial Partitioning and Bounding Volume methods. Spatial Partitioning is used to divide the objects or the scene into small parts, or cells to simplify future computation among all the objects involved in the interaction. The idea is to have more control over the location of the object's primitives and consider the ones valid for collision. Ganovelli [GDO00] used a 'BucketTree' based on octrees to partition the scene and bounding boxes when dealing with objects. Another approach is shown in [GLGT98] in which the cells are decomposed in regular grids, and Teschner et. al. [THM*03] used a hash function to handle regions in a regular spatial grid.

Recently, Bounding Volumes have been used to approximate the surface of the objects and make the computations less expensive using these basic primitives. For rigid bodies, binary trees are used, and for DO it has been shown that 4-ary trees or 8-ary trees give better performance [LAM01]. Common Bounding Volumes are spheres [Hub95, BO04], Axis Aligned Bounding Boxes [vdB97], Oriented Bounding Boxes [GLM96], K-Discrete Oriented Polytopes [Klo98] and Convex Hulls [PLM95].

James and Pai [JP04] worked with spheres to wrap the objects being dealt with which were reduced deformable models and Alexa et. al. [KZ05] focused on skeletally deformable models. Also Kimmerle et. al. [KNF04] worked with stochastic collision detection and a Bounding Volume Hierarchy (BVH).

Hardware has been very useful to increase the processing speed, GPUs are valuable tools to work with image space techniques [HTG03, HTG04] and other hardware methods have also been used [GRL03, SOM04].

3. The Algorithm

Initially we have n objects $\{O_1, \ldots, O_n\}$ placed in an environment that is divided using an octree, which recursively partitions the regions into octants. Object i has V_i vertices, E_i edges, and F_i faces. The octree holds the object's BV.

The algorithm handles solid models that deform while in translation. We decided to use AABBs and spheres because they are cheap to store and fast for intersection tests. Spatial partitioning recursively partitions the scene containing the objects, whereas a BVH is based on a recursive partitioning of an object. A similar procedure is applied in the Narrow Phase to check BV collisions using octrees. Also, the motivation to explore the behaviour of DO in motion around the scene and their interactions as detected by our algorithm made us decide to combine the space partition and object decomposition methods.

In Figure 1 the main modules of the program are shown. They consist of the Pre-computation stage, Physical Simulation, Broad Phase, and Narrow Phase, and are described in the next sections. We do not assume a particular simulation method and we do not consider the contact determination phase for accurate approaches.

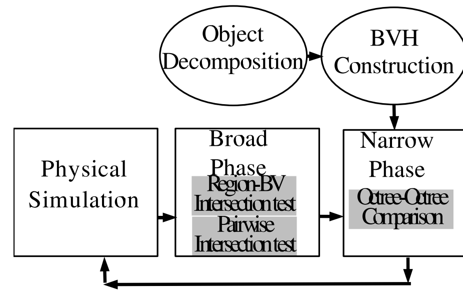

Figure 1: *Modules of the Collision Detection Program.*

4. The motion

To make the objects deformable, we apply a deforming process using the mass spring model. The physical body is represented by a set of mass-points connected by springs exerting forces on neighbouring points as a mass is displaced from its rest position. The physical simulation module involves two sub-modules, to apply spring forces and to move masses using an integration scheme. Meshes are considered as mass springs to which forces are applied. To prevent excessive deformations the area of each triangle is preserved. The spring forces are linear and the equation of motion used for this process is:

$$m\ddot{x} + k_d\dot{x} + f_s = f_{ext}, \tag{1}$$

where the coordinates x are functions of time, $\dot{x}$ and $\ddot{x}$ are their first and second time derivatives, m is the mass matrix and k_d is the damping factor. In the spring force, we handle some parameters that can be modified; spring length, spring constant force, air friction. Hooke's law with the damping term is given by

$$f_s = -(k_s(|d| - s) + k_d\dot{d} * \frac{d}{|d|})\frac{d}{|d|}, \tag{2}$$

© The Eurographics Association 2006.

where d is the elongation distance, $\dot{d}$ its first derivative, and s the original length of the spring. The force on the mass is given by $F = -kx$, where F is the force, k_s the spring constant and $|d| - s$ the length of the spring.

