Mälardalen University Licentiate Thesis
No.14

Adaptive Algorithms for Collision Detection and Ray Tracing of Deformable Meshes

Thomas Larsson

October 2003

Department of Computer Science and Engineering
Mälardalen University
Västerås, Sweden

ISSN 1651-9256
ISBN 91-88834-11-5
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

Many applications in computer graphics and visualization are directly dependent on accurate and fast intersection queries. To prevent bodies passing directly through each other, the simulation system must be able to track touching or intersecting geometric primitives. In real-time graphics simulations, in which hundreds of thousands of geometric primitives are involved, highly efficient collision detection algorithms are needed. The efficient handling of deformable models constitutes a particular challenge to the simulation system, since the possibilities of precomputing efficient data structures are decreased dramatically. The same type of problem arises in interactive ray tracing, where a huge number of geometric intersections must be determined in just a fraction of a second.

For these reasons, new efficient collision detection and ray tracing methods for deformable meshes are suggested in this thesis. The proposed solutions are based on bounding volume hierarchies which allow the models they represent to be deformed at every time step of the simulation. Different update methods to efficiently refit the bounding volumes in the hierarchies as the models deform are presented. The models considered are represented by polygon meshes that are either deformed by arbitrary vertex repositioning or by mesh morphing. The update methods postpone updates in the hierarchies until they are absolutely needed in order to avoid unnecessary updating work. The results from the experiments performed indicate that significant speed-ups can be achieved by using these new methods in comparison with approaches suggested previously. The thesis also shows that mesh morphing constitutes a specific example of a restricted type of deformation that allows particularly efficient hierarchical data structures, with expected sub-linear collision queries in the number of geometric primitives of the meshes.

To Paulina, André, and William

Preface

Ever since I bought my first home computer in 1982, a Sinclair ZX Spectrum 48K, I have been fascinated by the possibilities of computer programming. As a teenager, my interest was limited to simple game and graphics programming, and the Spectrum computer provided an excellent environment in which to begin learning their basics. At that time, I had no idea that this interest would later lead me into the academic world, performing graduate studies in computer graphics, parallel with employment as a university teacher. Today, I wonder what direction my working life would have taken, if I had not bought that tiny machine with those rubber keys.

Now, at the conclusion of this work, I would like to take the opportunity to thank my adviser Tomas Akenine-Möller for his support, guidance, and correspondence. I hope there will be many more opportunities for us to work together in the future. I would also like to thank my principal adviser Björn Lisper for the support and help he has given me during my research activities. And of course, my thanks go to my colleagues here at the department. All have helped by contributing to the positive and creative working environment which we share daily. Thank you all!

More than anyone else, I would like to thank my wife Paulina for her encouragement and patience during my studies. Without her love and support I would not have finished this work. I am also indebted to our two wonderful sons, André and William, 4 and 2 years old respectively, for the inspiration they provide and for teaching me more about life than anything else. Finally, I would like to thank my parents for always being there, and for their support during my undergraduate studies.

Thomas Larsson
Västerås, September 27, 2003

Contents

I

Thesis

Chapter 1

Introduction

1.1 Background and motivation

Computer graphics is a field of study concerned with the creation, storage, and manipulation of three-dimensional models and images. The perhaps most widely known application areas for computer graphics are in TV and moving picture production, in which images generated by computer graphics play an increasingly important role. Other important application areas are computer-aided design and virtual prototyping, in which computer graphics are used with great success as an engineering tool.

A very important application in the computer graphics field is visual simulation. For example, in flight simulation, virtual worlds are created to mimic the real world, so that novice pilots can be trained for future flight operations under safe conditions. Virtual surgery makes it possible for surgeons to practise advanced operations under realistic, but safe, circumstances. Architectural walk-through applications help architects to design buildings, and make it possible for potential customers to experience buildings before they have been built. In interactive storytelling, fantasy worlds can be explored and experienced through computer graphics imagery.

To make such applications possible, real-time rendering systems are required [1]. In this context, the term real-time means that images of the scene can be generated or rendered at a high and steady frame rate, for example a pair of stereo images at 30 frames per second. How the simu-

lation is driven forward is, in general, application-specific. For example, it can be done by applying physical laws of motion, or by applying some kind of procedural simulation rules. In interactive graphics systems, the user is allowed to control and dynamically change the state of the simulated scene, for example by using different kinds of input devices, such as mice, data gloves, and force feedback devices.

The ability to model, render, and simulate different types of scenes with bodies or models in motion is of course essential in many application areas. Most often, however, the simulated scenarios tend to become too complex, if realistic models are to be used. Imagine a visual traffic simulation application using a scene that includes detailed geometric models of all the buildings and vehicles in a city. Clearly, sophisticated techniques would be needed to simplify the simulation sufficiently to make it computationally possible, while ensuring that by running the simulations we get valuable feedback and results. To be able to handle efficiently the complexity of computer graphics simulations, specialized data structures and algorithms are needed.

The simulation of soft or elastic bodies constitutes a particular challenge. Imagine a scene inhabited by complex deforming meshes, modelled by hundreds of thousands of geometric primitives. A naive collision detection solution would be to check all possible pairs of primitives for intersection, which obviously would result in unacceptable inefficiency. Acceleration data structures are needed, but since the bodies undergo deformations, the data structures must be rebuilt or updated in some way to remain useful.

Therefore, adaptive hierarchical data structures and algorithms for accelerating collision detection in dynamically changing scenes are presented in this thesis. Such queries are essential, for example, in physical simulation, robotics, computational surgery, virtual prototyping, molecular modelling, animation, cloth simulation, and computer games. Deformable models whose contact behaviours need to be simulated include articulated characters with clothing, soft tissues and organs, biological structures, and other soft or elastic materials.

Collision or intersection queries are not only important for the detection and resolution of the collisions of moving bodies in graphics simulations. In interactive ray tracing, a huge number of ray/scene intersections must be determined as part of the rendering process. The thesis therefore presents data structures and algorithms similar to those used for collision detection to accelerate the ray tracing algorithm in scenes

with deformable models.

1.2 Solutions and results

The solutions and results achieved have been published in three different papers, now included in this thesis, and referred to as paper **A**, **B**, and **C**. These are:

A Thomas Larsson and Tomas Akenine-Möller. Collision Detection for Continuously Deforming Bodies. In Eurographics Conference 2001, Short presentations, pages 325–333, Manchester 2001.

B Thomas Larsson and Tomas Akenine-Möller. Efficient Collision Detection for Models Deformed by Morphing. The Visual Computer, 19(2–3):164–174, 2003.

C Thomas Larsson and Tomas Akenine-Möller. Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models. MRTC Report, Mälardalen University, February 2003. (Submitted for publication)

I am the main author of these three papers and I have performed the work described in them aided by beneficial and constructive discussions with my advisors.

The main contribution of paper **A** is a new algorithm for hierarchical collision detection of meshes undergoing arbitrary vertex repositioning. In particular, an efficient new hierarchy update method is given, yielding a significant speed-up compared with previous approaches. The main contribution of paper **B** is a novel algorithm for collision detection of morphing models, whose performance is of the same order as algorithms previously used for hierarchical collision detection of rigid bodies. In paper **C**, the main contribution is a new algorithm for ray tracing of deforming meshes. The paper shows that the hierarchy update method, which was proposed in paper **A**, gives a speedup of an order of magnitude in the reconstruction phase, which is needed in ray tracing of dynamically changing scenes. This makes feasible the interactive ray tracing of scenes with hundreds of thousands of deforming geometric primitives.

In all these works, the methodology employed has been based on simulation experiments and running benchmarks. The results of the experiments have been compared with those of previously suggested meth-

ods. This part of our work clearly shows that our proposed solutions are efficient and useful in practice.

It is also worth noting that the improvements of the performance of CPUs, from year to year, will not be great enough to make advanced collision detection algorithms obsolete. In computer graphics, there is a never-ending demand for larger and more complex three-dimensional scenes. In many respects, the solutions and algorithms proposed in this thesis scale well with scene complexity, and they may therefore be useful for simulation applications for many years to come.

1.3 Outline of thesis

The rest of this thesis is organized as follows. In Chapter 2 the proposed algorithms for hierarchical collision detection of deforming meshes are described. Chapter 3 discusses means of accelerating ray tracing of deforming models. These chapters include a summary of the papers **A**, **B**, and **C**, and a brief description of the main contributions of each. For a more detailed treatment, please consult the original papers, which can be found in Chapters 5, 6, and 7. Finally, Chapter 4 presents the conclusions as well as some interesting directions for future work.

Chapter 2

Collision detection of deformable meshes

2.1 Related work

Hundreds of papers have been written on collision detection in various situations, primarily in the fields of computer graphics, robotics, and computational geometry. Most described efforts, however, have been focused on solving the collision detection problem in rigid body simulation [2, 3, 4]. There is currently no single best collision detection method. The algorithm to be choosen depends on many factors that play different roles in different applications [5].

In some applications it is sufficient to use approximate methods whereas other applications might require accurate collision calculations. The best performance is often achieved by using specialized or simplified methods that utilize specific knowledge about the application. For example, in a virtual bowling application, simple cylinder approximations were used to represent the pins in the collision detection calculations with plausible results [6].

In many other cases, a sufficient accuracy of the collision calculations must be guaranteed. For example, in robotics, inaccuracies in the virtual simulation process might lead to severe damage, since the simulations are often used to verify the correctness of the corresponding real world simulations. Furthermore, in rigid body simulation, when the force com-

putations are based on the intersection data reported from the collision detection algorithm, small errors might cause fundamentally different body trajectories, which is unacceptable in certain applications.

The combined need for accuracy and speed in real-time simulations makes the collision detection problem very challenging. The time available to resolve the collisions may be less than 10 milliseconds, so highly efficient solutions are needed. Some fast search methods are available, when the involved bodies are convex [7, 8, 9, 10]. Concave objects can also benefit from these methods if they are decomposed into convex parts. For more general and complex rigid bodies, bounding volume hierarchies have often been found to be the best choice. Examples of types of bounding volumes (BVs) that have been used in such hierarchical data structures are spheres [11, 12], axis-aligned bounding boxes (AABBs) [13, 14], arbitrarily oriented bounding boxes (OBBs) [5], discrete orientation polytopes (k-DOPs) [15, 16], spherical shells [17], and convex pieces [18].

To check the collision status of two models, their bounding volume hierarchies are traversed in tandem while searching for intersecting primitive pairs. The performance of such a dual hierarchy traversal is dependent on the number of geometric primitives in the models, as well as the number of intersecting primitive pairs found during the traversal. For rigid bodies, a traversal is expected to be sub-linear in most cases, since the height of a hierarchy storing n primitives is expected to be proportional to $\log n$.

The choice of which bounding volume to use is not simple. To evaluate the performance of bounding volume hierarchies, it has been suggested that a cost function can be used [5, 15]. This function states that the cost, t, of a certain collision query is given by

$$t = n_v c_v + n_p c_p + n_u c_u \tag{2.1}$$

where n_v is the number of performed BV/BV intersection tests and c_v is the cost of one such test. Similarly, n_p is the number of geometric primitive pairs that are intersection-tested and c_p is the cost of one such intersection test. Finally, n_u is the number of BVs that are updated or recalculated because of model changes and c_u is the cost of updating one BV. By using tighter bounding volumes in the hierarchies, n_v, n_p, and n_u can be lowered, but on the other hand, tighter volumes often mean larger values of c_v and c_u. To minimize the cost function, one has to deal with such conflicting goals.

To further speed-up hierarchical collision detection methods, temporal coherence can be utilized in many types of simulations. By using different types of caching techniques, results from the previous simulation time step can be reused for faster determination of new results [19, 20, 18].

In the case of soft or deformable bodies, much work remains to be done [3]. Some interesting initial efforts aimed at different geometries and types of applications have been described. Examples of these include methods for higher order surfaces [21, 22, 23], triangle soups [24, 25] and cloth simulation [26, 27].

Some initial work has also been performed in the field of virtual surgery. One proposed method relies on graphics hardware to test the interpenetration of a deformable organ and a user-controlled rigid tool [28]. In a work on laparoscopic surgery, a special bucket data structure was used to store closely located polygons [29]. This data structure was then used to search for contacts between a simple tool and an organ represented by a polygonal mesh.

Despite these initial efforts, new and faster collision detection methods are needed to increase speed and realism in deformable body simulations. The algorithms proposed in this thesis are fast and accurate down to the finest resolution of the models. The methods are based on bounding volume hierarchies that can be updated efficiently as the models deform. Summaries of the approaches are given in the following sections together with a description of the main contributions.

2.2 Technical contribution of paper A

2.2.1 Summary

This paper suggests a new collision detection algorithm for multiple deforming bodies represented by polygon meshes. The bodies are allowed to undergo a complete change of shape from one time step to another by arbitrary vertex movements, but the topology or mesh structure must be kept the same. First, potential collisions are sorted out in a broad phase, which is done by a sweep and prune method which was originally suggested by Cohen et al. [9]. The result is a list of close body pairs that need to be examined more carefully. This is done in a narrow phase, where a hierarchical search is performed for each body pair in the list by using bounding volume hierarchies of axis-aligned boxes. When the

bodies deform, the boxes can be refitted very efficiently and they also make the needed BV/BV intersection tests very efficient.

The bounding volume trees are constructed in a pre-process before simulation time. A simple top-down tree building strategy is used in which the mesh is recursively split into new tree nodes until only one face remains. During simulation, the structure of the tree is kept fixed. When a body deforms, the axis-aligned bounding boxes in the nodes are refitted according to a scheme referred to as the hybrid bottom-up/top-down update method, which aims at finding an efficient balance between the number of updated tree nodes and the number of tree nodes encountered during collision traversals.

The performance of the proposed collision detection algorithm compares very favourably with other suggested approaches. In the experiments performed, it was found to be four to five times faster than a previously leading method. Deforming meshes of up to tens of thousands of geometric primitives were used in these experiments.

2.2.2 Main contributions

The main contributions of paper **A** are summarized below:

- A new collision detection algorithm is proposed for deforming meshes. All the vertices of the meshes can be arbitrarily deformed at every simulation time step. The algorithm is based on bounding volume hierarchies that can be pre-built and then efficiently updated during simulation.

- A novel hierarchy update method is presented. The upper levels of the hierarchies are updated bottom-up in an incremental way, whereas the lower levels of the hierarchies are updated lazily, as needed, during the collision traversals. This update method is significantly faster than previously suggested methods.

- By examining and comparing the performance of bounding volumes trees with k-ary tree nodes, where $k = 2$, 4, and 8, it was found that $k = 8$ gave slightly better performance in all conducted experiments. A higher value of k gives lower heights of the trees, but increases the work to be performed per node in the collision traversals.

– Two different ways of partitioning the faces of the meshes into the hierarchy nodes during tree construction are proposed. The first operates on the initial shape of the deforming body and the other on the interconnectivity of the faces of the mesh, which remains the same during simulation. Which is the better method depends on the model and how it is deformed.

