Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models

Thomas Larsson
Department of Computer Science and Engineering
Mälardalen University
Sweden

Tomas Akenine-Möller
Department of Computer Engineering
Chalmers University of Technology
Sweden

February 21, 2003

MRTC Report, ISSN 1404-3041 ISRN MDH-MRTC-92/2003-1-SE

Key words: ray tracing – deformable models – hierarchical data structures – interactive simulation

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.

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 [Whit80]. 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 [Park99].

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 [Szir98]. 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 [Park99]. 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 [Henn99]. 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

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

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 [Lars01]. 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 [Wal01a]. Supporting other types of geometry can be done through tessellation [Snyd97] 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.

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 [Glas89]. 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 [Glas88]. 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 [Ward99, Park99, Wal01a, Wal01b]. 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 [Park99]. An acceleration data structure allowing scenes of dynamically moving models with rigid parts, like for example walking robots, has also been proposed [Lex01b]. 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. [Wald02].

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 [Wald02]. 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 [Rein00]. 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 presented by Scmittler et al. [Schm02]. 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 [Purc02], 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 re-positioned each frame of the animation. Our approach builds upon our earlier work on efficient collision detection of deforming models [Lars01].

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 [Berg97]. A more efficient hybrid update approach was later developed [Lars01]. 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 defined 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.

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 [Klos98], bottom-up [Caza95, Bare96], or an incremental insertion heuristic [Gold87]. 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 [Lars01].

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 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.

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 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 va-

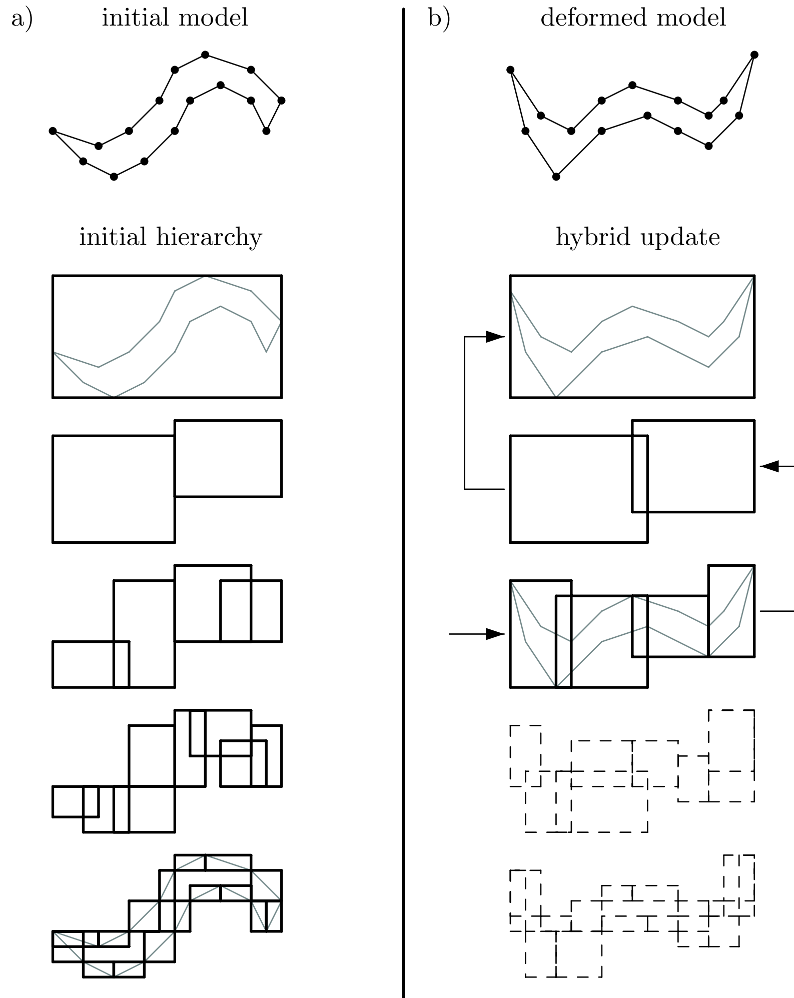

Figure 1: a) A pre-built bounding volume hierarchy built from a deforming models initial shape. b) Example of a hierarchy update after the model has been deformed during simulation. The hybrid update method first updates the boxes in the middle level of the hierarchy. Then the levels above are updated 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

lence 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

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

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

$$v = \frac{1}{2}f + 2 \tag{3}$$

since each triangle has three edges and each edge is shared by two triangles [Hain01]. 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

$$m_{total} = n + \frac{n-1}{k-1} = \frac{k^{h+1}-1}{k-1} \tag{4}$$

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

$$m_{upper} = \sqrt{n} + \frac{\sqrt{n}-1}{k-1} = \frac{k^{\frac{h}{2}+1}-1}{k-1}. \tag{5}$$