Objects are deformed in an arbitrary way but the shape of the surface is constrained within pre-defined limits. The BV should be updated every time frame to catch the exact size of the object. Normally, an elastic object is elongated and its shape restored but this does not happen in other objects, such as plastic.

Since $F = ma$, where F is the force, m the mass, and a the acceleration, then $a = -\frac{k}{m}x$. Now, given the acceleration, initial velocity and position, we can find the position at any later time, integrating the acceleration directly [VB04, ES04].

In the motion simulation, two integration schemes were used, the Euler Backward integration, and the Verlet integration suggested by Teschner et. al. [THMG04]. We observed more stable and realistic deformations with the Verlet scheme. Velocity is not constant, due the acceleration applied in the springs, as well as the random velocity translation applied in objects.

Initially objects are just deformed, preserving their position in the environment, so that no collisions occur. We then apply a force to get interaction. We create a random vector for each object to determine the direction, this is made in the integration scheme after deformation. Therefore, for each mass, we apply the random velocity vector, maintaining the borders of the environment. For each mass, $v_i = v_i + v_d$, where v_d is the random vector to translate the object.

Initially objects start moving in any direction and deforming randomly; users can introduce their own physical simulation at this time before the collision process starts.

5. The Broad Phase

This module begins with the creation of the octree for the global scene. The initial region is a cube that represents the root node in a tree and can be expanded in the next level with eight children. Subroutines are divided in two types, for the scene and for the objects, having sections for creating and removing nodes. In the Broad Phase, the brute force method takes $O(n^2)$, and the sweep and prune algorithm [Bar92] requires pre-sorting along each coordinate axis, indicating that the computational complexity is $O(nlogn)$, where n is the number of BVs considered. For a scene S, we have a region in the octree $R_h^k(S)$ in level k, where h is the number of the node ordered top to bottom, left to right, being the root of the hierarchy $R_0^0(S)$. When expanding, eight nodes are created, from $R_1^1(S)$ to $R_8^1(S)$ in level 1, that are the root's children. If $R_1^1(S)$ is expanded, then nodes from $R_9^2(S)$ to $R_{16}^2(S)$ are created, and so on. Two basic operations are considered, expansion/reduction and work for a group of eight nodes, in such a way that we will have a balanced tree in levels 0 and 1, but not necessarily in the other levels.

A region can be seen as an AABB of eight vertices, six faces, and twelve edges, being $R_h^k(S).v_j$ the vertex j. For an Object O_i, we have an octree defined in the same way as above, and it can be AABB, $A_h^k(O_i).v_j$, or sphere, $S_h^k(O_i).v_j$.

The main octree works by partitioning the scene into eight regular regions. If two or more objects are inside a region, then this region is partitioned into eight regular parts as well (its children). The same object can be placed in one, two or four regions at the same time, so these regions need to be considered for future partitions. The extra problem with deformable objects is that we need to check all vertices in order to refit the boundary. The general algorithm is shown in Figure 2.

$Broad - Phase\ (\ O_1, O_2, ..., O_n\ )$
1. $\forall R_h^k(S)$ that is a leaf
2. if $A_p^0(O_i)$ inside in more than one region
3. if $k = level - allowed$ then $check - collision - in - pairs()$
4. else expand node k

Figure 2: *The Broad Phase pseudocode.*

The aim is to cull away objects that are not in close proximity, i.e. that are not placed in the same region of the tree, otherwise, the region is divided into eight new children. We exploit the fact that it prunes out unnecessary collision tests by localizing potentially colliding regions. In line 1, we visit all the nodes of S that are leaves. Obviously, this is from level 1 and forwards. In line two, we compute the number of objects in each region. Nodes in S are bigger than nodes in objects, O_i can be totally inside node h, or partially, meaning that its other parts are occupying other nodes h.