2.3 Technical contribution of paper B

2.3.1 Summary

In this paper, a new algorithm for the rapid detection of collisions or intersections between morphing models is proposed, which further extends the usage of bounding volume hierarchies for collision detection. The morphing model is a polygonal mesh that is gradually transformed or blended between a set of reference meshes. All the possible deformations are bounded locally by the reference meshes, which make it possible to create a morphing-aware bounding volume hierarchy per morphing model that can be updated by transforming or blending sets of reference bounding volumes, which correspond to associated parts of the reference meshes.

The hierarchies are built in a preprocess before simulation and then their structures are never changed. Collision queries are performed by dual hierarchy traversals which sort out possible intersections in an efficient way. Whenever an outdated tree node is reached during a traversal, it is updated by a constant time bounding-volume blending operation. In practice, the performance of such queries is expected to be sub-linear, as in hierarchical collision detection methods for rigid bodies. Note that only for face pairs that are found to be located closely during the collision traversals, must the actual blended vertices of those faces be calculated. This means that for rendering, the vertices of the meshes can be blended by the graphics processing unit (GPU), which saves CPU time, and may improve the overall performance.

The expected high performance of the proposed algorithm has been verified by different types of experiments with morphing meshes defined by tens of thousands of triangular polygons. This algorithm for morphing models was found to be significantly faster than the more general collision detection method presented in paper **A**.

2.3.2 Main contributions

The main contributions of paper **B** are summarized below:

- A novel collision detection algorithm for morphing models is suggested. It is based on morphing-aware bounding volume hierarchies. The performance of this algorithm is of the same order as that for hierarchical rigid body collision detection.
- A simple hierarchy construction method is presented that creates a single hierarchy per morphing model with k-ary tree nodes, which store one bounding volume per reference model per tree node.
- In the experiments, $k = 8$ was found to be a slightly more efficient choice when compared with $k = 2$ and $k = 4$. Note that it is very common in other works on hierarchical collision detection for rigid bodies to suggest binary tree structures.
- A top-down update method is proposed, which is completely integrated with the collision traversals. Only the nodes that are encountered during the dual hierarchy traversals are updated, and a node is updated simply by blending the stored reference bounding volumes. This is a constant time operation since the number of volumes to blend is fixed.
- It is shown that the proposed method works for bounding volume hierarchies of axis-aligned bounding boxes, discrete oriented polytopes, and spheres.
- The proposed collision detection strategy applies generally to other types of bounded deformations for which a bounding volume refit function is known that is independent of the geometric primitives stored in the volume.

Chapter 3

Ray tracing of deformable meshes

3.1 Related work

Ray tracing is a classic image synthesis technique. It was introduced as early as 1968 by Appel as a shadow determination technique [30]. In 1980, Whitted published an article describing the basic recursive ray tracing algorithm, which extended the original algorithm to handle specular reflection and refraction [31]. This version of the algorithm is essentially the basic ray tracing method described in many computer graphics textbooks of today [32, 33]. Hundreds of articles on different aspects of the subject have been published and there are books entirely devoted to ray tracing [34, 35].

In ray tracing, images are generated by tracing rays of light backwards, from the eye though the pixels in the image plane. These rays are then recursively traced according to the rules that have been set up for their interaction with the three-dimensional scene while the color contributions are gathered.

While ray tracing is well-known for its ability to create stunning pictures, it seems to be equally well-known for its extremely high computational cost. For example, to generate a single image of 800x600 pixels, super-sampled with 9 primary rays per pixel, over 4 million primary rays need to be intersected with the geometry in the scene and

potentially more than 20 million secondary rays, i.e., shadow, reflections and refraction rays. Thus, given complex scenes, with millions of geometric primitives, the number of intersection calculations required is huge. This disadvantage of ray tracing caused it for many years to be considered as an offline rendering method only.

The tremendous recent improvements in computer technology, however, have begun to change the view of ray tracing as a rendering method which is too slow for interactive graphics applications. In interactive ray tracing, something like 5–30 generated images per second are needed. To be able to handle the computational burden, clever algorithms are needed to reduce the number of intersection computations and utilize the inherent parallelism of the ray tracing process.

Today, it has been shown that interactive ray tracing is possible, even on affordable PCs [36, 37, 38]. Fast ray tracing can also enable faster global illumination computations, since there are many global illumination algorithms that rely heavily on ray tracing [39]. Thus, ray tracing may play an important role in achieving a long term goal in computer graphics — physically correct simulation of light transport at interactive frame rates in complex and dynamic environments.

Most proposed ray tracing approaches, however, are dependent on acceleration data structures that are pre-built before simulation. Many of these methods rely on certain limiting assumptions, such as that it is only the camera which is animated, the rest of the scene remaining static. Others have permitted the presence of certain dynamic models in the scene, as a special case, but these have been assumed to be rigid bodies [36]. Recently, programmable shaders have also been used to implement ray tracing on commodity graphics hardware, but the suggested approach was found unsuitable for dynamic scenes [40].

To make ray tracing an interesting alternative for interactive graphics applications in general, dynamically changing scenes must be supported. This also includes scenes with complex deformable or flexible models. Very few data structures have been proposed for such scenes [41]. The acceleration algorithm proposed in paper **C** is based on bounding volume hierarchies that permit the models to deform over time. A summary of this approach for ray tracing of deforming bodies is described next together with its main contributions.

3.2 Technical contribution of paper C

3.2.1 Summary

In this paper, a new acceleration method for ray tracing of deforming meshes is presented. In complex and dynamically changing scenes, the reconstruction phase, responsible for updating the data structures as the simulation proceeds, is likely to become a major bottleneck. Furthermore, the ray tracing phase can be parallelized very efficiently, as compared with the reconstruction phase, for which it seems much harder to create successful parallel solutions. With the new approach, it is shown that by only updating the upper levels of the pre-built bounding volume hierarchies, the reconstruction phase can be made an order of magnitude faster. The remaining parts of the hierarchies are updated lazily as needed in the ray tracing phase. This approach saves computation and leads to significant speed-ups in many scenes.

The high performance of the approach has been verified in complex scenes. It has been demonstrated that scenes with hundreds of thousands of deforming and reflective triangles can be ray traced at interactive frame rates. Completely dynamic and interactive scenes are supported, since no a priori information of forthcoming deformations are used.

3.2.2 Main contributions

The main contributions of paper **C** are summarized below:

- The proposed solution extends the set of scenes that can be ray traced at interactive frame rates. It is shown that interactive ray tracing is possible for scenes with complex deforming meshes consisting of hundreds of thousands of geometric primitives.
- By using the hybrid bottom-up/top-down update method from paper **A**, the reconstruction phase runs an order of magnitude faster as compared with using bottom-up refitted hierarchies. This is important since the reconstruction phase risks becoming the bottleneck in complex dynamic scenes.
- The suggested update scheme takes advantage of the fact that the hierarchies do not need to be updated for the occluded parts of the scene, although they may be deforming. This yields significant speed-ups in the total rendering time of many complex scenes.

Chapter 4

Conclusions and future work

4.1 Summary

A new hierarchical collision detection algorithm for moving and deforming polygonal meshes was presented in paper **A**. It was demonstrated that this method can be used in real-time graphics simulations, even for meshes consisting of tens of thousands of triangles. The novel way of updating the hierarchies is significantly faster than previously suggested approaches, and it is efficient in both simple and hard collision detection cases. The algorithm works for meshes undergoing arbitrary vertex repositioning, and all the vertices of the meshes may be repositioned simultaneously in every simulation time step. The suggested way of updating the hierarchies is linear in the number of vertices of the meshes. These advantages of the method make it attractive for use in many different types of simulations of soft or elastic materials. For example, it has already been used to speed-up the collision detection process in a work on realistic cloth simulation [42].

In paper **B**, a new collision detection algorithm for morphing meshes was described. In this case, any bounding volume in the created hierarchies can be updated by a simple and extremely fast bounding volume blending operation. This makes it possible to update only those bounding volumes that are visited during the collision traversals. The algo-

rithm was used in a number of practical experiments and it was found to be significantly faster than the more general method for meshes undergoing arbitrary vertex deformations. In fact, its performance is expected to be sub-linear in the number of faces, just as in hierarchical collision detection for rigid bodies. Other types of deforming bodies, for which the deformations are bounded by some known function can also benefit from the same basic strategy of updating the bounding volumes in the nodes of the hierarchies top-down, as they are reached in the collision traversals.

If a hundred, or more, morphing models are included in a simulation, an efficient broad phase [43] must be added to the algorithm, whose purpose is to sort out all body pairs out of n bodies that are potentially capable of colliding. The sweep and prune method suggested by Cohen et al. for rigid body simulation seems to be particularly suitable for this [9]. For the morphing bodies, all that needs to be done to use this method is to first update the root nodes of the hierarchies by blending the reference bounding volumes they store, an extremely efficient operation.

Furthermore, since both suggested collision detection approaches are based on bounding volume hierarchies and rather similar hierarchy traversals, they are easily integrated to support simulation that includes both morphing meshes and meshes deformed by repositioning the vertices arbitrarily. Hierarchies of AABBs have also been shown to be an efficient alternative in the case of moving rigid bodies [14]. By using small variations on the AABB trees and adding dual tree collision traversals for all pairs of tree types, collisions between all combinations of the types of models mentioned can be efficiently tracked. The only intersection tests that must be supported between tree nodes for all these methods are the trivial AABB/AABB test and the OBB/OBB test.

As a possible extension, the integration of OBB trees in the collision detection framework would be straightforward and beneficial in certain applications [5]. Other types of bounding volumes trees can also be supported if efficient intersection tests between tree nodes can be provided. One possible example is the sphere tree. An efficient sphere/AABB overlap test method exists [44] and a sphere/OBB test can also use this test by first transforming the sphere into the coordinate system of the OBB. By permitting the use of different types of hierarchies, it is also possible to choose the most suitable for each model, which can improve the tightness and lead to better performance.

Finally, in paper **C**, a means of accelerating ray tracing of deforming

meshes was presented. This is of great importance for extending the set of scenes which can be used in interactive ray tracing. With the proposed way of updating the hierarchies, the reconstruction phase executes an order of magnitude faster in comparison with a complete bottom-up hierarchy update. This is important since the reconstruction phase might very well become a bottleneck, because it seems to be much harder to parallelize than the actual ray tracing phase. The experiments conducted also showed that a substantial improvement in the total rendering time can be expected for scenes in which some deforming models only partly contribute to the final image.

4.2 Directions for future work

Much further work remains to be performed in the field of deformable body simulation. Certain directions in which research relating to collision detection and interactive ray tracing is desirable are described briefly below:

- The extension of the proposed collision detection methods to support efficient handling of self-intersections would be beneficial. This is very important in certain applications. In cloth simulation, for example, some approaches have already been published [45, 46, 47].

- The proposed collision detection methods are not designed to handle situations where a single model is torn apart or cut into several separated parts. The generalization of the algorithms to support body cutting in an efficient way is a subject worth more attention, and could lead to algorithms useful in, for example, virtual surgery.

- For rigid bodies, a technique designated front tracking has been suggested to utilize the temporal coherence that is present in many types of simulations [20, 18]. It would be interesting to examine this technique further in the context of deformable body simulation.

- The extremely fast developments of new GPUs make it very interesting to study how they can be utilized to accelerate collision detection. Such an approach for deformable triangulated objects has been suggested recently [48]. Alternative approaches, which for

example take advantage of the programmability of modern GPUs, may be of interest for use in future applications.

- In the field of bioinformatics and computational biology new specialized collision detection algorithms are needed. The simulation of protein folding, for example, is expected to become very important.

- The design of a set of standardized benchmark scenes for collision detection of deformable bodies would make fair comparisons between algorithms much easier. Such scenes must be designed with care so that they include a broad spectrum of different and interesting contact scenarios.

- In interactive ray tracing, it would be interesting to study acceleration data structures for scenes with millions of independently deforming primitives, including parallelization of the reconstruction phase. In the method proposed in this thesis, the deforming models are limited to polygonal meshes undergoing arbitrary vertex deformation. The meshes were not allowed to be torn apart during simulation.

Bibliography

[1] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering, 2nd Edition*. A K Peters, 2002.

[2] M.C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA, Conference of Mathematics of Surfaces*, pages 602–608, 1998.

[3] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25:269–285, 2001.

[4] Carol O'Sullivan, John Dingliana, Fabio Ganovelli, and Gareth Bradshaw. T6: Collision handling for virtual environments. In *Eurographics 2001 Tutorial Proceedings*, 2001.

[5] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM Press, 1996.

[6] Zhigeng Pan, Weiwei Xu, Jin Huang, Mingmin Zhang, and Jiaoying Shi. Easybowling: a small bowling machine based on virtual simulation. *Computers & Graphics*, 27:231–238, 2003.

[7] E. G. Gillbert, D. W. Johnson, and 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.

[8] S. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of the Interna-*

tional Conference on Robotics and Automation, pages 3112–3117, 1997.

[9] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–196. ACM Press, 1995.

[10] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3):177–208, 1998.

[11] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, July 1996.

[12] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[13] G. Zachmann and W. Felger. The boxtree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *First Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.

[14] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[15] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[16] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 90–97, March 1998.

[17] 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*, pages 287–296, 1998.

[18] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–510, September 2001.

[19] James T. Klosowski. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments*. PhD thesis, State University of New York at Stony Brook, May 1998.

[20] Tsai-Yen Li and Jin-Shin Chen. Incremental 3D collision detection with hierarchical data structures. In *Proceedings of the ACM symposium on Virtual reality software and technology 1998*, pages 139–144. ACM Press, November 1998.

[21] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 39–48. ACM Press, 1990.

[22] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 19–28. ACM Press, 1990.

[23] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 303–308. ACM Press, 1992.

[24] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 136–145, 1995.

[25] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference in Computer Graphics (SCCG2000)*, pages 156–163, 2000.

[26] Pascal Volino, Martin Courchesne, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 137–144. ACM Press, 1995.

[27] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):261–267, 2001.

[28] J. Lombardo, M. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Proceedings of Computer Animation*, pages 33–39, 1999.

[29] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformation of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, 1999.

[30] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the spring joint computer conference*, pages 37–45, 1968.

[31] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[32] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice, 2nd edition.* Addison-Wesley, 1996.

[33] Alan Watt. *3D Computer Graphics, 3rd edition.* Addison-Wesley, 2000.

[34] A. Glassner. *An Introduction to Ray Tracing.* Academic Press, 1989.

[35] Peter Shirley. *Realistic Ray Tracing.* A K Peters, 2000.

[36] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 119–126. ACM Press, 1999.

[37] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. In *State of the art reports, Eurographics 2001*, September 2001.

[38] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[39] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive global illumination using fast ray tracing. In *Eurographics Workshop on Rendering*, 2002.

