Collision Detection for Continuously Deforming Bodies

Thomas Larsson[†] and Tomas Akenine-Möller[‡]

[†]Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden
[‡]Department of Computer Engineering, Chalmers University of Technology, Gothenburg, Sweden

Abstract

Fast and accurate collision detection between geometric bodies is essential in application areas like virtual reality, animation, simulation, games and robotics. In this work, we address the collision detection problem in applications where deformable bodies are used, which change their overall shape every time step of the simulation. We propose and evaluate suitable bounding volume trees for deforming bodies that can be pre-built and then updated very efficiently during simulation. Several heuristics for updating the trees due to deformations are compared to each other. By combining a top-down and a bottom-up update strategy into a hybrid tree update method, promising results were achieved. Experiments show that our approach is four to five times faster than a previously leading method.

1. Introduction

Fast and reliable collision detection is of great importance in areas like real-time graphics, virtual reality, games, animation, CAD, robotics and manufacturing. Today, scenes of hundreds of thousands of polygons can be rendered in real-time using dedicated commodity graphics hardware and powerful workstations or desktop PCs. This rendering and computational power make new kinds of applications possible, with higher demands on performance and geometric detail. One such possibility is the simulation of geometrically complex scenes of multiple continuously deforming bodies. Some examples of deformable objects include soft tissues and organs, articulated characters with clothing, biological structures as well as other soft or elastic objects or materials.

A significant amount of research has been done regarding collision detection algorithms in virtual environments. Most of the efforts have been concentrated around solving the collision detection problem for rigid body simulation. When rigid bodies are used, many of the techniques used for collision detection are heavily based on data structures that can be more or less pre-computed before the simulation start. This work is of great importance in many industrial applications, for example in virtual prototyping or in virtual walkthroughs of architectural models. But in cases where deformable bodies are used, the proposed methods for rigid bodies cannot be used directly. Since the shapes of the bodies are changed, the data structures used to accelerate the collision queries must either be rebuilt or updated in ways that are not normally needed for rigid bodies.

In this paper we describe a method for efficient collision detection of multiple translating, rotating and deforming bodies. It is assumed that all bodies change their overall shape every time step throughout the simulation, i.e. all the meshes' vertices are repositioned at every time step. The proposed algorithm uses bounding volume trees adapted for such deforming bodies. The effects of different variations in the way the trees are constructed and updated are examined. Some of the interesting questions are: What kinds of bounding volume trees are suitable to use? What heuristic should be used to partition the geometric primitives of a body into its tree? How can the bounding volumes in a tree be updated efficiently? We examine trees where the nodes can have up to two, four or eight children. For the partitioning we use two basic strategies. The first is based on the initial body's shape and primitives close to each other are grouped together in the nodes of the tree. The second strategy is based on the body's mesh connectivity; all the primitives placed under any given node in a tree are neighbours in the body's polygon mesh. In the first case we call a tree the *initial shape tree* and in the latter case we call it the *mesh connectivity tree*. Both of these two tree hierarchies can be pre-constructed and efficiently updated during simulation. To update the necessary bounding volumes in a tree after a deformation has been applied to a body, we use a combination of an incremental bottom-

© The Eurographics Association 2001.

up update and a selective top-down update, which we call a *hybrid update*.

The rest of this paper is organised as follows. The next section gives a brief overview of some of the previously suggested methods for solving the collision detection problem. Then follows a description of the new collision detection algorithm. After that, experiments and results are presented. Finally, some possible future work and conclusions that can be drawn from this work are described.

2. Previous Work

The collision detection problem has been addressed in many papers. A recent survey[1] classifies different solving approaches into four general groups. Another survey[2] focuses more on how the model representation leads to different collision detection algorithms.

In environments with n moving bodies, the first step of an algorithm is typically to reduce the $O(n^2)$ running time needed to perform intersection tests on all possible pairs out of n bodies. This part of a collision detection algorithm is commonly referred to as the broad phase. One possibility is to use a spatial subdivision of the space in cells[3]. In another approach[4], a sort and prune method is used. Other spatial decomposition techniques that have been used are octrees[5], k-d trees[6], BSP-trees[7] and brep-indices[8]. An event-driven approach has also been proposed[9] that efficiently detects collisions among multiple moving spheres by using a hierarchical uniform space subdivision scheme.

Typically, in those cases where the broad phase of the algorithm is not able to determine the collision status, the narrow phase takes over in order to do more detailed intersection calculations. To speed up the intersection tests of these close body pairs, bounding volume hierarchies are commonly used. Some of the bounding volumes that have been used to build such hierarchies are for example spheres[10, 11, 12, 13], Axis Aligned Bounding Boxes (AABBs)[4, 6], Oriented Bounding boxes (OBBs)[14, 15], k-DOPs[16], Quantized Orientation Slabs with Primary Orientations (QuOSPOs)[17] and spherical shells[18]. Another possibility is to partition objects into voxelised containers, without using any hierarchical organisation within the containers[19].