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

$$r = \frac{m_{upper}}{m_{total}} = \frac{\sqrt{k^h}k-1}{k^h k-1} < \frac{\sqrt{k^h}}{k^h} = \frac{1}{\sqrt{k^h}} = \frac{1}{\sqrt{n}}, \qquad k \geq 2, h \geq 2 \tag{6}$$

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:

$$s = \frac{3n + n + \frac{n-1}{k-1}}{\frac{1}{2}n + \sqrt{n} + \frac{\sqrt{n}-1}{k-1}}, \qquad n \geq 4, k \geq 2. \tag{7}$$

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

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

$$8 < s < 8\frac{2}{3}, \qquad k = 4, h \geq 5 \tag{9}$$

$$8 < s < 8\frac{2}{7}, \qquad k = 8, h \geq 4 \tag{10}$$

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 [Brow01].

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 [Smit98].

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 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

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

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 [Wald02]. Other data structures that might be suitable for this case have also been described [Rein00], [Ulri00]. 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. 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.

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 [Bish94]. 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 [Daya02].

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 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 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

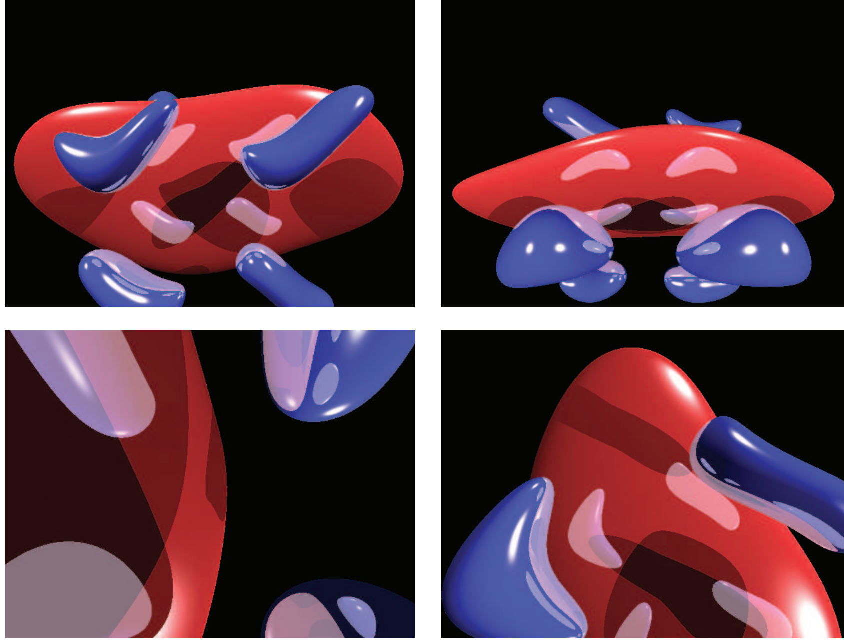

Figure 3: Images of frames 50, 65, 185, and 250 in the first experiment

updated as they were needed during the ray/hierarchy traversals.

In another experiment, we used the Museum scene defined in BART Benchmark scenes [Lex01a]. 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 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 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.

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 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)

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

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 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 by Wald et al. [Wald02]. 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.

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

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

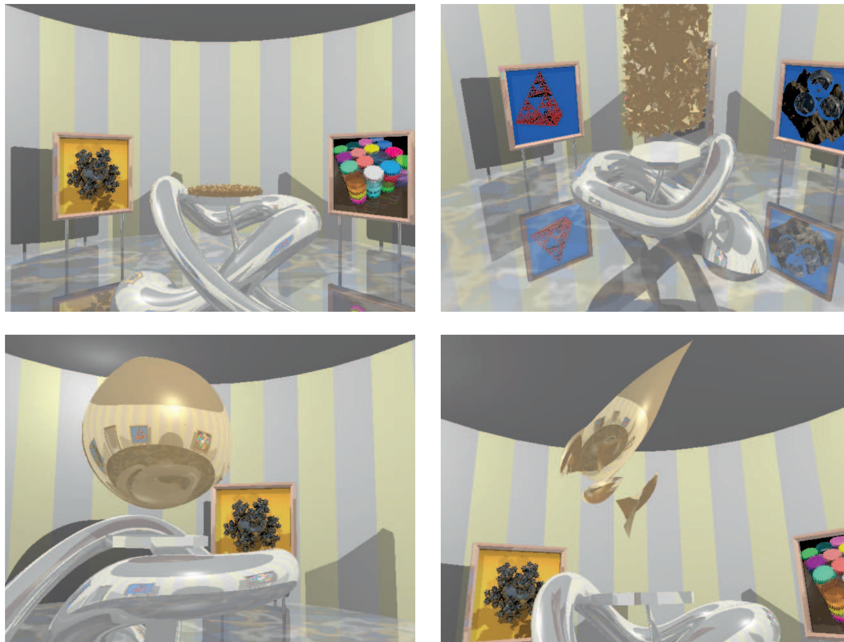

Figure 4: Images of frames 20, 120, 190, and 230 from the second experiment. The test scene used was the BART Museum with complexity level 8

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 [Park99, Wal01a]. 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 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 [Schm02]. 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 deforming meshes. 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 [Fuji86, Snyd97]. 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 [Caza95, Klim97]. 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 [Rein00, Ulri00].