[40] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (TOG)*, 21(3):703–712, 2002.

[41] E. Reinhard, B. Smits, and C. Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 299–306, 2000.

[42] Kiran Bhat, Christopher Twigg, Jessica Hodgins, Pradeep Khosla, Zoran Popović, and Steven Seitz. Estimating cloth simulation parameters from video. In *Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.

[43] Philip M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[44] J. Arvo. A simple method for box-sphere intersection testing. In Paul Heckbert, editor, *Graphics Gems*, pages 247–250. Academic Press, 1990.

[45] P. Volino and Nadia Magnenat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

[46] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (TOG)*, 21(3):594–603, 2002.

[47] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics (TOG)*, 22(3):862–870, 2003.

[48] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32. Eurographics Association, 2003.

II

Included Papers

Chapter 5

Paper A: Collision Detection for Continuously Deforming Bodies

Thomas Larsson and Tomas Akenine-Möller
In Eurographics Conference 2001, Short presentations, pages 325–333, Manchester 2001

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.

5.1 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-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.

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

5.3 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 5.1. 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.

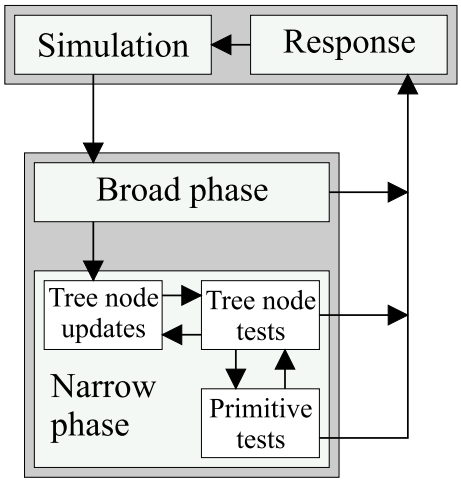


Figure 5.1: Schematic overview of the collision detection module for deforming bodies and its communication with an application specific simulation module.

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 varia-

tions, are described in more detail in the following sections.

5.3.1 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.

5.3.2 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 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.

5.3.3 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.

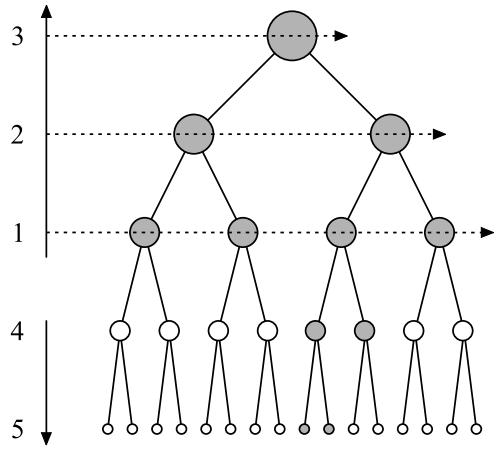


Figure 5.2: Example of a hybrid tree update method, combining the bottom-up and top-down strategy.

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 5.2 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.

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.

5.3.4 Multiple Body Simulation

For simulations with up to approximately 100 bodies, the naive 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.

5.4 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 5.3. In Figure 5.3a, the timings for reporting all intersecting triangle pairs are reported, and in Figure 5.3b the timings for finding a first arbitrary intersecting triangle

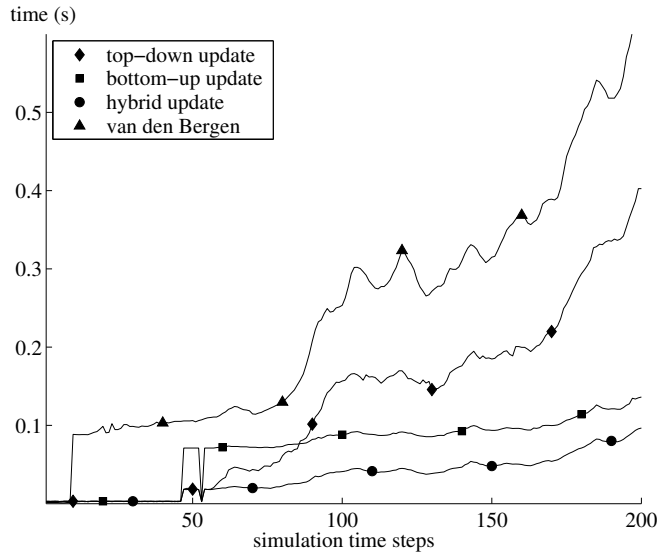
pair is reported, which in cases where there are a lot of intersecting triangles is much faster. Reporting only one triangle pair might be sufficient for many applications. For example, if we want to search for the exact time for the bodies' first contact, it is sufficient to find one intersecting triangle 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].

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 5.1 we can see the results. The last column reports the number of intersecting 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.

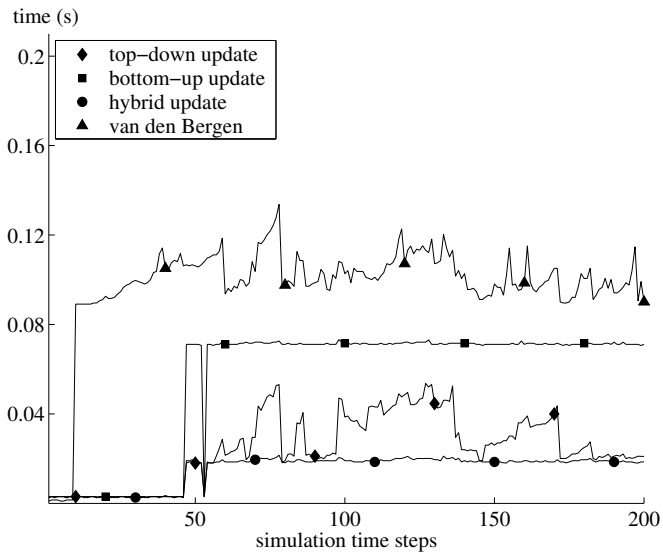
faces per body	first contact (ms)	worst contact (ms)	triangle intersections
1280	2	4	138
5120	6	15	562
8192	8	28	1010
16384	15	64	2422
20480	20	98	3760
32768	30	151	4775
65536	62	334	12358

Table 5.1: 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 5.4. Also, the performance of



(a)



(b)

Figure 5.3: 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.

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 5.4a, and a factor of 4.5 faster, in Figure 5.4b, 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 5.3a). Another difference is that 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 5.5 and 5.6, images of the types of bodies that were used in our experiments are shown. Animations showing the reported experiments have also been produced.

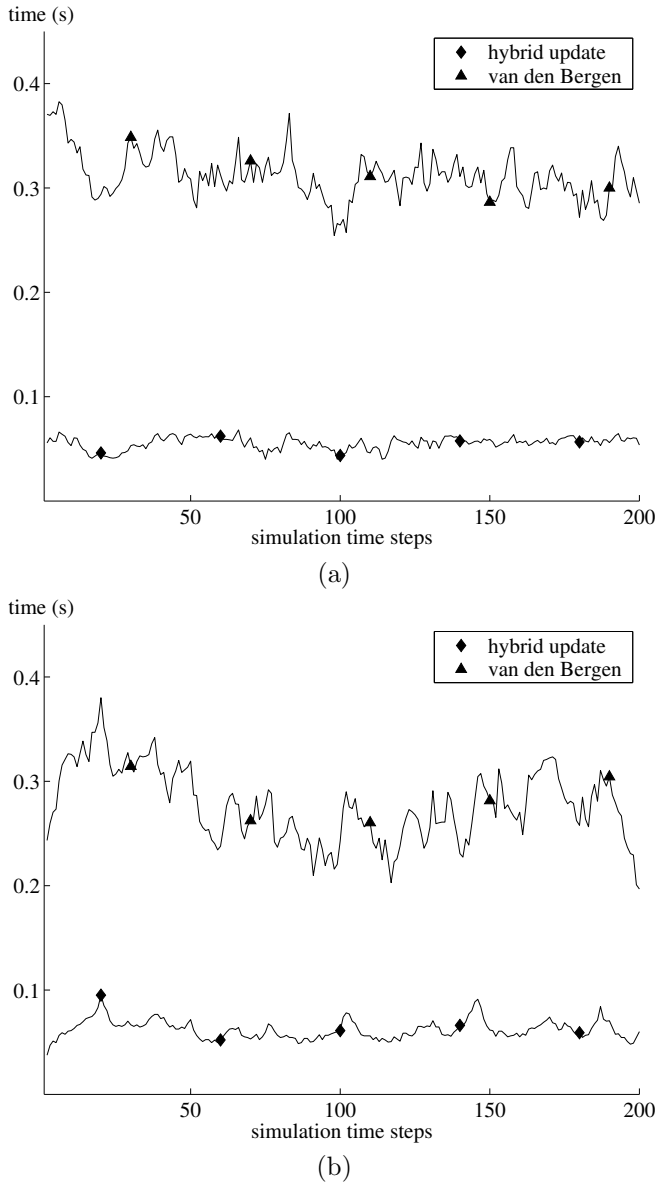


Figure 5.4: Collision detection performance per time step according to experiment 3 where a) 27 bumpy sphere bodies were used. b) 27 bodies with multiple arms were used.

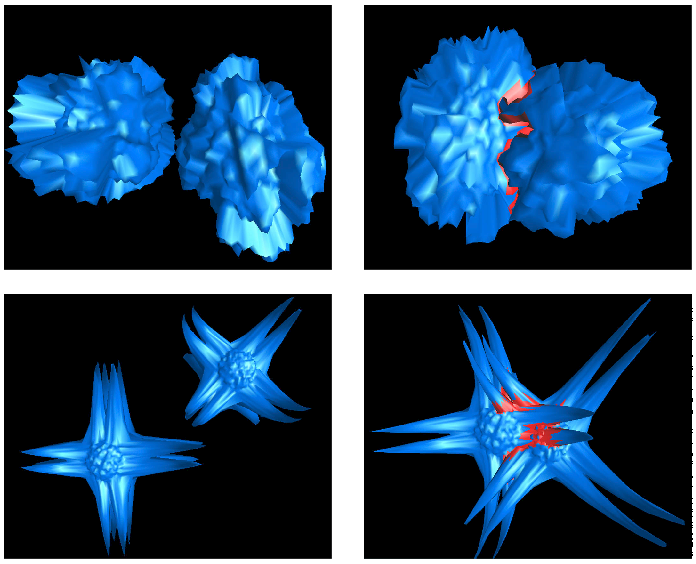


Figure 5.5: Some moving deforming bodies before and after they have interpenetrated each other. Intersecting triangles are in red colour.

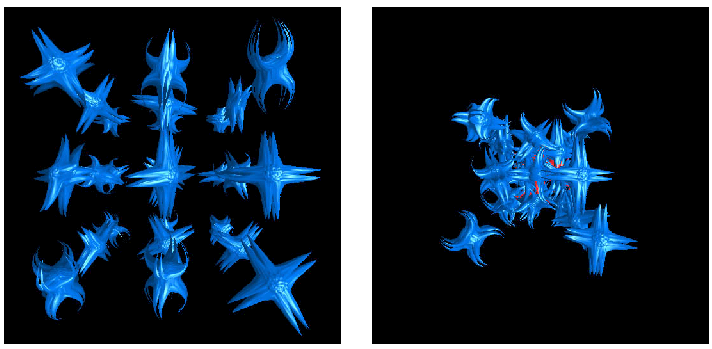


Figure 5.6: Multiple deforming bodies moving from a simple start case towards a common goal point in order to stress the collision detection algorithm.

5.5 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.

5.6 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 pre-build 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.

Bibliography

[1] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25:269–285, 2001.

[2] M.C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA, Conference of Mathematics of Surfaces*, pages 602–608, 1998.

[3] G. Turk. Interactive collision detection for molecular graphics. Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 1990.

[4] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–196. ACM Press, 1995.

[5] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 289–298. ACM Press, 1988.

[6] M. Held, J. T. Klosowski, and J. S. B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proceedings of the Seventh Canadian Conference on Computational Geometry*, pages 205–210, 1995.

[7] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 115–124. ACM Press, 1990.

[8] W. Bouma and Jr. G. Vanecek. Collision detection and analysis in a physical based simulation. In *Eurographics Workshop on Animation and Simulation*, pages 191–203, 1991.

[9] D. Kim, L.J. Guibas, and S. Shin. Fast collision detection among multiple moving spheres. *IEEE Transactions on Visualisation and Computer Graphics*, 4:230–242, 1998.

[10] Philip M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[11] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[12] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, July 1996.

[13] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[14] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM Press, 1996.

[15] G. Zachmann and W. Felger. The boxtree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *First Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.

[16] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[17] Taosong He. Fast collision detection using QuOSPO trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62. ACM Press, 1999.

[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*, pages 287–296, 1998.

[19] A. Garca-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 14(3):36–43, 1994.

[20] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3):177–208, 1998.

[21] E. G. Gillbert, D. W. Johnson, and 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.

[22] S. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of the International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[23] Gino van den Bergen. A fast robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[24] L. J. Guibas, D. Hsu, and 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.

[25] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Animation*, 6(3):291–302, 1990.

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

[27] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 136–145, 1995.

[28] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference in Computer Graphics (SCCG2000)*, pages 156–163, 2000.

[29] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[30] Gino van den Bergen. Software library for interference detection (SOLID), 1999.

[31] A. Joukhadar, A. Scheuer, and Ch. Laugier. Fast contact detection between moving deformable polyhedra. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 1810–1815, 1999.

[32] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 303–308. ACM Press, 1992.

[33] M. Hughes, M. Lin, D. Manocha, and C. Dimattia. Efficient and accurate interference detection for polynomial deformation. In *Proceedings of Computer Animation*, pages 155–166, 1996.

[34] Tomas Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[35] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58. ACM Press, 2001.

Chapter 6

Paper B: Efficient Collision Detection for Models Deformed by Morphing

Thomas Larsson and Tomas Akenine-Möller
The Visual Computer, 19(2-3):164-174, 2003

Abstract

We describe a fast and accurate collision-detection algorithm specialised for models deformed by morphing. The models considered are meshes where the vertex positions are convex combinations of sets of reference meshes. This new method is based on bounding-volume trees that are extended to support efficient tree-node updates by blending associated sets of reference bounding volumes. With our approach, it is possible to use either axis-aligned bounding boxes, discrete-orientation polytopes, or spheres as bounding volumes. The expected performance of our algorithm is of the same order as for rigid hierarchical collision detection. In our tested scenarios, the speed-up we achieved ranged from 1.5 to 58, compared to another more general algorithm for deforming bodies.