Another class of algorithms efficiently tracks the closest features between convex bodies or bodies decomposed into a set of convex pieces. By doing so, they are not only able to report collisions, but also to report the shortest distance between bodies. Some of these methods use pre-computed Voroni regions[4, 20]. Others treat the body as the convex hull of a point set and operate on simplices defined by subsets of these points[21, 22, 23]. The incremental hierarchical walk algorithm[24] efficiently maintains the distance between moving convex bodies by exploiting both motion coherence and hierarchical representations.

There are also some four-dimensional approaches for solving the collision detection problem for moving bodies[10, 25]. By considering the intersection of four-dimensional volumes swept out by body motion over time, future contact times can be calculated. These methods require that information about the bodies' velocities and accelerations can be given beforehand. Some cases of dynamic object-object intersection are described by Eberly[26].

Usually these mentioned methods have been demonstrated to work efficiently in different kinds of environments for rigid body simulations. When we consider the problem of deforming bodies, they are not as useful, since they rely heavily on pre-computed data and data structures or they are dependent on certain body characteristics, for example bodies that must be decomposed into convex pieces. In fact, even if there exist many documented works on collision detection in virtual environments, there are significantly fewer that have dealt with deforming bodies.

A very general collision detection method for deformable objects has been proposed by Smith et al.[27] The input models can be groups of deforming triangle soups freely moving in space. At every time step, the AABB of all objects is calculated. When two overlapping AABBs are found, object faces are first pruned against their overlap region. Remaining faces from all such overlap regions are used to build a world face octree, which is traversed to find faces located in the same voxels. The high performance of this method breaks down in hard cases, i.e. when an overlap region is large and there are many geometric primitives (overlapping or not) in that region, which are passed on to the face octree building stage.

A data structure called the BucketTree has also been proposed[28], which is an octree data structure with buckets as leaves where geometrical primitives can be placed. At every time step of a simulation, the models' primitives are assigned to an appropriate bucket. Then the intersection tests between any two models are done recursively by testing the nodes AABBs as their trees are traversed. This algorithm is also very general, since it only sees an object as a soup of freely moving primitives.

Another approach is suggested by van den Bergen[29], which is also used in the collision detection library called SOLID[30]. Initially, AABB trees are built for every model in its own local coordinate system. The AABBs in the trees are then transformed as the models are moved or rotated in the scene. This transformation causes the models' locally defined AABBs to become OOBs in world space. When a model is deformed an update of the affected nodes in the trees has to be done.

In the literature, there are also some other algorithms for flexible objects[5, 31]. Some methods are designed for bodies undergoing polynomial deformations[32, 33].

All the work mentioned above is interesting, but efficient interference detection between deformable bodies is definitely an area worth more attention[1]. Better methods are

© The Eurographics Association 2001.

needed and much work remains to be done. In the following sections we describe our implemented method in more detail.

3. Algorithm Overview

Many practical algorithms are for performance reasons so called discrete methods, i.e. they report contact between bodies when they have already interpenetrated each other. Our algorithm is also a discrete method. If needed, back tracking in simulation time might be used to determine the colliding bodies' first contact.

To efficiently detect collisions between multiple continuously deforming bodies represented by polygon meshes, we propose an algorithm divided into two loosely coupled main phases. The first or the broad phase uses a sort and prune method[4] to find the bodies that are close to each other, and the second or the narrow phase uses bounding volume trees to determine the intersection status between bodies, which have already been found to be close to each other, in a more detailed manner. A schematic overview of the algorithm's main parts and its working context is given in Figure 1. It is assumed that the application using the collision detection module drives the simulation forward and calls the collision detection algorithm at appropriate time intervals. The application is also responsible for translating the reported collision status into suitable response actions.

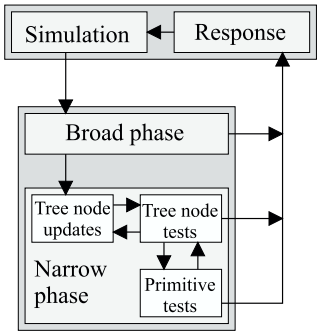

Figure 1: *Schematic overview of the collision detection module for deforming bodies and its communication with an application specific simulation module.*

In the narrow phase of the collision detection algorithm, there are three main problems that have to be solved efficiently. First of all, nodes in the bounding volume trees must be updated every time step of the simulation, since the bodies are deformed continuously. Therefore, we chose AABBs as our bounding volume. When bodies deform, the corresponding AABBs can be recalculated very efficiently to reflect the changes in the geometry. Also, when testing the intersection status of tree nodes during collision traversals, AABBs are very efficient to do intersection tests with. Finally, the close face pairs that the collision traversals sort out must be interference tested explicitly. For this test, we use the method provided by Akenine-Möller[34].