In the case that the number is greater than or equal to the value of $level - allowed$, line 3 asks for the level of the node in S. This value refers to the depth of the tree in the BVH, for instance, $level - allowed = 2$ when there are 64 leaves, 4 when there are 512 leaves, and so on. Clearly regions in S should be bigger than those in the BVs in order to contain them, otherwise, we would need to check if they overlap, the process made in $check - collision - in - pairs()$.

5.1. The region-BV intersection test

To check that objects are inside a region $R_h^k(S)$, we should compute the overlapping test between the region and the object's BV, that can be a AABB or a sphere. In the case of AABB, the problem is an intersection between two AABBs, that is the minimum and maximum axis interval; for x-axis, $R_h^k(S).v_0$ and $R_h^k(S).v_1$; for y-axis, $R_h^k(S).v_0$ and $R_h^k(S).v_4$; and for the z-axis, $R_h^k(S).v_0$ and $R_h^k(S).v_3$.

In the case of the AABBs objects, the process involves

© The Eurographics Association 2006.

updating the size due to the change of the position of a vertex at every step, so it can be expanded or reduced in any axis. To do that, we need to check in every time step the vertices that have been modified, to get the minimum and maximum values of the axis in order to refit the root's octree. This takes $O(V_i)$ for object i.

Alternatively, we can use spheres as a bounding volume. One way to construct the volume, is to consider the AABB values previously obtained. Let $S^k_p(O_i)$ be the sphere bounding object i in level k, labelled p, its center is computed getting the minimum values of the AABB:

$$S^0_1(O_i).c = (xmin+\frac{L}{2}, ymin+\frac{L}{2}, zmin+\frac{L}{2}), \quad (3)$$

where L is the major length among the three axes of its corresponding AABB, and the root's level is k=0. The radius is

$$S^0_1(O_i).r = \frac{L}{2}. \quad (4)$$

These values are compatible with the AABB, but the sphere is too large for the object. One remedy is as follows:

$$S^0_1(O_i).c = \frac{xmin+xmax}{2}, \frac{ymin+ymax}{2}, \frac{zmin+zmax}{2}. \quad (5)$$

The radius is computed by comparing the distances among all the vertices to this centre. The disadvantage of this scheme is that we need to update the radius every time step, reading all the vertices of the object, but it is more accurate, so that we have chosen to do it. We also considered working with a global octree of spheres for the scene, but there is less accuracy with them so we decided to keep with the cube regions.

As seen from the case of the AABBs we should compute if the sphere is inside the regions of the global octree. A method suggested by Arvo [Arv90] finds the point on the AABB that is closest to the center of the sphere; if its squared distance is less than the sphere's squared radius, they intersect; otherwise, they do not. We propose another way of doing this, taking in mind that it is easier to compute if a point, the centre of the sphere, is in the box. If so, we have a mistake because part of the sphere would be inside the box. To correct this, we increase the size of the cube, in each axis by the length of the radius for maximum values and decrease the length of the radius for the minimum values. Hereafter we just compute if the centre is inside this new box. Figure 3 shows a snapshot of the octree working in the Broad Phase using spheres as a BV.

5.2. The pairwises intersection test

In the $check-collisions-in-pairs()$ routine, we store the pairwise parts that are colliding, this is the output in the Broad Phase and the input for the Narrow Phase. A data structure is used to store the new pair while in collision, it can be released when the pair is separated again. This pair will be useful in the Narrow Phase for checking the collisions among primitives.

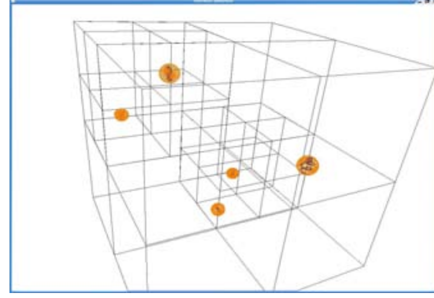

Figure 3: *The Octree in the Broad Phase with Spheres as a Bounding Volume.*

In the AABB-AABB overlapping test, the process is the same as the region-AABB intersection. For spheres the intersection test is to compute the distance of the two centers and compare it with the sum of both radii.