Key words: Collision detection – Morphing – Deformable bodies – Hierarchical data structures – Virtual reality

6.1 Introduction

Fast and accurate collision detection is an essential part in almost all graphics applications that involve the simulation of moving models or bodies. In this paper, we present a very efficient collision-detection algorithm specifically developed for models deformed by mesh morphing (also called *mesh inbetweening* or *blending*). This deformation technique has become very important in many computer graphics applications, because of its ability to model living material and objects in a simple and efficient way. In fact, morphing techniques have been used in various application areas, ranging from games, animation and special movie effects to industrial design and scientific visualisation. The efficiency of our algorithm makes simple morphing a feasible technique to use in real-time applications.

Recent advances in consumer graphics hardware have made certain deformation techniques, such as morphing, even more attractive, since the deformed meshes can be both deformed and rendered by the graphics processing unit (GPU) [1]. This means that we can expect more detailed and realistic deforming models to be used in graphics applications in the near future, which in turn calls for new or improved collision detection algorithms.

Our proposed algorithm is based on using pre-built bounding-volume trees. Some popular types of bounding volumes that can be used in these trees are axis-aligned bounding boxes (AABBs), discrete-orientation polytopes (k-DOPs), or spheres. We show how the bounding volumes can be conservatively updated using the same morphing function as used for the vertices. Since only the bounding volumes, and not the entire geometry in them, are updated, this operation is extremely fast. In addition, the bounding volumes are only updated lazily in a top-down manner, which further improves performance. This way of refitting the volumes is expected to work very well for all types of deformations where there is an efficient bounding-volume update function available that is not dependent on knowing the details of the model geometry inside the bounding volumes but still able to refit each volume reasonably tightly. Our methods do not require that the deformed local vertices of the models be calculated before the collision tests are done. Instead, the vertices are calculated sparsely, as they are needed, being when leaves are reached in our tree traversals. As expected, experiments have shown that our collision-detection solution is very efficient in practice.

The rest of this paper is organised in the following way. In the next section, related work is presented. Then, in Section 6.3 and 6.4, the details of our collision-detection method and the promising results we have achieved are discussed. In Section 6.5, further optimisation techniques are presented. Finally, our conclusions and some opportunities for future work are given.

6.2 Previous work

The collision-detection problem for bodies undergoing arbitrary deformation is an open research area [2]. Some initial efforts are described in the literature. For example, Smith et al. suggested a very general algorithm [3, 4] based on re-computing all bodies' AABBs in world space at every simulation time step. A face octree is then built on the fly, when it is necessary to sort out potential face intersections among close faces. The generality of this method results in performance problems that clearly can be avoided when more restricted types of deformations are used. There are also some other proposed methods for deformable models aimed at different types of geometry and situations. Some examples are methods for higher order surfaces [5, 6, 7] and for cloth simulation [8, 9, 10, 11].

The fastest methods available, however, are those that solve the collision-detection problem when all bodies are rigid. The most general methods for rigid bodies use precomputed hierarchical data structures, like different types of bounding-volumes trees. Some interesting collision-detection algorithms have used spheres [12, 13, 14, 15], axis-aligned bounding boxes [16, 17], oriented bounding boxes [18], discrete-orientation polytopes [19, 20], quantized orientation slabs with primary orientations [21], and convex surface decompositions [22] as their bounding volume of choice.

It has been shown that an approach using bounding-volume trees, similar to the ones used in the suggested methods for rigid bodies, also can be used for certain types of deformable models. For example, the benefits of prebuilding the hierarchical data structures might be used for many kinds of deforming models, primarily those that keep their face connectivity and are not torn apart. The problem is, of course, that the size of the bounding volumes in the trees must be recomputed or refitted in an efficient way as the models change their shape and

when they are needed in a collision test. Van den Bergen suggested this approach for AABB trees [16], and he described a way to refit the bounding volumes in the trees bottom-up, from the leaves up to the root, during simulation time. Larsson and Akenine-Möller described a more efficient tree update method for similar AABB trees, by combining the benefits of a bottom-up and a top-down update into a hybrid update method [23]. Both these methods can handle polygon models deformed by arbitrary vertex repositioning. If all vertices in a model are deformed arbitrarily at one time instant, this means that all of them have to be processed during some necessary bounding-volume refit operation, which gives the algorithms a time complexity with a linear lower bound in the number of vertices.

The above mentioned methods can be used for bodies undergoing specific types of deformation, for example morphing. Unnecessary computations can, however, be avoided, if the deformation scheme is known and the collision-detection algorithm can take advantage of that information. In the following sections, such a collision-detection strategy is discussed, primarily for models deformed by morphing. The general approach can, however, be used also for other types of bounded deformations.

6.3 Collision detection algorithm

To accelerate the collision-detection queries, we use AABB trees, defined in model space. We also show that spheres and k-DOPs can be used as bounding volumes in the hierarchies in the same manner. The models we consider are assumed to be built of triangles. Other types of models can be supported by first tessellating them into triangle primitives.

Our hierarchical data structures are built in a pre-processing pass. Bounding-volume hierarchies are often built by using a top-down [19], bottom-up [24] or an incremental-insertion heuristic [25]. It is unclear how to build optimal hierarchies in most cases, even when rigid bodies are used. How to initially build them, when the vertices of the models will be deformed later on, is an even more complicated problem. We have chosen a simple top-down approach, which builds a bounding-volume hierarchy by recursively splitting the geometry of the whole input model into sub-parts while building the tree data structure. The recursion proceeds until child nodes with only one face left are found and these

nodes form the leaves in the tree. We have found 8-ary tree nodes to be an efficient choice, giving slightly better performance than both 4-ary and binary tree nodes. The split rule we use is simply to subdivide along the three principal coordinate axes at the mid-value of the sub-model's extents along these axes. Then the midpoints of the triangles are used to determine which nodes they are assigned to. Since no triangles are split, certain overlaps between children's bounding volumes are necessary to cover the geometry. If any of the child nodes are empty, after the geometric primitives have been assigned, it is removed and the parent node becomes a k-ary node with $k < 8$.

During simulation, exact collision detection between two models is done by a dual tree traversal procedure that efficiently sorts out the close or colliding parts of the models. It starts by testing the overlap status of the root nodes. If no overlap is found, testing is complete. Otherwise, the traversal proceeds by descending to the children in the tree where the volume of the node's AABB is largest. The overlap status between these children and the other tree node is then determined. Whenever an overlapping node pair is found the tree traversal continues recursively. The whole tree traversal proceeds until the trees are separated or until all intersecting geometric primitives inside the leaf nodes have been found. The basic control flow of this collision traversal is well captured by a procedure given by Gottschalk [26]. To improve performance, the traversal can also be aborted as soon as the first intersecting primitive pair has been found, whenever this is sufficient for the application.

During the tree traversals, we use an OBB–OBB overlap test that is based on the separating-axis theorem (SAT) [18]. The reason for this is that our refitted AABBs are defined in model space, which means the required overlap test becomes an OBB–OBB test in world space. This test can be implemented very efficiently when AABB trees are used, since during a dual tree traversal all the boxes in the intersection tests have the same orientation relatively each other. Furthermore, we do not perform the full SAT test. Instead, we use a variation, which is commonly referred to as *SAT lite* [16]. This test gave a better overall performance, although, it sometimes inaccurately reports overlap, resulting in further intersections tests deeper down in the bounding-volume trees.

When we are dealing with models undergoing deformation, a refit operation must be carried out to adjust the bounding volumes in the tree nodes before their overlap status can be determined. This means that, given a specific kind of operation deforming the original vertices of

a model, we have to provide an associated update operation that is able to refit the bounding volume of the original geometry into a bounding volume of the deformed model. To be really useful, the update operation has to be extremely efficient, preferably an operation that runs in constant time per node, no matter what the number of geometric primitives in the underlying model. This gives the collision-detection algorithm the possibility of having sub-linear performance in the number of involved faces in collision queries. The performance behaviour of the collision-detection traversal is expected to be comparable with collision-detection methods for rigid bodies that are using the same type of bounding volume tree traversals. In the following sections, fast refitting of bounding-volumes trees is described and evaluated for models deformed by morphing.

6.3.1 Morphing models

Mesh morphing refers to the process of transforming a shape into another shape in a smooth way. This is generally done by first establishing the correspondence between geometric parts in a source model and a target model and then interpolating between these parts to produce the inbetween meshes. The source and target models are also called the *reference* models of the morph. More than two reference models can also be used to define a morphing shape. In this case the inbetween meshes are defined in a space of n shapes, which might be very useful, for example, when two reference models are not enough to describe the needed key poses of a model.

We define our morphing objects in such a way that the reference models are created from the same original mesh structure; that is, they all have the same number of vertices and mesh connectivity. In many applications, the morphing models are defined in such a simple and convenient way. The reference models are key poses of the "same model" that are suitable to blend between. The morphing is often done simply by linearly interpolating corresponding vertex pairs in the reference models [27].

To be able to handle collisions among the models efficiently, the idea is that during a collision traversal the bounding volume of the nodes that are visited can be updated simply by blending the right bounding volumes created from corresponding parts of the reference models. In this way, we can avoid calculating the blended vertices of the inbetween

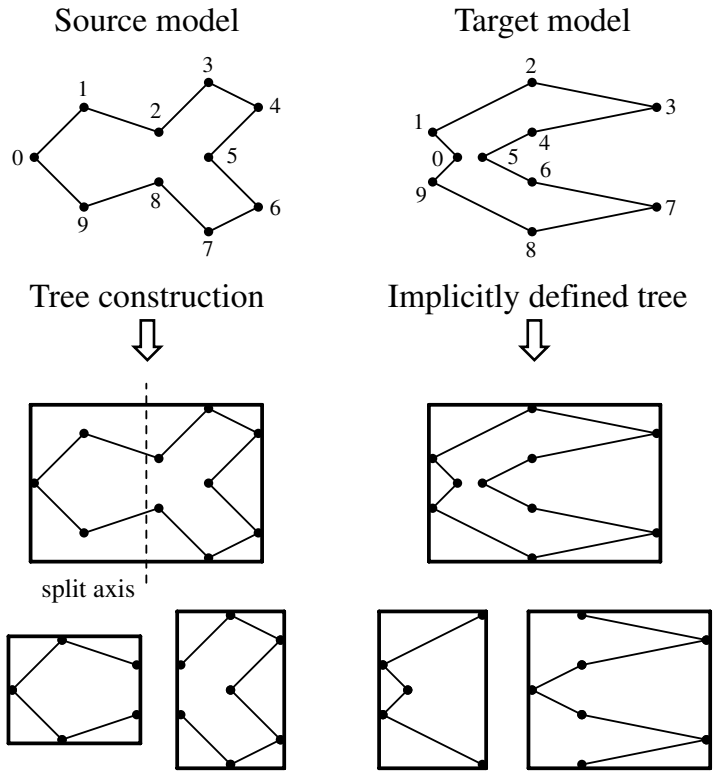


Figure 6.1: Creating a morphing-aware bounding volume hierarchy. A source and target model and their vertex mapping, defined by the corresponding vertex labels, are shown at the *top*. Only the mesh of the source model is used to partition the geometric primitives into a bounding-volume tree, which is illustrated at the *bottom left* where the creation of the first two levels in the tree is shown. The corresponding hierarchy of the target model is defined implicitly by the tree of the source model together with the vertex mapping between the source and the target models as shown at the *bottom right*. Note the resulting overlap between the two children's boxes

meshes, as well as processing them to update the bounding volumes they reside in. Note that this quality of our method fits very well together with modern graphics hardware where the blending of the vertices in the morphing models can be done by the GPU, thus freeing the CPU from doing this burdensome task at each frame. To make our strategy possible, a morphing-aware bounding-volume tree is prebuilt.

The tree building is straightforward given that the reference models are defined as described above. We can simply build a tree structure as we have described earlier, using one of the reference models (or one inbetween mesh) to determine the geometry partitioning into tree nodes. Then we assume that this tree structure is a relatively good structure also for the other reference models, and we add one bounding volume per node in the tree for every reference model in such a way that all the bounding volumes in a tree node are associated with the related geometric sub-models that are blended during morphing. An illustration of this tree building procedure for two reference models in 2D is given in Figure 6.1. When a morphing model is defined by n reference models each tree node has to store n bounding volumes, and each leaf node must also store n indices to the reference faces associated to it. Which mesh is best to use for the geometry partitioning is hard to say. The most frequent or average mesh in the morph might be a good choice. A potential drawback of our tree-building approach is that there will often be more overlaps between the implicitly defined bounding-volume hierarchies than would be the case if these hierarchies were built from scratch. This type of hierarchy is, however, motivated by the very fast updating of the hierarchies they provide. For all the morphing models we have used, the hierarchies have been found to give very good performance.

If more incompatible reference models are to be used, extra care has to be taken to make sure that there is a strict correspondence between the underlying geometry in the bounding volumes that will be blended to refit the hierarchy during simulation. The tree-construction phase then becomes a more challenging task. One solution is to replace the incompatible reference models in a preprocessing operation with suitable models having the same shape as the original models [28].

If n reference models are used to define the morphing model, each vertex of the morphing model is defined as a convex combination of its n associtated vertices, one from each one of the reference models. For each time instant during the morph, there is a unique set of weights, denoted w, that determines the influence each of the corresponding n reference

models will have on the final morphed shape. These weights are defined as n scalars such that

$$\sum_{i=1}^{n} w_i = 1 \qquad \text{where each } w_i \in [0,1]. \tag{6.1}$$

Any vertex p_m of the morphing body can then be defined as a convex combination of its corresponding reference vertices p_{im} and weights w_i, that is,

$$p_m = \sum_{i=1}^{n} p_{im} w_i. \tag{6.2}$$

In the following two subsections, we will show that both k-DOPs and spheres can be morphed with the same morphing function as used on the vertices. This allows for extremely fast and proper updating of the bounding volumes in the hierarchical data structures during the collision traversals. Note that AABBs also can be morphed since the AABB is a special case of the k-DOP. Bounding volume trees of OBBs, on the other hand, cannot be blended or morphed in the way we propose, unless all the OBBs to be blended in the tree nodes have the same relative orientation, which defeats the purpose of OBBs [18].

6.3.2 Blending k-DOPs

If we use k-DOPs as our bounding volume of choice, the reference k-DOPs of the models can be denoted $B_i = (s_i, e_i)$, where the different reference k-DOPs are subscripted by i, and s_i and e_i are $k/2$-tuples (note that k is always an even number) that define the start and end values of the extents of the k-DOPs along its $k/2$ slabs or 1D intervals. The directions of these intervals are along the corresponding normals of the k-DOPs, denoted by $n = (n_1, n_2, ..., n_{k/2})$. These normals are normalised, and they are the same for all reference models. The blended k-DOP, which we call B', is then calculated in the following simple way:

$$B' = (s', e') = \left(\sum_{i=1}^{n} s_i w_i, \sum_{i=1}^{n} e_i w_i \right). \tag{6.3}$$

We will show that the blended vertices will always stay inside the blended intervals along the n_i directions defining the k-DOPs. An individual slab in B_i is denoted by $B_{ij} = (s_{ij}, e_{ij})$, where $0 \leq j \leq k/2$.