Different parts and aspects of the algorithm, as well as some variations, are described in more detail in the following sections.

3.1. Deformation Types

A body might undergo a complete change of shape, from one time step to the other, by moving the relative position of all of its vertices. We refer to this type of deformation as *arbitrary vertex repositioning*. During such deformations, the mesh connectivity stays the same, i.e. the mesh is not torn up in any way. Our method efficiently handles this kind of deformation. Sometimes other types of deformation are desired; such as increasing or decreasing the number of geometric primitives in the mesh or splitting the body into new separated pieces. Currently, we do not support these kinds of deformation in our method.

3.2. Bounding Volume Pre-processing

For all input bodies, bounding volume trees are initially built as a pre-processing stage. A tree is built by repeatedly splitting the geometry in the parent AABB into smaller AABBs until there is only one geometric primitive left in them. We have chosen to support the building of bounding volume trees where the maximum degree of a node can be two, four or eight, and we refer to these trees as binary trees, 4-ary trees or 8-ary trees.

For the geometric splitting, many different rules can be used. We have tried two different main strategies. Both of them build the trees in a top-down manner. The first one builds the tree based on the initial shape of the body. Depending on the maximum degree of a tree node, a parent AABB is split along one, two or three coordinate axes into two, four or eight sub-volumes. Then the midpoint of each geometric primitive is assigned to one of these sub-volumes and a child node is created for every non-empty sub-volume. If the degree of the tree is two, then the parent AABB is split along its longest side, if it is four the split is done along the two longest sides and if it is eight all three sides of the parent AABB are split. To choose the actual values for the split planes two different heuristics have been examined. The simplest one picks values from the coordinates of the centre point of the box. The other heuristic calculates the average point of all polygons' midpoints and the values for the split planes are chosen from that point. In our experiments, there is no significant difference between these two ways of choosing the values for the split planes, but we prefer the first one since it is a more efficient operation. (Some other split methods have also been described and examined[29, 16]). The partitioning into new child nodes is repeated recursively

© The Eurographics Association 2001.

until there is only one geometric primitive left per node. We call the resulting tree an initial shape tree.

Another interesting way of building the trees is based on partitioning the geometry of a body into a tree we call the mesh connectivity tree. The partitioning is done so that all faces under a certain node in the tree form a connected neighbourhood. Even for highly deformable bodies, the connectivity of our meshes always stays the same. This way of partitioning the geometry does not pay much attention to the initial shape of a body, which might be completely different after some deformations have occurred. A tree is built in the following way. The whole mesh is associated with the root node. Then, depending on the maximum node degree in the tree, which in our case can be either two, four or eight, the mesh is split into a suitable number of sub-meshes and placed in new nodes inserted as the root node's children. The partitioning into new child nodes is repeated recursively until there is only one geometric primitive left in a node. A possible advantage of these trees is that they avoid the potential risk of grouping faces deep down in the trees that are very close to each other initially, although they may not be close at all, when we only consider the connectivity of the faces in the mesh. This type of surface-based hierarchy has been suggested for building good fit OOBs and used to speed up radiosity calculations as well as for collision detection of rigid bodies[35]. In contrast, we are interested in examining their properties when dealing with deforming bodies.

A potential problem with building the bounding volume trees initially is that deformations applied during run-time can drastically change the volumes of the AABBs and also cause increases in their overlap among themselves. An alternative to pre-build the trees would be to rebuild them when their qualities have degenerated to a certain extent. Rebuilding the trees, however, is a much more expensive operation compared to only updating the bounding volumes in an otherwise fixed tree. In many practical cases, rebuilding is not needed[29].

When dealing with continuously deforming bodies, we have also found in our experiments that using 8-ary tree versions of the bounding volume trees was a slightly better choice than both 4-ary and binary versions of the trees. In the 8-ary tree case, fewer bounding boxes need to be calculated each time step and the search towards contact regions converges faster per entered level in the tree traversals.

3.3. Run-time AABB Updates

During run-time, we have to update bounding volume trees due to deformations. But in a typical collision traversal, far from all bounding boxes in a tree are needed. Therefore, we have tried to update as few AABBs as possible by updating them top-down, as they are needed during the traversals. In this case the AABB of a node is calculated by traversing the faces placed under it. If the meshes have connectivity information, i.e. the polygons share a list of common vertices, we update the node's AABB by traversing the shared vertices of the faces in the node, instead of the faces themselves, which is typically much faster. As an alternative, the AABBs in a tree can be updated incrementally bottom-up[29], starting from the AABBs of the leaves and merging them upwards to the root of the tree. The strength of this method is that a parent AABB can always be calculated very efficiently directly from the AABBs of its children, but on the other hand all tree nodes are visited and updated, despite the fact that only some of them will be needed in the following tree traversal.