Comparing the actual costs of sphere test and AABB test, using squared distance spheres requires 4 multiplications, 6 additions, and 1 comparison, where AABB are handled with only 12 comparisons and 3 logical tests. Thus, spheres are generally more expensive when considering just a single pairwise test.

Before finishing this section, we need to consider the translation movement in every time step, so that having the new AABB size, we add a translation of d, taken from the velocity and time employed $(v = d/t)$. This is a quick process that is applied to each vertex of the AABB, or the centre in the case of the sphere. The last process to consider is the border of the scene, trying to keep objects inside the global region, this is just an overlap comparison with the global AABB. In the case where the object reaches the threshold of the global cube, we apply a change of direction in velocity to continue the translation movement.

We take advantage of the physical simulation process to add these routines rather than use more loops for updating the AABBs. In the case of the translation motion we follow the route defined by the direction of the object, and add this displacement every time step.

6. The Narrow Phase

With DO, the problem is to constantly update the BV, so we need to find out a method to reduce this process. We tried to handle the neighbours of a vertex in a region, avoiding a check of all vertices in the objects, but this is a problem because the changeable motion can make parts of the surface that are distant in one time step be close in the next time

© The Eurographics Association 2006.

step. Another approach is to use the global octree; we tried to use the scene's octree detecting the regions that intersect both objects, but it requires a check of all the vertices of the objects, which is too expensive.

Our third trial was to work with the common region (CR) between the two BVs. Having this CR, it is important to know its place in each BV, and we would have three bounding boxes to deal with, trying to get their intersection and later to check for the vertices involved. This process also requires more computation.

Finally, we decided to work with a hierarchy because the number of vertices to be checked is less, subdividing the object with an octree. This is because objects move and deform over time and we can use the trees comparison to decide which part to expand. We decomposed each object into subobjects, using AABBs or spheres. The subdivision process is made in a preprocessing stage using a regular octree to store the vertices of all the objects. This process takes $O(nlogn)$ time, n being the number of vertices of the object and log is in base 8. Remembering that the BV should be updated every time step, in the Narrow Phase we can reduce the number of vertices to deal with, using only the parts colliding.

Now, we are ready to start the Narrow Phase. The input is the pair collided that the Broad Phase reports. Using spheres the basic idea in pseudocode is presented in Figure 4.

Narrow − Phase($S_k^h(O_i)$, $S_k^h(O_j)$)
1. *For each child of* $S_k^h(O_i) in level h+1$
2. *If collision with* $S_k^h(O_j)$
3. *For each child colliding*
4. *Compare with children of* $S_k^h(O_j)$
5. *If collision happens*
6. *store pair* ($S_k^{h+1}(O_i)$, $S_k^{h+1}(O_j)$)

Figure 4: *The Narrow Phase pseudocode.*

The procedure is between two nodes in the same level k, starting with comparisons between children of node 1 from O_i and node 2 from O_j (lines 1 and 2); if a collision occurs, then we compare this result with children of node 2 (line 4) to obtain the final result, pairs colliding in level $k+1$ (line 6). The complexity is $O(4^{logn}logn)$ and the routine is called several times, depending on which nodes are colliding. The maximum number of children colliding when comparison takes place is $4*4 = 16$. Observe that in binary trees, the maximum number of nodes colliding is 1, and with the 4-ary trees, the maximum number of nodes colliding is 2.

In pairwise collisions we could update the colliding region on object O_i, but problems can arise when other objects are closer and collide with that object. Object decompositions are different, so that we can have several levels that complicate the tracing process. Of course an object can be in two or more pairwise collisions reported by the Broad Phase because it has collided with several objects at the same time. The maximum number of collisions per object equals $n-1$, where n is the number of objects. A snapshot of the Narrow Phase using AABBs is shown in Figure 5.