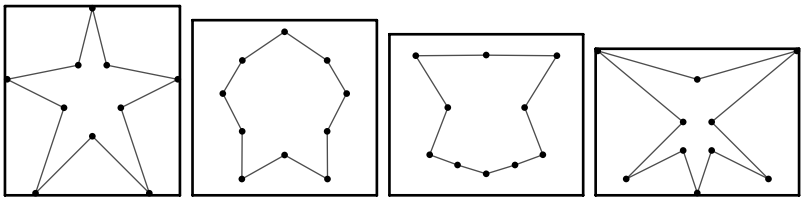


Figure 6.2: A morphing star polygon and its AABB in 2D. The source model and target model are shown to the *left* and *right* respectively, and two of the inbetween models are shown between them. Although not optimal, the blended AABB always covers the morphing model. How large the excess volume of the blended bounding volume is, compared to the corresponding tight bounding volume, depends on the model

Then, for each reference k-DOP B_i, and each slab j in it, we already know by definition that each vertex p_{im} is within the interval, that is,

$$s_{ij} \leq p_{im} \cdot n_i \leq e_{ij}. \tag{6.4}$$

Note that $p_{im} \cdot n_i$ gives the signed distance from the origin to the orthogonal projection of p_{im} onto the vector n_i, since n_i is normalised. By multiplying with w_i, it follows that

$$s_{ij} w_i \leq (p_{im} \cdot n_i) w_i \leq e_{ij} w_i, \tag{6.5}$$

since the weights always are non-negative. Hence, as expected, the terms from the different reference boxes can be summed to complete the blending and the following requirements hold:

$$\sum_{i=1}^{n} s_{ij} w_i \leq \sum_{i=1}^{n} (p_{im} \cdot n_i) w_i \leq \sum_{i=1}^{n} e_{ij} w_i. \tag{6.6}$$

Equation 6.6 guarantees that the blended k-DOP is a conservative estimate of the bounding volume of the blended geometry at any time instant during the morph. This means that the blended k-DOP always will contain the blended geometry.

When a model is defined by using only two reference models, the morphing is done by linearly interpolating the corresponding vertex pairs in the reference models. For example, to blend two corresponding AABBs

(which also are 6-DOPs) in a bounding-volume tree, we simply interpolate their extents linearly, using the same interpolation parameter as for the morph of the model. This is illustrated in 2D in Figure 6.2. Clearly, this is an extremely efficient update or refit operation.

6.3.3 Blending spheres

Another common bounding volume that can be used for morphing models is the sphere. In that case, there exists n reference spheres, here denoted $B_i = (c_i, r_i)$, where c_i defines the centre position and r_i the radius. At all stages during a morph, the reference spheres can be blended into a valid bounding sphere B' with centre point c' and radius r' in the following way:

$$B' = (c', r') = \left(\sum_{i=1}^{n} c_i w_i, \sum_{i=1}^{n} r_i w_i \right). \tag{6.7}$$

This can be shown as follows. From the reference models and their associated bounding spheres, we know that the length of each vector, $p_{im} - c_i$, for any vertex p_{im} in reference model i, compares to the radius r_i as follows:

$$\|p_{im} - c_i\| \leq r_i. \tag{6.8}$$

This also holds when all the terms in Equation 6.8 are multiplied with w_i (since $w_i \geq 0$); that is,

$$\|p_{im} w_i - c_i w_i\| = \|p_{im} - c_i\| w_i \leq r_i w_i. \tag{6.9}$$

Then the sum of the lengths of all scaled $p - c_i$ vectors must also be shorter than the sum of the corresponding radii, which gives

$$\sum_{i=1}^{n} \|p_{im} - c_i\| w_i \leq \sum_{i=1}^{n} r_i w_i. \tag{6.10}$$

Equation 6.10 shows, as expected, that the blended sphere will always contain the blended geometry.

Spheres might be a strong alternative to AABBs and k-DOPs in certain cases depending on the shapes of the morphing models. The proposed refit operation is slightly faster for spheres than for AABBs, and also the sphere–sphere overlap test is extremely fast.

6.4 Results

To verify the expected high performance of our method in practice, we have done many experiments. In this section we report the results from two of them that we have found to reveal characteristic behaviour of the algorithm. We ran these simulations using a standard PC computer with a 1333 MHz AMD CPU. In all the reported cases, we used our morphing-aware AABB trees as described in Section 6.3.

In the first experiment, we used two morphing models, where both of them were defined by three reference models, each one of them having 20480 triangles. The three reference models used can be seen in Figure 6.3. However, only two of them are used at the same time step; that is, the morphing model first transforms from the start model to the middle model, and then from the middle model to the end model. In order to stress the collision-detection algorithm, the two morphing bodies, which initially were positioned in a non-penetrating situation, moved during 400 simulation time steps in such a way that they passed right through each other. In Figure 6.4, images from some of the simulation time steps are given.

The time to determine the intersecting primitives in each time step in this experiment is reported in Figure 6.5, both when all intersecting triangle pairs were reported and when only the first-found intersecting triangle pair was reported. The timings are also given for the algorithm we used for comparison [23], here simply called the hybrid method, which is an efficient collision-detection method for bodies deformed by arbitrary vertex repositioning each frame. In the case when all intersecting triangle pairs between the models were reported, we can see that the morph method performs better or approximately the same, during time steps 1 to 90 and 270 to 400. The first intersection occurs at time step 41 and the last intersection at time step 364. At most of the time steps between 90 and 270 the models are at deep interpenetration, a situation that is not expected to arise in practice, since, in real applications, a collision-response mechanism is expected to prevent it. When only the first-found intersecting triangle pair was reported at each time step, the morph method is clearly superior at all time steps.

In the second experiment, twelve morphing dolphin models were used. Each dolphin was defined by three reference models, which are shown in Figure 6.6. In our scenario, the dolphins swam closely together as a group in order to generate interesting intersections. Besides the ongoing

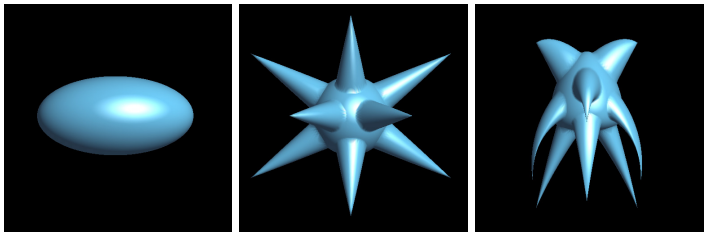


Figure 6.3: The three reference models used to define the morphing models in the first experiment

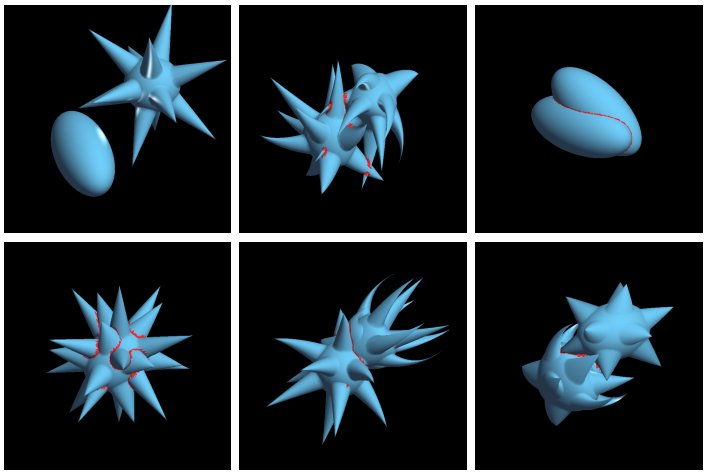


Figure 6.4: Images from the first experiment. The two morphing bodies are passing through each other while all the intersections are calculated. Intersecting triangles are in *red* colour. The images shown are from time steps 60, 120, 180, 220, 260, and 320

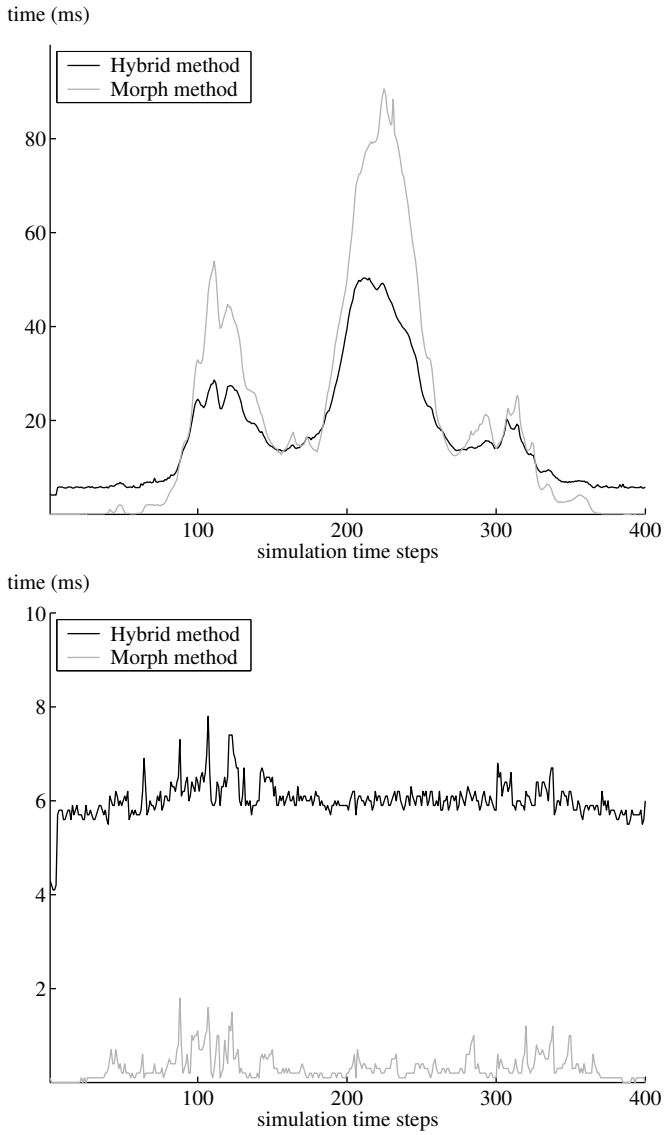


Figure 6.5: Measured performance of the collision-detection algorithm in the first experiment when reporting all intersecting triangle pairs (*top*), and when only reporting the first-found triangle pair (*bottom*)

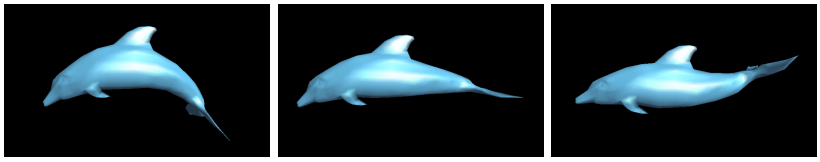


Figure 6.6: The three reference models used to define the morphing dolphin models in the second experiment

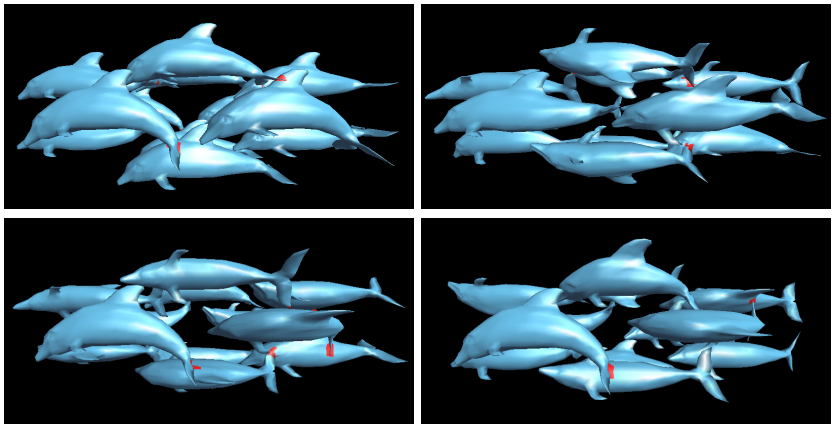


Figure 6.7: Images of four frames from the second experiment. Among the moving and morphing dolphins many different intersections occur during the simulation. Intersecting triangles are in *red*

morphing of the dolphins, they were also rolling, again to get more interesting intersections between individuals in the group. The simulation was executed for a period of 400 time steps. In Figure 6.7, images from some of the frames in this simulations are presented.

This simulation was repeated with varying triangle counts for the dolphin models, but we ensured that the shape of these different models were the same. In these repetitions, all the dolphin models had 564, 2256, 9024 and 36096 triangles respectively. The results of the simulations are reported in Figure 6.8. Results are given both when all intersecting triangle pairs were reported and when only the first-found intersecting triangle pair was reported at every time step. The time reported for each triangle count is the average time per frame that the collision detection algorithm spent during the simulation. As can be seen, the achieved performance for the morph method is superior to that of the hybrid method. As expected in this test case, with no deep interpenetrations or large areas in close proximity, the performance of the morph method was sub-linear, whereas the performance of the hybrid method was linear, in the number of faces of the models. The speed-up we achieved by using the morph method compared to the hybrid method ranged from 1.5 to 9 when all intersecting triangle pairs were reported and from 2 to 58 when only the first-found intersecting triangle pair was reported.

6.5 Optimisations

During each frame, some of the bounding volumes in the trees will be processed more than once during the tree traversals. It is, however, not necessary to refit a bounding volume more than the first time it is used. By adding a variable in the tree nodes that keeps track of which frame the nodes were last updated, we can avoid reupdating nodes during the same frame. When we reran our experiment using this technique, and searched for all intersecting triangle pairs, the achieved overall speed-up was approximately 15% in the first experiment and 10% in the second.