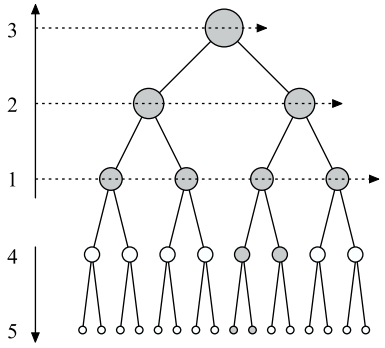

Figure 2: *Example of a hybrid tree update method, combining the bottom-up and top-down strategy.*

We have found that in hard cases, where many deep nodes in a tree are reached during a collision test, it gave a better overall performance to update the AABBs in a tree bottom-up. In simple cases, however, with only a few deep nodes visited in a collision test, the top-down update normally performs better. What we would like is a method to update the trees, which performs well in both simple and hard cases. Therefore, our approach is to use a hybrid update method that combines efficient bottom-up calculations with selective top-down updates, which gives the desired result. The method attempts to update as few AABBs as possible, while still updating the ones covering most faces in the top of a tree bottom-up. For a tree with depth n, we initially update the n / 2 first levels bottom-up, which we have found to be an efficient choice. During a collision traversal, when non-updated nodes are reached, they can either be updated top-down as needed or a specified number of levels in their sub-trees may be updated bottom-up. For the models that we have examined, with a typical triangle count between 5000 and 32000, we have found it fastest to update these nodes top-down as they are needed. An illustration of the hybrid tree update is given in Figure 2 for a very simple binary tree. First the three topmost levels in the tree are updated bottom-up (step 1 to 3). Then during a collision traversal, when non-updated nodes are reached, they are updated on the fly (step 4 and 5). There are 31 nodes in this small example tree, but only 11 of them are updated (those that are marked grey). In practise, the trees are much larger and so is the difference between the number of non-updated and updated nodes.

© The Eurographics Association 2001.

A drawback of our hybrid update method (as well as the top-down method) is that we have to store vertex or face information in the internal tree nodes, not only in the leaf nodes. This memory cost is another reason for using 8-ary trees, with fewer nodes, compared to 4-ary or binary trees.

3.4. Multiple Body Simulation

For simulations with up to approximately 100 bodies, the naïve brute force technique, comparing $n(n\text{-}1)/2$ body pairs for n bodies, performs very well. But if there are more bodies in a simulation, our first phase uses the sweep and prune sorting technique suggested by Cohen et al.[4] Initially, all extents of the objects along the three principal axes are sorted into three lists. These lists can be used to efficiently find all objects close to each other. As objects move, the lists are resorted during all stages of the simulation. The changes in relative placement of the bodies are expected to be small from one time step to the next and the resorting operation is thus expected to take $O(n)$ time for n bodies. The $O(n^2)$ running time complexity for checking collision among n bodies is reduced to $O(n+m)$, where m is the number of pairwise overlaps between the bounding volumes of the bodies.

To avoid calculating the best fitting AABB for all bodies in the world at every time step, we first use predetermined loose AABBs that are large enough to bound every possible orientation and deformation of the bodies, whenever possible. If it is not possible to determine such loose AABBs, for example, because of the unknown bounds of the possible deformations, then an AABB has to be calculated for all bodies before the sweep and prune technique can be used.

4. Experiments and Results

We have done many different experiments to investigate the performance characteristics of our proposed method. The experiments used have to be chosen with care, since the results depend on the shapes of the models, their relative orientation and the deformations applied. Three of the experiments are presented here, which were all done using a Pentium III, 550 MHz CPU. Our scenarios were chosen before our hybrid update method was developed and the results of our algorithm are compared to the results from our implemented version of the method by van den Bergen[29], which is similar to ours.

In the first experiment, two continuously deforming bumpy sphere bodies were moved slowly into each other during 200 simulation time steps. Each one of the two bodies consisted of 20 480 triangles. The collision queries ranged from very simple cases in the beginning to quite hard cases towards the end of the simulation. The very first intersection of the bodies was reported at time step 60. In the last time step, 3760 intersecting triangle pairs were reported. We used the 8-ary version of our initial shape trees and we reported the collision detection times per time step for the top-down, bottom-up as well as the hybrid tree update methods. The results are given in Figure 3. In Figure 3a, the timings for reporting all intersecting triangle pairs are reported, and in Figure 3b the timings for finding a first arbitrary intersecting triangle pair is reported, which in cases where there are a lot of intersecting triangles is much faster. Reporting only one triangle pair might be sufficient for many applications. For example, if we want to search for the exact time for the bodies' first contact, it is sufficient to find one intersecting triangle pair to know that we need to back track in simulation time. As we can see, the hybrid update method performs best, both in simple and hard cases. Furthermore, it is approximately five times faster than the competing method[29].