As we saw in the pre-computation stage, the only difference when dealing with spheres and AABBs is the question of collision. The process finishes when $level-allowed$ is reached, and the output of the Narrow Phase is a set of vertices colliding between two nodes. We have not made the next phase, the contact determination which computes the collision among primitives, V, E, or F.

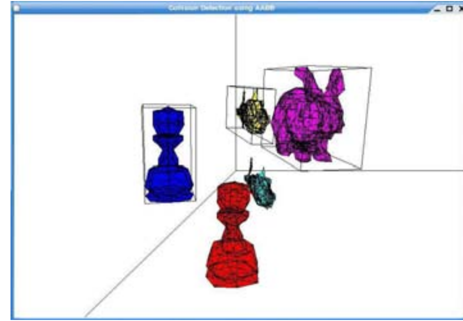

Figure 5: *A collision in the Narrow Phase using AABBs.*

7. Experiments

The time complexity for computing the collision depends on the number of vertices of the objects, and the number of objects. Also, the smaller the size of the scene, the larger the number of collisions among objects. Experiments were focused in the comparison of the two BVs, AABBs and spheres, with three, six and nine objects. The models used were a pawn with 154 vertices and 304 faces, a fish with 216 vertices and 338 faces, and a rabbit with 453 vertices and 902 faces. For instance, handling three objects with spheres, we ran five times with a 10 minute execution time. In each one we recorded the number of collisions in the Broad and Narrow Phases. Results for the Broad Phase were 18,35,32,17,21, as Table 1 shows. Note that AABB allows more collisions than spheres, so it proves that the overlapping test is less expensive than the test for spheres.

According to the theory, the Narrow Phase takes more time to be computed and explains the smaller number of collisions reported. In the Broad Phase a tree is traversed, but in the Narrow Phase two trees are traversed so there is more computation. Experiments have shown that the average number of collisions in Broad Phase, using spheres, with 3 objects is 12.4, giving a collision every 48.38 seconds. This is lower when compared with the use of the AABB under the same condition which took 24.39 seconds. In the Narrow Phase we have a collision every 96.77 seconds using spheres and 41.09 seconds with AABBs.

This relationship is reduced when dealing with 6 objects;

© The Eurographics Association 2006.

in the Broad Phase using spheres we had a collision every 12.34 seconds and every 10.16 seconds using AABBs. In the Narrow Phase we have 21.27 seconds per collision with spheres and 15.70 seconds with AABBs.

With 9 objects in the Broad Phase, with spheres, we had a collision every 7.89 seconds and every 7.20 seconds with AABBs, (very similar). In the Narrow Phase we got 17.24 seconds with spheres and every 12.29 seconds with AABBs.

Using our results to study the behaviour of the octree, we can see that less collisions have been detected when using spheres, but this number is almost the same as that in the Broad Phase for 10 or more objects. In the case of the Narrow Phase the number of collisions is smaller but it has the same pattern of behaviour. In other words it will not be similar in both BVs until we get 20 or more objects.

Object	Collisions in BP	Collisions in NP
3	AABB=18,35,32,17,21	14,22,18,10,9
3	Spheres=12,18,10,15,7	7,8,7,4,5
6	AABB=85,69,53,43,45	51,45,37,29,29
6	Spheres=46,53,53,49,42	31,29,32,30,19
9	AABB=109,78,72,66,91	64,42,51,45,42
9	Spheres=61,84,85,86,64	24,37,45,44,24

Table 1: *Number of collisisons detected.*

To use more complex objects in the octree, we conducted another experiment with two more models, each one was used three times in the scene. Three objects of the first model, a human model with 1250 vertices and 2500 faces, gave the same behaviour with both BV as shown in table under the same conditions; the difference was that with more primitives, the physical motion reduces its speed. The other model, also with three in the scene, a bike with 9543 vertices and 18238 faces, reduced the difference between the number of collisions in both phases, about one or two collisions more with AABBs than with spheres. That means that with more complex objects we could get a similar number of collisions in the two BV.