When only two reference models are used at the same time instant during morphing, there is room for some further optimisations. Generally, the blending operation we use to refit the bounding volumes does not produce optimal tightness, which can be seen by the simple example in Figure 6.2. We can improve the tightness of our blended boxes by inserting new reference bounding volumes between the ones produced

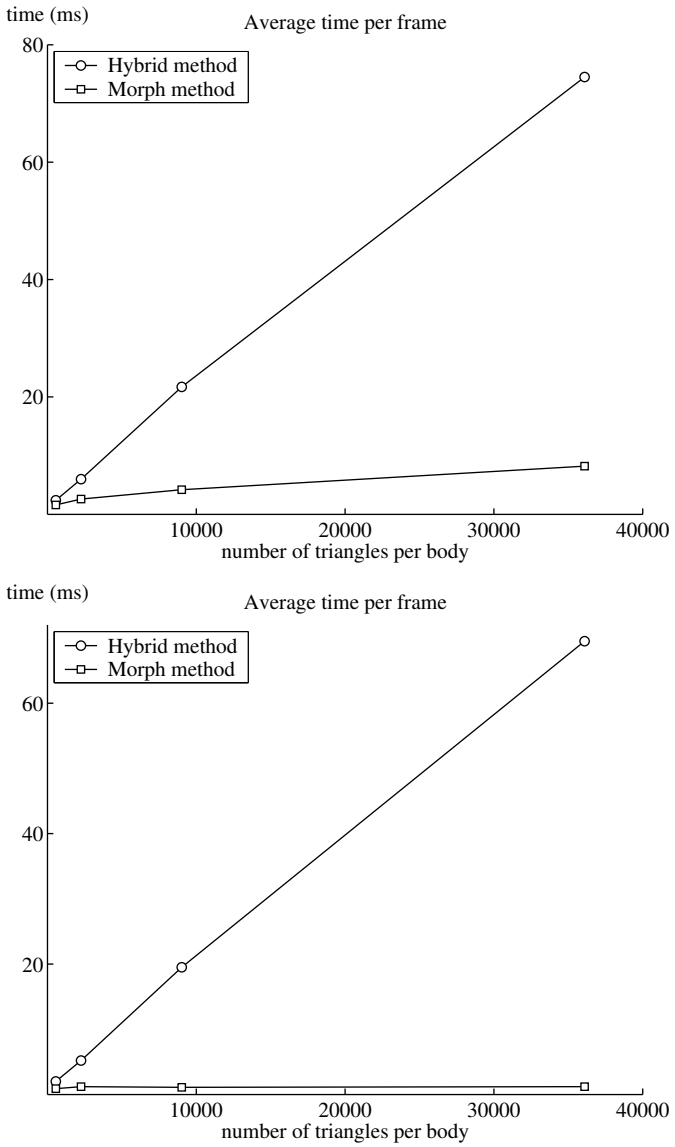


Figure 6.8: Results from the second experiment when reporting all intersecting triangle pairs (*top*) and when only reporting the first-found intersecting triangle pair (*bottom*) for all intersecting dolphin pairs in the scene

from the reference models in the tree nodes. This can be done in a preprocess before simulation. Whether this is needed or not depends on the degree of tightness achieved for the bounding volumes along the morph parameter interval, which is model dependent. It is also appropriate to consider if the speed-up we can get is worth the extra memory cost of storing extra bounding volumes in the tree nodes. In the extreme case, there would be precomputed bounding-volume hierarchies for all intermediate morph steps, which clearly would be an unreasonable approach in terms of memory usage.

We have tried this approach in our experiments by adding one extra bounding box between all reference boxes in every tree node. For example, when we reran the scenario that was chosen for the first experiment, and searched for all intersecting triangle pairs, the overall speed-up we achieved was approximately 5%. However, during some of the time steps, where the tightness of the boxes were improved the most, the resulting speed-up was up to 40%.

6.6 Conclusions and future work

In this paper, a fast and accurate collision-detection algorithm for morphing models has been presented. The method is based on bounding-volume trees where each node covers the corresponding locations in the reference models used for the morph. In this way, an efficient update function that blends the reference bounding volumes can be applied to update the tree nodes, as they are needed during collision tests. We have demonstrated the usefulness of our method in a number of experiments. The measured performance is very promising, and it is significantly faster than the performance of the more general method [23].

The same strategy, as the one we have used for collision detection between morphing models, might also be used for other types of deforming bodies. A requirement is of course that an efficient refit operation for the bounding volumes in tree nodes can be supplied, which is not directly dependent on the geometric primitives inside the node's bounding volume. Our method is also easy to integrate with other commonly used collision-detection methods using bounding-volume trees in order to support simulations including different types of models.

There are several possibilities for future investigations related to the work we have presented. Front tracking [22, 29] has been suggested

as a speed-up technique utilising frame-to-frame coherence for collision queries with bounding-volume hierarchies in rigid-body simulations. This technique might be beneficial in methods for deformable models as well. We expect the frame-to-frame coherence to be almost as high for many types of deforming models as it is in the rigid-body case, and it would be interesting to examine this technique further.

Another area worth more attention is how to generalise the collision-detection method between morphing models to more general morphing schemes. We have restricted our method to support morphing between models that possess the same mesh structure. There is ongoing research about morphing between 3D models with different surface topology and different model representations. From 1998, there is a survey by Lazarus and Verroust [30] describing various approaches. More recently, an overview of available methods in the area of mesh morphing has also been published [31].

Bibliography

[1] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 149–158. ACM Press, 2001.

[2] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25:269–285, 2001.

[3] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 136–145, 1995.

[4] Yoshifumi Kitamura, Andrew Smith, Haruo Takemura, and Fumio Koshino. A real-time algorithm for accurate collision detection for deformable polyhedral objects. *Presence*, 7:36–52, 1998.

[5] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 39–48. ACM Press, 1990.

[6] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 19–28. ACM Press, 1990.

[7] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proceedings of the 19th annual con-*

ference on Computer graphics and interactive techniques, pages 303–308. ACM Press, 1992.

[8] P. Volino and Nadia Magnenat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

[9] Pascal Volino, Martin Courchesne, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 137–144. ACM Press, 1995.

[10] Jen-Duo Liu, Ming-Tat Ko, and Reui-Chuan Chang. Collision avoidance in cloth animation. *The Visual Computer*, 12:234–243, 1996.

[11] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):261–267, 2001.

[12] Philip M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[13] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[14] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, July 1996.

[15] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[16] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[17] G. Zachmann and W. Felger. The boxtree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *First Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.

[18] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM Press, 1996.

[19] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[20] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 90–97, March 1998.

[21] Taosong He. Fast collision detection using QuOSPO trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62. ACM Press, 1999.

[22] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–510, September 2001.

[23] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics Conference 2001, Short presentations*, pages 325–333, September 2001.

[24] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, August 1996.

[25] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[26] S. Gottschalk. *Collision Queries using Oriented Bounding Boxes.* PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2000.

[27] M. Zöckler, D. Stalling, and H. Hege. Fast and intuitive generation of geometric shape transitions. *The Visual Computer*, 16:241–253, 2000.

[28] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 47–54. ACM Press, 1992.

[29] Tsai-Yen Li and Jin-Shin Chen. Incremental 3D collision detection with hierarchical data structures. In *Proceedings of the ACM symposium on Virtual reality software and technology 1998*, pages 139–144. ACM Press, November 1998.

[30] F. Lazarus and A. Verroust. Three-dimensional metamorphosis: A survey. *The Visual Computer*, 14:373–389, 1998.

[31] M. Alexa. Mesh morphing. In *State of the art reports, Eurographics 2001*, pages 1–20, September 2001.

Chapter 7

Paper C: Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models

Thomas Larsson and Tomas Akenine-Möller
MRTC Report, Mälardalen University, February 2003
(Submitted for publication)

Abstract

In this paper, we describe strategies for bounding volume hierarchy updates for ray tracing of deformable models. By using pre-built hierarchy structures and a lazy evaluation technique for updating the bounding volumes, the hierarchy reconstruction can be made very efficiently. Experiments show that for deforming triangle meshes the reconstruction time of the bounding volume hierarchies per frame can be reduced by an order of magnitude compared to previous approaches, which also results in a significant speed-up in the total rendering time for many types of dynamically changing scenes. We believe our approach is a step towards interactive ray tracing of scenes where moving objects can be dynamically changed in non-deterministic ways.

Key words: Ray tracing – Deformable models – Hierarchical data structures – Interactive simulation

7.1 Introduction

Ray tracing is a well-known rendering method that generates high quality images of virtual scenes. Shadows as well as lighting effects arising in scenes with reflective and refractive objects are produced with high realism [1]. The very high computational cost involved in ray tracing, however, has limited its usefulness to offline renderings, but due to computer technology advances, together with acceleration algorithms improvements, this has started to change.

Today, interactive ray tracing systems exist, but the efficiency of the rendering computations relies heavily on precalculated acceleration data structures. Thus, the scenes are often restricted to scenes where only the camera is animated and the geometry is assumed to be static, with the possible exception of a few moving rigid objects [2].

In this work, our purpose is to extend the set of scenes that can be ray traced at interactive rates. We have focused on ray tracing highly dynamic worlds consisting mostly of freely moving deformable models. To make interactive ray tracing possible under such circumstances, there are two main phases that have to be solved; these are the reconstruction phase, which make sure the acceleration data structures are up-to-date, followed by the actual ray tracing process. Both phases must run fast enough for interactive frame rates. In complex dynamic scenes, however, the reconstruction phase is likely to become the bottleneck that destroys the interactive performance. In fact, the ray tracing phase can be made in $O(\log n)$ time per pixel in the worst case and practical heuristic ray shooting methods have been found to have $O(1)$ time complexity in the average case [3]. This means that the reconstruction phase will eventually be the bottleneck as the scene complexity grows, given that it has an asymptotically worse time complexity.

Furthermore, it has been shown that the ray tracing phase is highly suitable for parallelization; it is often referred to as "embarrassingly parallel". Almost linear speed-ups for approximately 64 – 128 processors have been demonstrated [2]. On the other hand, we have not been able to find any published results on parallelization of the reconstruction phase, probably because it is much harder in that case to realize a successful parallel solution. Amdahl's law says that performance will be limited by the part of the program that is not parallelized [4]. In our case, according to Amdahl's law, the total rendering time $T(c)$ resulting from using c

processors in the ray tracing phase, t_{rt}, can be described by

$$T(c) = t_{rc} + \frac{1}{c} \cdot t_{rt}. \tag{7.1}$$

Thus, the reconstruction phase, t_{rc}, limits the performance improvements as c grows.

To improve this situation, we present strategies for efficiently refitting or reconstructing the bounding volume hierarchies when models are deformed during simulation. Compared to completely rebuilding the hierarchies each frame, our update strategy has been found to be orders of magnitudes faster in terms of reconstruction time.

In our solution, we use pre-built hierarchies together with a hybrid bottom-up/top-down update scheme to refit the bounding volumes in these hierarchies during simulation. Primarily, our approach has been found successful for models where the vertices of the models are allowed to be arbitrarily repositioned during simulation, but the meshes are not torn apart, i.e., the connectivity is static. In the hybrid update method, all the bounding volumes in a middle level of the hierarchy are refitted first and then the volumes above are refitted bottom-up incrementally from that level. In this way, the deeper levels are left as they are, until they actually are needed in some later ray/tree traversal. Thus, our update method is a kind of lazy evaluation technique, where the lower levels are not updated until it is necessary. Our method has already been used with successful results in collision detection [5]. It runs in $O(n)$ time for n deforming primitives.

For highly deforming polygon soups, where each polygon can be deformed completely independently of the other polygons, a different approach is generally required. But in our experiments, we illustrate how our method can be applicable even for deforming polygon soups with independently moving primitives under certain circumstances.

Like others have done recently, we only consider models defined by triangle primitives [6]. Supporting other types of geometry can be done through tessellation [7] or, when possible, by adapting the algorithm to ensure proper updates of the data structures when the models deform.

In the following section, related work is presented briefly. Then we explain our approaches for efficient reconstruction of the acceleration data structures. We present performance measurements from different scenes and discuss our methods' applicability under different circumstances. Finally, we present our conclusions and directions for future

work.

7.2 Previous work

Due to the very high computational cost involved, ray tracing research has mainly been focused on accelerating the creation of single images [8]. Some approaches have also been proposed to accelerate the creation of animated sequences. For example, Glassner transforms the problem of rendering moving three-dimensional objects into rendering static four-dimensional objects in space-time [9]. This method cannot be used in non-deterministic environments since the objects' space-time bounds must be known in advance.

Recently, however, it has been shown that interactive ray tracing is possible. Promising performance has been achieved mainly by utilizing multiple CPUs and/or SIMD instructions sets on today's computers [10, 2, 6, 11]. These solutions are rather limited to walk-through applications, i.e., applications where only the camera is animated, but not the objects in the scene. In some cases, a few dynamic rigid objects can be handled separately, as a special case [2]. An acceleration data structure allowing scenes of dynamically moving models with rigid parts, like for example walking robots, has also been proposed [12]. In this work, pre-constructed hierarchies of oriented bounding boxes were used, where the boxes themselves contained uniform grids. During animation, only the transforms associated with the grids contained in boxes need to be updated and then the rays need to be transformed into the local coordinate systems of these data structures. A similar approach has also been chosen by Wald et al. [13].

Ray tracing of dynamic scenes which allow deformations has also become an active research area. For example, objects undergoing unstructured motion have been handled by rebuilding the acceleration data structures each frame. This approach, however, immediately destroyed the interactive frame rates for a single complex model in a benchmark scene [13]. Reinhard et al. use a logically replicated grid over space for ray tracing dynamic scenes. Moving objects can be inserted and deleted in $O(1)$ time [14]. However, it might become necessary to rebuild the acceleration data structure during simulation once in a while, depending on how the objects move.

A hardware architecture for real-time ray tracing has also been pre-

sented by Scmittler et al. [15]. Impressive performance was achieved for camera animated scenes with otherwise static geometry, but no support for dynamically changing scenes was described. Purcell et al. implement ray tracing using commodity graphics hardware with programmable shaders [16], but none of the studied architectures was found to be suitable for accelerating ray tracing of dynamic scenes.

None of the earlier mentioned approaches seems to be suitable for truly interactive scenes inhabited by complex deformable models. In the following sections, we describe our approach for highly dynamic scenes, where all the vertices of the models are allowed to be arbitrarily repositioned each frame of the animation. Our approach builds upon our earlier work on efficient collision detection of deforming models [5].

7.3 Adaptive hierarchies

When ray tracing highly dynamic scenes, the possibilities to pre-compute efficient data structures are decreased dramatically. For many types of deforming models, however, it still makes sense to pre-build bounding volume hierarchies that can be updated during simulation. In particular, this is the case for models that are not torn apart and never fold into themselves during simulation time.

Efficient approaches for updating the hierarchies have been developed as part of collision detection algorithms for such deforming bodies. When all the vertices are repositioned in a model, it has been suggested that hierarchies of axis-aligned bounding boxes (AABBs) can be completely refitted bottom-up from the leaf nodes [17]. A more efficient hybrid update approach was later developed [5]. AABBs were chosen as bounding volumes for three major reasons. First, finding the optimal AABB of a point set or a polygon set is a very fast operation. Second, it is very efficient to merge k child AABBs into one parent AABB with an optimal fit. Finally, the needed intersection tests between these boxes are very efficient operations.

In this work, we have examined the usefulness of our hybrid update approach for speeding up interactive ray tracing of deforming models. As mentioned earlier, our work focuses on making the reconstruction phase as fast as possible, which is expected to become a serious bottleneck in complex deforming scenes. Our results show that our reconstruction method is very efficient for ray tracing scenes with deforming meshes de-

fined by hundreds of thousands of geometric primitives. In the following sections we discuss the hierarchy preprocessing and the adaptiveness of the hierarchies due to model deformation during interactive simulation.