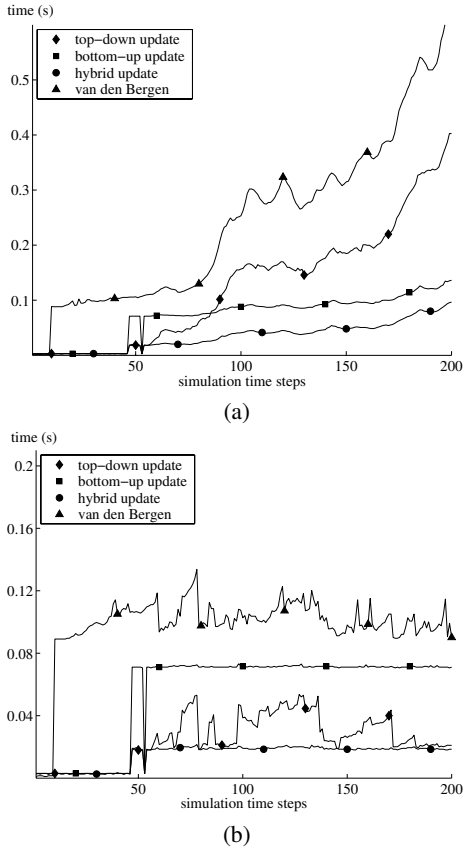

Figure 3: *Collision detection performance reported from the first experiment when a) all intersecting triangle pairs were reported, b) only the first found intersecting triangle pair was reported.*

In the second experiment, we used the same scenario as in the first experiment, but this time we ran the simulations multiple times with varying polygon counts for the bumpy spheres. For every simulation the collision detection time at the time step where the bodies first hit each other as well as the worst time were reported. In Table 1 we can see the results. The last column reports the number of intersecting

© The Eurographics Association 2001.

triangle pairs that were found during the worst time step. It is obvious that, for this experiment, the measured results indicate roughly linear performance in the number of used faces.

faces per body	first contact (ms)	worst contact (ms)	triangle intersections
1280	2	4	138
5120	6	15	562
8192	8	28	1010
16384	15	64	2422
20480	20	98	3760
32768	30	151	4775
65536	62	334	12358

Table 1: *Running time for deforming bodies with different polygon counts.*

In the third experiment, 27 translating, rotating and deforming bodies hit each other frequently during 200 simulation time steps in a rather dense environment. A simple collision response method was applied to prevent the bodies passing through each other. Each one of the bodies consisted of 5120 triangles. The simulation was repeated twice. In the first simulation, the bodies were bumpy spheres and in the second simulation the bodies had multiple deforming arms. The collision detection performance using our hybrid update method with 8-ary versions of the initial shape trees is presented in Figure 4. Also, the performance of van den Bergen's method is included for comparison. When using our method, the average collision detection time per time step is about a factor of 5.6 faster, in Figure 4a, and a factor of 4.5 faster, in Figure 4b, than when using the method by van den Bergen.

The major differences between our method and van den Bergen's are in the way the trees are updated and how the intersection tests are done between the bounding volumes in the nodes during tree traversals. Where van den Bergen uses a complete bottom-up update of the bounding volume trees, visiting every node, we use the hybrid update, combining benefits from both bottom-up and top-down approaches. Also, because we use world coordinate space AABB trees, we have to calculate world coordinates of the vertices in the bodies before a collision tree traversal, i.e. if the simulation process does not already provide them. (In some cases it might be more convenient to apply deformations directly in world coordinate space). Anyway, this makes it possible to use very fast AABB intersection tests during the tree traversals. Van den Bergen, on the other hand, uses local coordinate space AABB trees, which in fact becomes OBBs in world coordinate space, and then the intersection tests between tree nodes are a more expensive operation, like the used *SAT lite* test[29], which starts to dominate the running time in hard cases (see Figure 3a). Another difference is that