The increase in the number of primitives does not mean a large depth in the BVH, assuming they are of the same size. Smaller primitives imply more depth in the octree because regions can bound tighter primitives, or using other methods for the BVH. There are two basic methods to wrap a surface: top-down and bottom-up. In this work we use a top-down approach starting to bound the whole object in the first step, and subdivide in octants the surface in the second step, and so on. The process stops when a triangle is bounded; if we continue the algorithm, then this triangle would be subdivided into eight regions, that is incorrect because we do not have more vertices in there. Surfaces with the same triangle sizes have the same number of subdivisions and surfaces with smaller triangle sizes will have more subdivisions. This is important in the Narrow Phase because the algorithm finishes when the $level - allowed$ is reached, a value that should be specified in the BVH.

The main routines in the algorithm are the Broad Phase and the Narrow Phase. The Broad Phase involves two processes, the region-BV intersection test and the overlapping intersection test in pairs. The first process takes $O(nlogn)$ time because in each level of the region we must check the number of vertices inside a region of the octree. The second process is a pairwise collision detection technique, that in the case of the AABBs takes O(12) and O(11) for the spheres. This phase can be changed using k-ary trees to observe the performance, so that the complexity would change in the $log_k n$. Our Broad Phase method is better than the brute force Broad Phase, and comparable with the sweep and prune method.

The second big routine is the Narrow Phase that uses the BVH to compare the two trees of the objects, making the traversal process in a top down approach. Basically, it is a tracing process considering the two object's octrees. In the first step the eight children nodes of O_i are compared with the parent node of O_j. The result that can not be more than 4 children nodes of O_i, is compared with the eight children nodes of O_j, getting a maximum number of steps equals 8 + 4(8) = 8(4+1) that means $4^k 8(5) logn$, being the result no more than 4 pair nodes, i.e. four nodes in O_i and four nodes in O_j. This is for one level, but it is an increasing process, thus we have $O(4^{logn} logn)$ complexity time. The advantage is that we deal with a smaller number of vertices, because when we are moving down along the tree, we take the vertices bounded with smaller boxes or spheres, and then we spend less computation time.

8. Conclusions

We have presented a fast method that uses octrees in two phases (the Broad Phase and the Narrow Phase), to maintain uniform processing in order to reduce computation in deformable objects. The algorithm is particularly useful for interactive simulation in large 3D environments. Algorithms based on spatial partitioning are independent of the model used for the deformation calculation of objects and we have shown that it can be used in the Broad Phase and with BVH in the Narrow Phase.

Collision detection is based on the comparison between the objects involved and it can be done using trees in Narrow Phase and Broad Phase, taking advantage of the spatial coherence in the primitives. An object can be represented in several ways, and sometimes we could examine the behaviour between its motion and deformation to know at interactive rates the location of the primitives. However, our program is independent of this relationship, and the user can insert his own deformation routine. Using octrees the procedure can be done in an easy and flexible way to adjust to the application.

© The Eurographics Association 2006.

Given an environment composed of triangulated objects, our algorithm computes the overlap between regions and BVs, as well as between BVs. The complexity of the algorithm in the Broad Phase is less than $O(n^2)$, and in the Narrow Phase is $O(4^{logn} logn)$. We implemented the algorithm on a PC with a 3 Ghz processor and 1Gb of memory. The tests involved 3,6, and 9 moving DO in an octree region. The algorithm is easy to implement and makes no assumption about the physical model, and motion. It is able to compute all the contacts among multiple objects at interactive rates.

There are some improvements to be done in the future such as the pre-computation stage making tighter nodes, using a method similar to [BO04]. Also, it would be interesting to apply the bottom up method, and study the use of other BVs, such as the OBB. In addition to the contact determination phase, the use of a more complex DO, and a larger number of objects to be dealt with are further topics to examine in detail.

References

[Arv90] ARVO J.: *A simple method for box-sphere intersection testing*. Andrew Glassner, editor, Graphics Gems, Academic Press, New York, 1990.