7.3.1 Initial hierarchy construction

In the hierarchy construction preprocess all the triangles are assigned to different hierarchy nodes. This can simply be done by using the model's initial shape, or alternatively, some average shape of the deforming model, if such information is available. The hierarchy construction can be done by using a top-down [18], bottom-up [19, 20], or an incremental insertion heuristic [21]. Very little is known about how to best pre-build the hierarchies for models that are deformed later on. We have chosen a simple recursive top-down tree building approach [5].

Our nodes use a branching factor of $k = 8$, which we empirically have found to give slightly better performance in our test scenes than hierarchies with branching factors 2 and 4. If a geometry split yields empty child volumes, they are removed and the parent node becomes a k-ary node with $k < 8$. Leaf nodes are formed whenever only one triangle is assigned to a node. A simple hierarchy with binary tree nodes for a two-dimensional model is illustrated in Figure 7.1a.

If a geometry split fails to create at least two non-empty child nodes special handling is required to prevent infinite recursion. This happens rarely, but when it does, we suggest the following split rule. First, sort the primitives' center points in three lists along the principal coordinate axes. Then let the median value in these three lists define the split planes for the geometry partitioning.

Note that the initial primitive partitioning in the hierarchy nodes is not supposed to be changed during simulation. This limits the task of the reconstruction phase to refitting the bounding volumes in the nodes according to the current shapes of the models.

A reasonable variation of our tree building approach would be to always build as balanced trees as possible to minimize the tree height. Perfectly balanced trees, however, do not guarantee better performance than our slightly less balanced trees. Furthermore, it would increase the hierarchy construction time. Nevertheless, when pre-building the hierarchies, without considering any possible future shapes of the models, it can make sense to minimize the tree height.

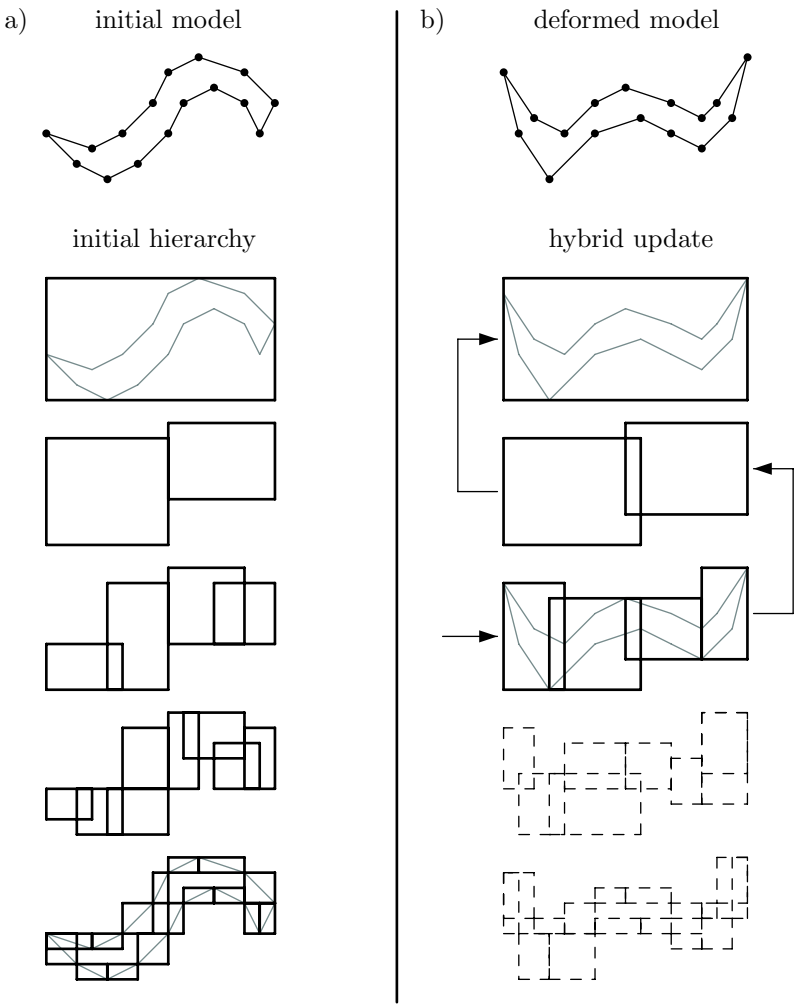


Figure 7.1: a) A pre-built bounding volume hierarchy built from a deforming models initial shape. b) Example of a hierarchy update after the model has been deformed during simulation. The hybrid update method first updates the boxes in the middle level of the hierarchy. Then the levels above are updated incrementally from the middle level boxes. Parts of the lower levels are updated later, as needed, during the ray tracing phase. Note that the resulting boxes have the same optimal fit regardless of they have been directly calculated from their underlying geometry or from child boxes

7.3.2 Efficient hierarchy refitting

The hierarchy reconstruction can be done by completely rebuilding the hierarchies of the deforming models in every time step. However, building a hierarchy from scratch in a top-down manner or by incremental insertion require $O(n \log n)$ time for n triangles. Clearly, this would be infeasible for complex deforming models. Instead, by pre-building the hierarchy structures, we are able to update them during simulation in $O(n)$ time for n deforming geometric primitives.

Our hybrid update method works in the following way. For a hierarchy with height h, we choose to first update boxes at depth $d = \lfloor h/2 \rfloor$. These boxes are updated directly from the point sets or sub-meshes inside them. Then the boxes in the levels above are updated bottom-up by merging child boxes to get parent boxes, starting at level d and proceeding upwards, level by level, towards the root. This part of the update operation is illustrated in Figure 7.1b. Another value of d could have been chosen, but the value we chose was empirically found to yield good results for the models and hierarchies we have made experiments with. Note that the levels below level d are left as they are until it becomes absolutely necessary to update them during the ray tracing phase. In this way, we avoid updating sub-trees in the lower levels that are not needed due to, e.g., occlusion.

Although still a linear operation in the number of triangles, just like a complete bottom-up update, the hybrid update is significantly faster. By exploiting the vertex sharing among triangles in meshes residing in the same bounding box, the updating of the middle level at depth d in the tree requires approximately $n/2$ vertices to be processed for n triangles, given that the average vertex valence is close to six in the triangle meshes. This is always the case for non-trivial closed meshes without holes since the Euler formula states that

$$v + f - e = 2 \tag{7.2}$$

where v, f, and e are the number of vertices, faces, and edges respectively. For meshes built of only triangles this means that

$$v = \frac{1}{2} f + 2 \tag{7.3}$$

since each triangle has three edges and each edge is shared by two triangles [22]. Generally, however, the sub-meshes residing in the middle level

boxes are not completely closed, but still the average vertex sharing is almost as good in many cases. A complete bottom-up update, on the other hand, starts by updating the lowest level in the hierarchy. Since we have one triangle per leaf node, this would require processing $3n$ vertices. Thus, following our reasoning above, this would be approximately six times slower.

Also, the number of boxes to merge to update the boxes above the first updated middle level is only a small fraction of the number of boxes that have to be merged for a complete bottom-up update. For example, a complete k-ary tree with height h has $n = k^h$ leaves and totally there are

$$m_{total} = n + \frac{n-1}{k-1} = \frac{k^{h+1}-1}{k-1} \tag{7.4}$$

nodes in the tree. Assuming h is an even number, we know that the number of nodes at depth $h/2$ is only $\sqrt{n}$, since $\sqrt{n} = \sqrt{k^h} = k^{\frac{h}{2}}$. Also, the total number of nodes in the upper half levels in the same tree is only

$$m_{upper} = \sqrt{n} + \frac{\sqrt{n}-1}{k-1} = \frac{k^{\frac{h}{2}+1}-1}{k-1}. \tag{7.5}$$

This means that the number of boxes updated in the hybrid method will only be

$$r = \frac{m_{upper}}{m_{total}} = \frac{\sqrt{k^h}k-1}{k^h k-1} < \frac{\sqrt{k^h}}{k^h} = \frac{1}{\sqrt{k^h}} = \frac{1}{\sqrt{n}}, \qquad k \geq 2, h \geq 2 \tag{7.6}$$

times the number of boxes updated by a complete bottom-up update method. Furthermore, if we sum the number of vertices and boxes that are processed during the hierarchy update, a predicted speed-up in the reconstruction phase for the hybrid method, compared to a complete bottom-up update, can be described by the following formula:

$$s = \frac{3n + n + \frac{n-1}{k-1}}{\frac{1}{2}n + \sqrt{n} + \frac{\sqrt{n}-1}{k-1}}, \qquad n \geq 4, k \geq 2. \tag{7.7}$$

Thus, s will be in the following intervals for commonly chosen values of

k:

$$8 < s < 10, \qquad k = 2, h \geq 8 \tag{7.8}$$

$$8 < s < 8\frac{2}{3}, \qquad k = 4, h \geq 5 \tag{7.9}$$

$$8 < s < 8\frac{2}{7}, \qquad k = 8, h \geq 4 \tag{7.10}$$

where the tree height, h, defines lower limits for the number of leaf nodes, n, for the different values of k. This implies that the updating of the d top levels might execute more than eight times faster than a complete hierarchy bottom-up update for models with triangle counts of more than a few thousands. In our practical experiments with triangle meshes, this reconstruction method has been found to even execute more than an order of magnitude faster than the full bottom-up update method. Note, however, that we will loose some of the gained execution time during the ray tracing phase, because unlike the complete bottom-up hierarchy update, the hybrid update method postpones updates in the lower levels of the hierarchies until it is known by ray/hierarchy traversals that they are necessary. These late updates are discussed further in the next section.

As stated previously, all of the models' vertices are deformed in every time step of the simulations. In situations where the deformations only occur in a few local areas of a model, there is a better alternative to update the hierarchy. For example, if only m neighboring or close triangles are deformed, where m is relatively small compared to the model's n triangles, the update can be done bottom-up in the hierarchy, but only along the paths from the leaves containing the m triangles up to the root. The running time of the hierarchy update will then be proportional to $O(m + \log n)$. If the m deforming triangles are spatially spread over the leaves in the hierarchy the update time will instead be $O(m \log n)$. Properly implemented, however, it will be no worse than the $O(n)$ running time for the full bottom-up hierarchy update. This approach have been used in a collision detection method developed for surgical training operations [23].

7.3.3 Hierarchy traversals

Our bounding volume hierarchies are stored in simple arrays. In this way, we increase the data locality during program execution. It is important

to keep each tree node stored in the array as small as possible for faster ray traversals. We have found the representation suggested by Smits to be an efficient choice [24].

As mentioned earlier, when using the hybrid update method, the ray tracing phase is responsible for updating the sub-trees in the lower levels that are reached during ray/hierarchy traversals. Pseudo code for the traversal of a deforming model's hierarchy is given in Figure 7.2. As can be seen, recursion has been eliminated by storing each hierarchy in an array with skip indices in the nodes. Each node also has to store an extra integer holding the frame when it was last updated and an additional if-statement (line 4) is executed per reached node during the traversal to trigger necessary node updates. The call on line 5 updates an outdated node's bounding volume the first time it is reached in a traversal during the current frame, which is done directly from the geometry it contains. Thus, we update the lower levels' nodes in a top-down fashion sparsely as they are needed. All that is needed to change the chosen heuristic for updating the lower level sub-hierarchies is to change the function call on line 5 to another node update operation. For example, when an outdated node is reached, one can choose to immediately update the whole subtree below it bottom-up. Another alternative would be to update the next q levels bottom-up. The top-down approach we chose, however, was empirically found to be a bit more efficient than the other alternatives for the models used in our experiments.

Note that additional information, needed for the refit operations, both during the reconstruction phase and the ray tracing phase, are stored in other separate arrays which have references into the main hierarchy arrays that we use during the ray/hierarchy traversals. In this way, we keep the arrays accessed the most during the ray tracing phase smaller.

When the number of deforming models in a simulation is more than just a few, our ray tracing approach needs to be extended to handle multiple deforming models efficiently. One approach is to insert the updated model hierarchies on-the-fly in a top scene hierarchy in which the ray traversals can start [13]. Other data structures that might be suitable for this case have also been described [14], [25]. Another simple alternative, which was used in our implementation, is to sort the updated root boxes along the three principal coordinate axes once per frame. Then, based on a ray's dominant direction with respect to these axes, a reasonably good front-to-back body traversal order is easily found from these sorted lists.

```
ClosestHitTraversal(r, H, b, hit)
    input:  r is the query ray, H the array storing the hierarchy for body b
    output: hit stores the intersection result

    begin
1.      stopInd ← H[0].skipInd
2.      nodeInd ← 0
3.      while(nodeInd < stopInd)
4.          if (H[nodeInd].lastUpdated ≠ currentFrame)
5.              SetBoxOfNodePointSet(b, nodeInd)
6.              H[nodeInd].lastUpdated ← currentFrame
7.          if (OverlapBox(r, H[nodeInd].aabb, hit.t)
8.              if (H[nodeInd].triInd ≥ 0)
9.                  IsectTri(r, b, H[nodeInd].tri, hit)
10.             nodeInd ← nodeInd + 1
11.         else
12.             nodeInd ← H[nodeInd].skipInd
    end
```

Figure 7.2: Pseudo code for the closest hit ray/hierarchy traversal including node updates the first time outdated nodes are reached

Note that, regardless of the number of dynamic models, these lists can be kept sorted in expected linear time, given a high temporal coherence for the moving bodies in the scene.

7.4 Experiments

To evaluate our update strategies, we have conducted experiments based on a number of test scenes. Here, we report results from two different test scenes, which reveal both the strengths and weaknesses of our approach. Our system was implemented in C++ and the experiments were run using a standard PC, with a single 1.5 GHz Pentium IV CPU and 512 Mb of memory.

The goal resolution we used for our images in these experiments was 640x480 pixels. However, to achieve interactive frame rates on a single CPU computer, we used image sub-sampling techniques. In this way the ray tracing phase can be executed 10 – 100 times faster and the reconstruction phase is likely to become a bottleneck. We used a regular sub-sampling pattern with bilinear interpolation to scale the image to the goal resolution. First, we render a lower resolution image into an internal bitmap. Then, to scale the image we let each neighboring group of 2x2 pixels define the colors in the corners of a quad, which then can be rendered by using OpenGL. In this way, the color interpolation was done in hardware. We found, however, that a somewhat better image quality was achieved when each quad was tessellated into four triangles meeting at the quad's midpoint, before the primitives were sent to OpenGL. A sophisticated filtering algorithm would of course produce better scaled images, but it would also be much slower.

Another alternative we have tried to speed up the ray tracing phase was to use frameless rendering [26]. By only rendering a fraction of all the pixels at a time, chosen according to a pseudo random pattern, a significant performance gain can be expected. The image quality, however, was far from acceptable in our test scenes, since both the viewer as well as most of the geometry in the scene are changing at nearly all times. There is, however, some ongoing research aiming at improving the applicability of frameless rendering [27].