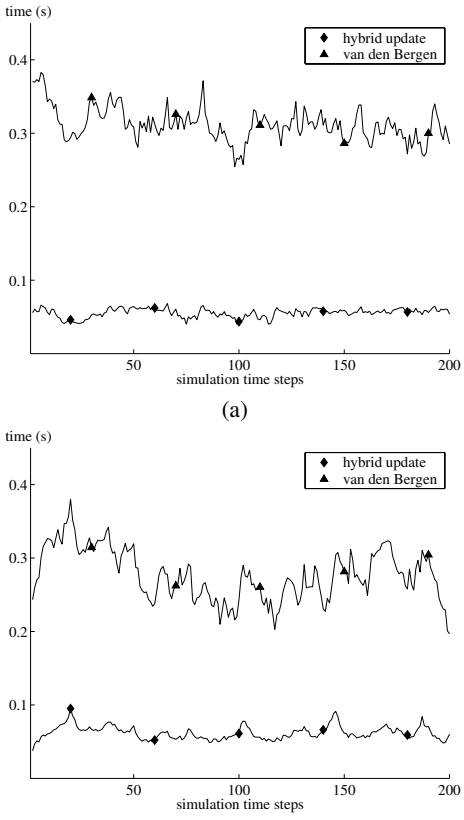

Figure 4: *Collision detection performance per time step according to experiment 3 where a) 27 bumpy sphere bodies were used. b) 27 bodies with multiple arms were used.*

we use 8-ary trees instead of binary trees for our bounding volume hierarchies. (We have implemented an 8-ary tree version of van den Bergen's method, which runs approximately 10 to 20 percent faster than the binary version of it in our experiments). Finally, it is worth mentioning that the purpose of van den Bergen's method is to deal with both rigid and deformable bodies in a unified framework. We have not aimed at supporting rigid bodies efficiently in our algorithm.

We have also tried our mesh connectivity trees in these experiments, but the performance difference is very small between them and the initial shape trees for the type of bodies we have used. The average collision detection time per time step is typically between zero to 10 percent better for the mesh connectivity trees than for the initial shape trees in these experiments. Despite this small difference, we believe that it would be interesting to study the mesh connectivity trees further.

In Figure 5 and 6, images of the types of bodies that were used in our experiments are shown. Animations showing the reported experiments have also been produced.

© The Eurographics Association 2001.

5. Future Work

There is much more interesting work to do regarding collision detection and deforming bodies. For example, cut operations, where bodies are torn into two or more separated pieces, might yield very hard close proximity situations for the collision detection algorithm and more efficient solutions would be desirable to increase realism while maintaining interactive performance. Fusion operations, where different bodies are merged together, form another interesting type of deformation, which might be interesting for certain kinds of applications.

Efficient algorithms that automatically create suitable mesh connectivity trees would be another interesting topic to study, so their usefulness for operations like collision detection could be evaluated. Another very important feature for deforming bodies is to avoid self-intersections. We have not included any support to avoid such situations automatically. Instead, we have assumed that the algorithm that applies the deformations to the bodies does it in a proper way. Another possible direction for future work would be to design parallel algorithms for collision detection of deformable bodies. It would also be beneficial to create test scenes suitable for comparison of different algorithms for collision detection of deforming bodies. If some suitable test scenes together with some general software were available, such comparisons would be much simpler to do.

6. Conclusions

Real-time graphics simulations, where the shapes of the bodies deform continuously over time, constitute a particular challenge since the possibilities of using pre-calculated data and data structures are dramatically decreased. The result of this work is an efficient collision detection algorithm that works well in real-time simulations for multiple moving and deforming bodies represented by polygonal meshes. The proposed bounding volumes trees are suitable to prebuild before simulation time for many types of deformable bodies and very fast to update during simulation time, due to the applied deformations. Our proposed hybrid tree update method performs well in both simple and hard collision detection cases. In our experiments, our method has been found to be approximately four to five times faster than a previously leading method for deformable bodies. The performance of the algorithm has been verified by experiments in complex dynamic environments with multiple continuously deforming bodies.