[Bar92] BARAFF D.: *Dynamic simulation of non penetrating rigid bodies*. PhD thesis, Cornell University, Computer Science Department, 1992.

[BO04] BRADSHAW G., O'SULLIVAN C.: Adaptive medial axis approximation for sphere-tree construction. *ACM Transactions on Graphics 23*, 1 (2004), 1–26.

[ES04] EBERLY D., SHOEMALE K.: *Game Physics*. Morgan Kauffmann Publishers, San Francisco, 2004.

[GDO00] GANOVELLI F., DINGLIANA J., O'SULLIVAN C.: Buckettree: Improving collision detection between deformable objects. In *Proceedings of Eurographics* (2000).

[GLGT98] GREGORY A., LIN M., GOTTSCHALK S., TAYLOR R.: *H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction*. Tech. Rep. TR98-032, University of North Caroline, Chapell Hill, 1998.

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: Obb-tree: A hierarchical structure for rapid interference detection. In *Proceedings on SIGGRAPH 96* (New York, 1996), ACM, pp. 171–180.

[GRL03] GOVINDARAJU N. K., REDON S., LIN M. C.: Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In *EUROGRAPHICS 03* (Germany, 2003), EG.

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M.: Volumetric collision detection for deformable objects. In *Proc. VMV 03* (2003), pp. 461– 468.

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self collisions using image space techniques. In *Proc. VMV 04* (2004), pp. 145–152.

[Hub95] HUBBARD P.: Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics 1*, 3 (1995), 218–230.

[JP04] JAMES D., PAI D.: Bd-tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (2004).

[JTT01] JIMÉNEZ, THOMAS F., TORRAS C.: 3d collision detection: A survey. *Computer and Graphics 25*, 2 (2001), 269–285.

[Klo98] KLOSOWSKI J.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans on Vis and Computer Graphics 4*, 1 (1998), 21– 36.

[KNF04] KIMMERLE S., NESME M., FAURE F.: Hierarchy accelerated stochastic collision detection. In *Vision, Modeling, and Visualization* (Stanford, California, 2004).

[KZ05] KAVAN L., ZARA J.: Fast collision detection for skeletally deformable models. *Computer Graphics Forum 24*, 3 (2005), 363–372.

[LAM01] LARSSON T., AKENINE-MöLLER T.: Collision detection for continuously deforming bodies. In *Eurographics 2001, Short Presentations* (Manchester, September 2001), Eurographics Association, pp. 325–333.

[LG98] LIN M. C., GOTTSCHALK S.: Collision detection between geometric models: a survey. In *Proc. IMA Conference on Mathematics of Surfaces, 1998* (1998).

[PLM95] PONAMGI M., LING M., MANOCHA D.: Incremental collision detection for polygonal models. In *Proceedings of the eleventh annual symposium on Computational geometry* (Vancouver, British Columbia, Canada, 1995), pp. 445– 446.

[SKTK95] SMITH A., KITAMURA Y., TAKEMURA H., KISHINO F.: A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95)* (1995), pp. 136–145.

[SOM04] SUD A., OTADUY M., MANOCHA D.: Difi: Fast 3d distance field computation using graphics hardware. In *EUROGRAPHICS 04* (Germany, 2004), vol. 3, EG.

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV'03* (2003), pp. 47– 54.

[THMG04] TESHNER M., HEIDELBERGER B., MÜLLER

© The Eurographics Association 2006.

M., GROSS M.: A versatile and robust model for geometrically complex deformable solids. In *Proceedings of Computer Graphics International 2004* (USA, June 2004), IEEE, pp. 312–319.

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M., FAURE F., MAGNENAT-THALMANN N., STRASSER W., VOLINO P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61– 81.

[VB04] VERTH J. M. V., BISHOP L. M.: *Essential Mathematics for Games and Interactive Applications*. Morgan Kauffmann Publishers, 2004.

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *Journal of Grpahics Tools 2*, 4 (1997), 1– 14.

© The Eurographics Association 2006.