In our first experiment, we used a scene with 9 deforming bodies, where each one of them had 81,920 triangles. Thus, in total, the scene was modelled by 737,280 deforming triangles. There were two light

sources in the scene and all 9 bodies were reflective. Apart from primary rays, shadow rays as well as the first order reflection rays were cast. The simulation was run for 280 frames. We defined the camera movements so that the number of visible bodies varied throughout the simulation, but during the majority of the frames more than half or all of the bodies contributed to the ray traced image. No a priori knowledge about the forthcoming deformations was utilized. The simulation can thus be regarded as completely dynamic and interactive. Some images from the experiment are shown in Figure 7.3.

We report the performance of this simulation for three different cases. First, we traced one ray per pixel. Then we used image sub-sampling for the other two cases, so that one traced ray was mapped to a pixel area of 5x5 and 10x10 respectively. We report the average update time as well as the ray tracing time over all frames. Furthermore, we give the best and the worst frame times in the whole simulation. These results and the achieved speed-ups are given in Table 7.1.

As we can see, in the two cases where sub-sampling were used, the hybrid update method was superior, yielding a total average speed-up of 1.5 and 2.6 respectively. In the best case, which occurred at frame 200, the speed-up was approximately 8 and 12, respectively. Note that the update phase took the same time every frame and it was not affected by the ray tracing time. The update phase runs in $O(n)$ time for n faces for both methods. Nevertheless, in this experiment the hybrid update method was approximately 17 times faster in every frame of the simulation. Part of this performance advantage was, however, lost during the ray tracing phase, when parts of the lower subtrees had to be updated as they were needed during the ray/hierarchy traversals.

In another experiment, we used the Museum scene defined in BART Benchmark scenes [28]. This scene includes a deforming piece of art, which essentially is a triangle soup with drastic changes over time. There are two light sources in the scene. We traced primary rays, shadow rays as well as the first order reflection rays. The deforming model exists in different levels of details, but note that we only consider the most detailed version of it which is modelled by 65,536 triangles. We generated 300 frames for the whole animation. Images from four of the frames are shown in Figure 7.4.

This scene is an example of a scene that clearly is unsuitable for our pre-constructed hierarchies. The deforming art piece is in fact defined by five different triangle soup constellations, each one having 65,536

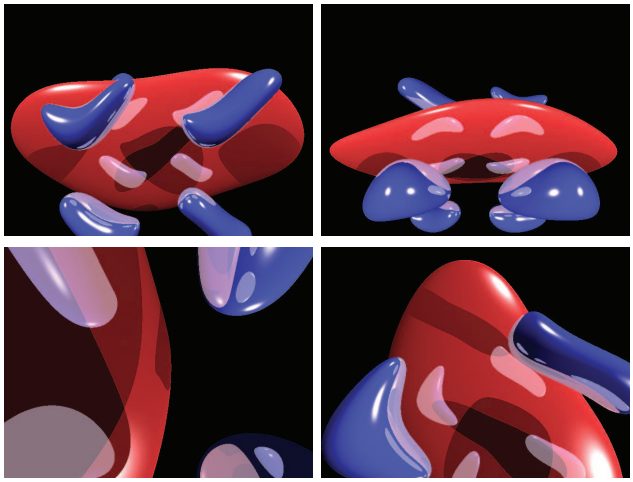


Figure 7.3: Images of frames 50, 65, 185, and 250 in the first experiment

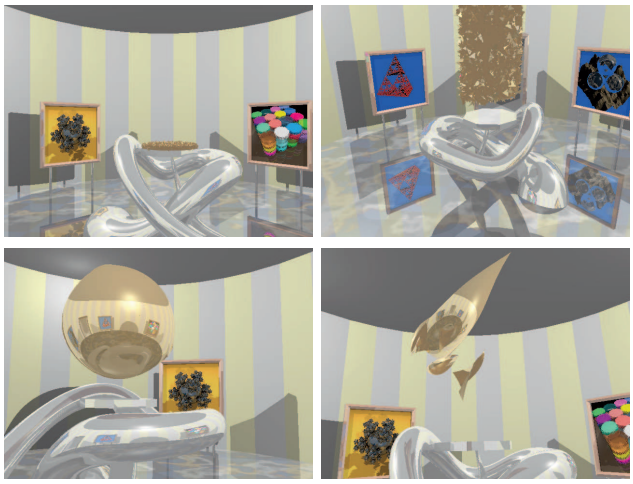


Figure 7.4: Images of frames 20, 120, 190, and 230 from the second experiment. The test scene used was the BART Museum with complexity level 8

case	sampl	bottom-up method (s)			hybrid method (s)			speedup
		up	rt	tot	up	rt	tot	
ave	1x1	0.449	8.249	8.698	0.026	8.705	8.730	0.996
	5x5	0.449	0.375	0.825	0.026	0.527	0.553	1.492
	10x10	0.449	0.100	0.550	0.026	0.183	0.209	2.633
worst	1x1	0.450	15.940	16.390	0.026	16.516	16.542	0.991
	5x5	0.451	0.711	1.161	0.026	0.912	0.938	1.237
	10x10	0.450	0.188	0.638	0.025	0.308	0.333	1.916
best	1x1	0.450	0.581	1.031	0.026	0.604	0.630	1.636
	5x5	0.450	0.035	0.484	0.026	0.037	0.063	7.738
	10x10	0.448	0.010	0.458	0.026	0.012	0.038	12.042

Table 7.1: Performance measurements from the first experiment. Timings are given for the reconstruction phase (up), the ray tracing phase (rt), and the sum of them (tot)

triangles, defined at the following key frame times: 0.0, 1.0, 2.0 3.0, and 4.0 seconds. During simulation, the triangles are deformed by linear interpolation between the key frame triangle soups. We ray traced the scene by pre-constructing five different bounding volume hierarchies, one for each key frame triangle soup. Then, during the course of simulation, we switched the active hierarchy. In this way, we always had an active pre-constructed hierarchy for the triangle soup and we were able to use the bottom-up update method as well as our hybrid update method. Note that we used a priori information for the deforming model in this experiment.

Despite this solution, the scene is still a really bad case for our method for two major reasons. There are no connected triangles in the triangle soup, which means that updating the middle level of the hierarchy requires processing $3n$ vertices for n triangles. Furthermore, almost all parts of the scene contribute to the final image, with the exception of the very last part of the whole animation. This is because the scene only consists of a single room and the deforming polygon soup is positioned in the center area of the room and it reflects much of the environment around it.

We report the results from rendering the Museum scene in Table 7.2. As can be seen, the update times are very fast for both update methods, 16 ms for the hybrid update and 63 ms for the bottom-up update. This can be compared to the on-the-fly hierarchy construction time reported

case	sampl	bottom-up method (s)			hybrid method (s)			speedup
		up	rt	tot	up	rt	tot	
ave	1x1	0.063	32.1	32.163	0.016	31.9	31.916	1.008
	5x5	0.063	1.32	1.383	0.016	1.40	1.416	0.977
	10x10	0.063	0.335	0.398	0.016	0.395	0.411	0.968

Table 7.2: Performance measurements from the BART Museum scene, complexity level 8, in the second experiment

by Wald et al. [13]. For exactly the same scene their reconstruction phase took more than 1 second.[1] Thus, their reconstruction method prohibits interactive simulation of complex deforming scenes.

Note, however, that since almost all parts of the scene contributes to the rendered frames at almost all times, the hybrid update method gives no advantage over the bottom-up update method in the average frame time. What was won in the reconstruction phase was later lost during the necessary remaining updates in the ray tracing phase. The overall performance was almost the same for both methods in this case.

7.5 Discussion

Drawing from the experiments we have carried out, we believe the hybrid update method is applicable and preferable in several different situations. For example, in scenes with much occluded geometry, many of the models in the scene do not contribute to a particular ray traced view at all and other models are only visible in parts. Hence, completely updating the acceleration data structures for such models might become a serious bottleneck. The hybrid update method, however, updates these structures very efficiently and sparsely. Note also that the benchmarks that we used here do not contain much occluded or otherwise invisible deformable geometry, and therefore, we expect that our algorithm will work even better for such scenes.

Although we have not tried, we expect our method to work well in multi-processor ray tracing [2, 6]. For example, when scene replication is used among the different nodes, the master machine would first execute the efficient hybrid update method and then the clients can get an early

[1] They used a cluster of dual AMD AthlonMP 1800+ machines with a dual AMD AthlonMP 1700+ server in their experiments.

start tracing rays. This also decreases network bandwidth because nodes only need to request the upper part of the hierarchies first, as they are needed, and only when it is absolutely necessary the bottom sub-trees would be updated and sent to clients.

Special purpose ray tracing hardware has also been designed, increasing the performance of the ray tracing phase by orders of magnitudes [15]. If the reconstruction phase is done on the CPU for the deformable models, it is very likely to become the bottleneck. Also in this situation, we expect the hybrid update method to be an attractive choice.

For time-critical ray tracing, we believe our method can be very attractive. To achieve a constant frame rate, the ray tracer is aborted according to a time budget of say 50 ms per frame. In this case, the rays are traced in a breadth-first manner to ensure that at all the primary rays are cast before any shadows, reflection, or refraction rays. If complete bottom-up refit operations are executed before the first ray is cast, there would be no time left to cast a single ray in many scenes. Among our test scenes, only the hybrid method would allow the tracing of rays to start.

Furthermore, we have found that the hybrid update method is completely superior when relatively few rays are cast in a scene with many complex deformable models. This means, for example, that we can expect our method to be highly suitable for applications that need picking or other algorithms depending on a moderate number of line/scene intersection tests.

When the scene includes triangle soups undergoing unstructured motion, a more general approach is needed. Pre-built bounding volume hierarchies, updated as we have described, tend to become more and more unsuitable as the simulation proceeds, given that drastic changes occur in the geometric primitives relative location to one another. In this case, a data structure that allows primitive insertion in $O(1)$ time would be beneficial, so that all the primitives in a triangle soup can be inserted in the acceleration data structure in linear time during interactive simulation.

A simple solution in this case can be based on uniform grids [29, 7]. First the AABB of the deforming model is calculated and the resolution of the uniform grid within this box is determined, for example, one can use the $\sqrt[3]{n}$-criterion, or some more efficient variation of it [19, 30]. Then all the primitives can be assigned to all cells they intersect in expected linear time. For deforming polygon soups, where the primitives stay

rather uniformly distributed, this might be a very efficient approach. Some other data structures, that might be suitable for this situation, have also been suggested [14, 25].

7.6 Conclusions and future work

Interactive ray tracing needs multi-processor environments or specialized hardware to be a feasible alternative for interactive graphics applications. Still, by using a single standard PC together with image sub-sampling and adaptive acceleration hierarchies, we were able to achieve interactive frame rates for dynamic scenes with hundreds of thousands of deforming polygons.

The reconstruction phase executes more than an order of magnitude faster when using the hybrid update method compared to the complete bottom-up update for connected triangle meshes. This allows the ray tracing phase to get an early start, because not so much time is wasted on updating parts of the hierarchies that are not needed in the ray/hierarchy traversals.

Our update approach, however, requires additional bounding volume updates in the lower levels of the hierarchies during the ray tracing phase. When a lot of volumes have to be updated in this way, the time won during the reconstruction phase might be lost in the ray tracing phase. Nevertheless, we have found that in scenes where some deforming models only partly contribute to the final image, they might be out of sight or occluded by other models, a significant speed-up can be achieved also in the total rendering time. For example, in the first experiment the average speed-ups were 1.5 and 2.6.

In our future work, there are many possible optimizations that would allow us to improve the frame rate and image quality. For example, in our current implementation we have not taken advantage of the frequently occurring coherence for neighboring rays [6]. Neither have we used any SIMD instructions, e.g., to optimize intersection or shading calculations.

Future improvements also include supporting different types of deforming geometric primitives as well as porting our implementation to a multi-processor environment. Apart from only parallelizing the ray tracing phase, it would also be interesting to examine possible ways of parallelizing the reconstruction phase. Another possibility would be to create hierarchies that are aware of how the models can deform and

take advantage of that information for faster reconstruction. Such an approach has been developed for collision detection between morphing models [31]. The same approach would also be possible in ray tracing of morphing models. Finally, it would also be interesting to investigate the possibilities for a hardware implementation of our approach.

Bibliography

[1] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[2] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 119–126. ACM Press, 1999.

[3] L. Szirmay-Kalos and G. Márton. Worst-case versus average case complexity of ray-shooting. *Computing*, 61(2):103–131, 1998.

[4] J. Hennesey and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2002.

[5] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics Conference 2001, Short presentations*, pages 325–333, September 2001.

[6] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[7] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 119–128. ACM Press, 1987.

[8] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[9] A. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8:60–70, 1988.

[10] Gregory Ward and Maryann Simmons. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics (TOG)*, 18(4):361–368, 1999.

[11] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. In *State of the art reports, Eurographics 2001*, September 2001.

[12] Jonas Lext and Tomas Akenine-Möller. Towards rapid reconstruction for animated ray tracing. In *Eurographics Conference 2001, Short presentations*, pages 311–318, September 2001.

[13] I. Wald, C. Benthin, and P. Slusallek. A simple and practical method for interactive ray tracing of dynamic scenes. Technical report, Saarland University, 2002.

[14] E. Reinhard, B. Smits, and C. Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 299–306, 2000.

[15] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. Saarcor: a hardware architecture for ray tracing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 27–36. Eurographics Association, 2002.

[16] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (TOG)*, 21(3):703–712, 2002.

[17] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[18] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[19] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: a new solution for ray tracing complex scenes. *Computer Graphics Forum*, 14:371–382, 1995.

[20] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, August 1996.

[21] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[22] E. Haines. Triangles per vertex. *Ray Tracing News*, 14, 2001.

[23] J. Brown, S. Sorkin, C. Bruyns, and J. Latombe. Real-time simulation of deformable objects: Tools and application. In *Proceedings of Computer Animation*, November 2001.

[24] B. Smits. Efficiency issues for ray tracing. *Journal of graphics tools*, 3(2):1–14, 1998.

[25] T. Ulrich. Loose octress. In Mark DeLoura, editor, *Game Programming Gems*, pages 444–453. Charles River Media, 2000.

[26] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. Frameless rendering: double buffering considered harmful. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176. ACM Press, 1994.

[27] A. Dayal, B. Watson, and D. Luebke. Improving frameless rendering by focusing on change. In *ACM SIGGRAPH 2002 Conference abstracts and applications*, page 201. ACM Press, 2002.

[28] Jonas Lext, Ulf Assarsson, and Tomas Akenine-Möller. A benchmark for animated ray tracing. *IEEE Computer Graphics and Applications*, 21:22–31, 2001.

[29] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated raytracing system. *IEEE Computer Graphics and Applications*, 6:16–26, 1986.

[30] K. Klimaszewski and T. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17:42–51, 1997.

[31] Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19:164–174, 2003.