References

1. P. Jiménez, F. Thomas, C. Torras. 3D Collision Detection: a Survey. *Computers & Graphics*, **25**(2):269–285, 2001. 2
2. M.C. Lin, S. Gotttschalk. Collision Detection Between Geometric Models: A Survey. Proceedings of IMA, Conference of Mathematics of Surfaces, pp. 602–608, 1998. 2
3. G. Turk. Interactive Collision Detection for Molecular Graphics. Technical Report TR90-014, Computer Science Department, University of North Carolina at Chapel Hill, 1990. 2
4. J.D. Cohen, M.C. Lin, D. Manocha, M. Ponamgi. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. Symposium on Interactive 3D Techniques, Proceedings of the 1995 symposium on Interactive 3D graphics, pp. 189–196, Monterey, CA USA, 1995. 2, 3, 5
5. M. Moore, J. Wilhelms. Collision Detection and Response for Computer Animation. *ACM Computer Graphics (Proc. of SIGGRAPH '88)*, **22**(4):289–298, 1988. 2
6. M. Held, J.T. Klosowski, J.S.B. Mitchell. Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs. Proceedings Seventh Canadian Conference on Computational Geometry, pp. 205–210, 1995. 2
7. B. Naylor, J.A. Amatodes, W. Thibault. Merging BSP Trees Yields Polyhedral Set Operations. *ACM Computer Graphics (Proc. of SIGGRAPH '90)*, **24**(4):115–124, 1990. 2
8. W. Bouma, G. Vanecek, Jr. Collision Detection and Analysis in a Physical Based Simulation. Eurographics Workshop on Animation and Simulation, pp. 191-203, Vienna, 1991 2
9. D. Kim, L.J. Guibas, S. Shin. Fast Collision Detection Among Multiple Moving Spheres. *IEEE Transactions on Visualisation and Computer Graphics*, **4**(3):230–242, July-September 1998. 2
10. P.M. Hubbard. Interactive Collision Detection. Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, pp. 24–31, 1993. 2
11. P.M. Hubbard. Collision Detection for Interactive Graphics Applications. *IEEE Transactions on Visualization and Computer Graphics*, **1**(3):218–230, 1995. 2
12. P.M. Hubbard. Approximating Poyhedra with Spheres for Time-Critical Collision Detection. *ACM Transaction on Graphics*, **15**(3):179–210, July 1996. 2
13. I. Palmer, R. Grimsdale. Collision Detection for Animation using Sphere-Trees. *Computer Graphics Forum*, **14**(2):105–116, 1995. 2
14. S. Gottschalk, M. C. Lin, D. Manocha. OOBTree: A Hierarchical Structure for Rapid Interference Detection. *ACM Computer Graphics (Proc. of SIGGRAPH '96)*, pp. 171–180, 1996. 2

© The Eurographics Association 2001.

15. G. Zachmann, W. Felger. The BoxTree: Enabling Real-Time and Exact Collision Detection of Arbitrary Polyhedra. Proceedings of SIVE '95, pp. 104–113, 1995. 2

16. J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, **4**(1):21–36, 1998. 2, 3

17. Taosong He. Fast Collision Detection using QuOSPO trees. Symposium on Interactive 3D Techniques, Proceedings of the 1999 Symposium on Interactive 3D Graphics, pp. 55–62, Atlanta, GA USA, 1999. 2

18. S. Krishnan, A. Pattekar, M. Lin and D. Manocha. Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries. In Proceedings of WAFR '98, pp. 287–296, 1998. 2

19. A. García-Alonso, N. Serrano, J. Flaquer. Solving the Collision Detection Problem. *IEEE Computer Graphics and Applications*, pp. 36–43, 1994. 2

20. B. Mirtich. V-Clip: Fast and Robust Polyhedral Collision Detection. *ACM Transaction on Graphics*, **17**(3):177–208, 1998. 2

21. E.G. Gillbert, D.W. Johnson, S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, **4**(2):193–203, 1988. 2

22. S. Cameron. Enhancing GJK: Computing minimum Penetration Distances Between Convex Polyhedra. Proceedings of International Conference on Robotics and Automation, pp. 3112–3117, 1997. 2

23. G. van den Bergen. A Fast Robust GJK Implementation for Collision Detection of Convex Objects. *Journal of Graphics Tools*, **4**(2):7–25, 1999. 2

24. L.J. Guibas, D. Hsu, L. Zhang. A Hierarchical Method for Real-Time Distance Computation Among Moving Convex Bodies. *Computational Geomtery: Theory and Applications*, **15**(1-3):51–68, 2000. 2

25. S. Cameron. Collision Detection by Four-Dimensional Intersection Testing. *IEEE Transactions on Robotics and Animation*, **6**(3):291–302, 1990. 2

26. D.H. Eberly. *3D Game Engine Design - A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2001. 2

27. A. Smith, Y. Kitamura, H. Takemura, F. Kishino. A Simple and Efficient Method for Accurate Collision Detection Among Deformable Polyhedral Objects in Arbitrary Motion. Proceedings of the IEEE Virtual Reality Annual International Symposium, pp. 136–145, 1995. 2

28. F. Ganovelli, J. Dingliana, C. O'Sullivan. BucketTree: Improving Collision Detection Between Deformable Objects. Spring Conference in Computer Graphics (SCCG2000), Bratislava, pp. 156–163, 2000. 2

29. G. van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, **2**(4):1–14, 1997. 2, 3, 4, 5, 6

30. G. van det Bergen. SOLID. Software Library for Interference Detection, 1999, Available at http://www.win.tue.nl/cs/tt/gino/solid 2

31. A. Joukhadar, A. Scheuer, Ch. Laugier. Fast Contact Detection between Moving Deformable Polyhedra. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pp. 1810–1815, 1999. 2

32. D. Baraff, A. Witkin. Dynamic Simulation of Non-Penetrating Flexible Bodies. SIGGRAPH'92 Conference Proceedings, pp. 303–308, 1992. 2

33. M. Hughes, M. Lin, D. Manocha, C. Dimattia. Efficient and accurate interference detection for polynomial deformation. Proceedings of Computer Animation, pp. 155–166, Geneva, Switzerland, 1996. 2

34. T. Möller. A Fast Triangle-Triangle Intersection Test. *Journal of Graphics Tools*, **2**(2):25–30, 1997. 3

35. M. Garland, A. Willmot, P.S. Heckbert. Hierarchical Face Clustering on Polygonal Surfaces. ACM Symposium on Interactive 3D Graphics, 2001. 4

© The Eurographics Association 2001.

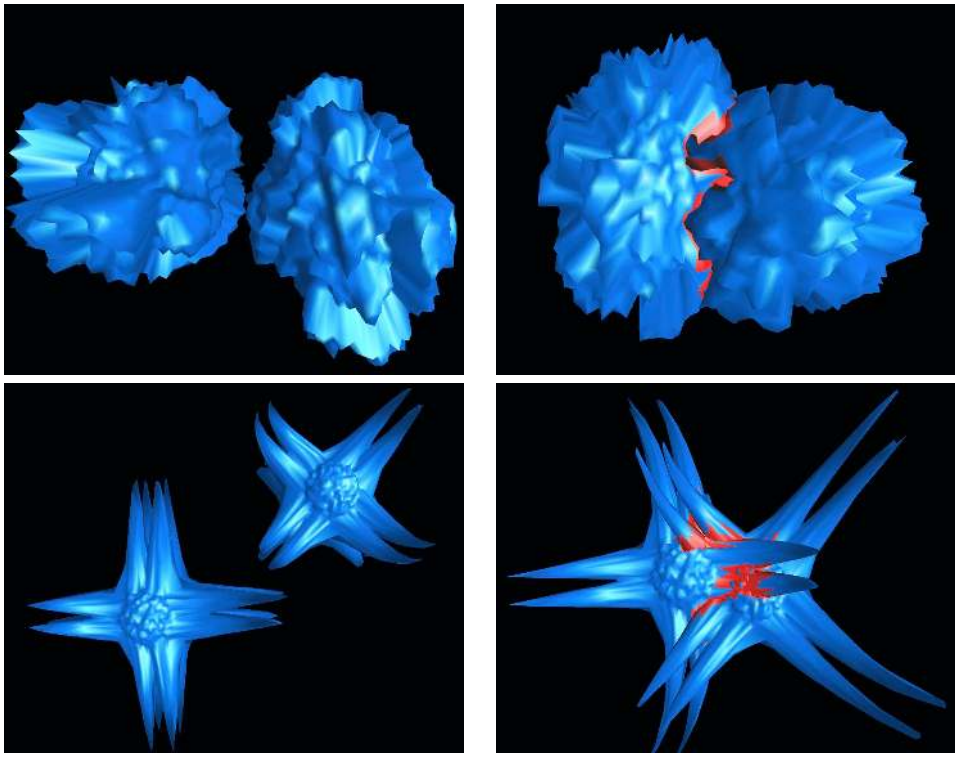

Figure 5: *Some moving deforming bodies before and after they have interpenetrated each other. Intersecting triangles are in red colour.*

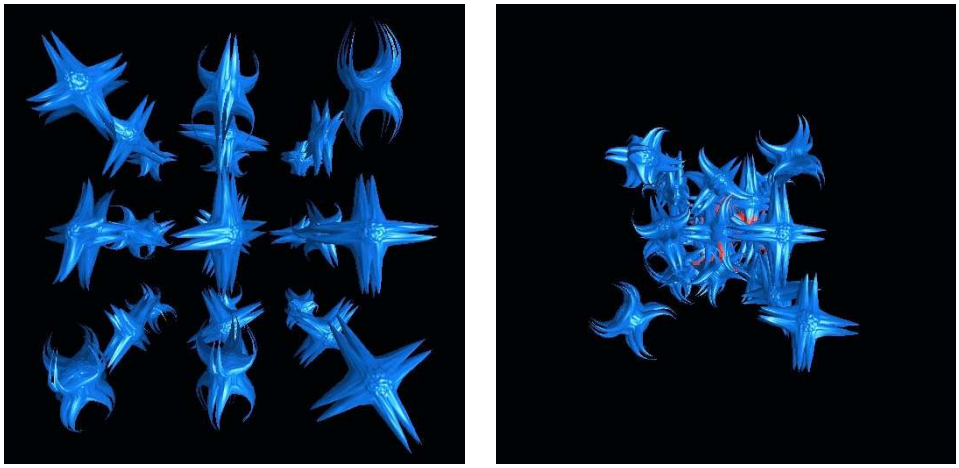

Figure 6: *Multiple deforming bodies moving from a simple start case towards a common goal point in order to stress the collision detection algorithm.*

© The Eurographics Association 2001.