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 [Wal01a]. 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 [Lars02]. 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.

References

[Bare96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell and A. Tal. "BOX-TREE: A Hierarchical Representation for Surfaces in 3D." *Computer Graphics Forum*, 15(3): 387-396, 1996.

[Berg97] G. van den Bergen. "Efficient Collision Detection of Complex Deformable Models using AABB Trees." *journal of graphics tools*, 2(4): 1–14, 1997.

[Bish94] G. Bishop, H. Fuchs, L. McMillan and E. Sher Zagier. "Frameless Rendering: Double Buffering Considered Harmful." *Computer Graphics (SIGGRAPH 94)*, pp. 175–176, 1994.

[Brow01] J. Brown, S. Sorkin, C. Bruyns and J. Latombe. "Real-Time Simulation of Deformable Objects: Tools and Application." *Proceedings of Computer Animation 2001*, Seoul, Korea, November 6–8, 2001.

[Caza95] F. Cazals, G. Drettakis and C. Puech. "Filtering, Clustering and Hierarchy Construction: a New Solution for Ray Tracing Complex Scenes." *Computer Graphics Forum*, 14(3): 371–382, 1995.

[Daya02] A. Dayal, B. Watson and D. Luebke. "Improving frameless rendering by focusing on change." Siggraph sketch, In conference abstracts and application, ACM Siggraph, 2002.

[Fuji86] A. Fujimoto, T. Tanaka and K. Iwata. "ARTS: Accelerated Ray-Tracing System." *IEEE Computer Graphics and Applications*, 16(6):16–26, 1986.

[Glas88] A. Glassner. "Spacetime Ray Tracing for Animation." *IEEE Computer Graphics and Applications*, pp. 60–70, March 1988.

[Glas89] A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.

[Gold87] J. Goldsmith and J. Salmon. "Automatic Creation of Object Hierarchies for Ray Tracing." *IEEE Computer Graphics and Applications*, 7(5): 14–20, May 1987.

[Hain01] E. Haines "Triangles per Vertex." Ray Tracing News, 14(1), 2001.

[Henn99] J. Hennesey and D. Patterson Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 2002.

[Klim97] K. Klimaszewski and T. Sederberg. "Faster Ray Tracing Using Adaptive Grids." *IEEE Computer Graphics and Applications*, 17(1)42–51, 1997.

[Klos98] 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.

[Lars01] T. Larsson and T. Akenine-Möller. "Collision Detection for Continuously Deforming Bodies." *Eurographics Conference 2001*, short presentation, pp. 325–333, 2001.

[Lars02] T. Larsson and T. Akenine-Möller. "Efficient Collision Detection for Models Deformed by Morphing." *The Visual Computer*, (DOI) 10.1007/s00371-002-0190-y, 5 February 2003.

[Lex01a] J. Lext, U. Assarsson and T. Akenine-Möller. "A Benchmark for Animated Ray Tracing." *IEEE Computer Graphics and Applications*, pp. 22–31, March/April 2001.

[Lex01b] J. Lext and T. Akenine-Möller "Towards Rapid Reconstruction for Animated Ray Tracing." *Eurographics Conference 2001*, short presentation, pp. 311–318, 2001.

[Park99] S. Parker, W. Martin, P.J. Sloan, P. Shirley, B. Smits and C. Hansen. "Interactive Ray Tracing." *ACM Symposium on Interactive 3D Graphics*, pp. 119–126, 1999.

[Purc02] T. Purcell, I. Buck, W. Mark and P. Hanrahan. "Ray Tracing on Programmable Graphics Hardware." *ACM Transactions on Graphics*, 21(3):703–712, 2002.

[Rein00] E. Reinhard, B. Smiths and C. Hansen. "Dynamic Acceleration Structures for Interactive Ray Tracing." Proceedings of the 11th Eurographics Workshop on Rendering, Brno, Czech Republic, pp. 299–306, 2000.

[Schm02] J. Scmittler, I. Wald, P. Slusallek "SaarCOR – A Hardware Architecture for Ray Tracing." Eurographics/SIGGRAPH Graphics Hardware Workshop 2002, Saarbrücken, Germany, September, 2002.

[Smit98] B. Smits "Efficiency Issues for Ray Tracing." *journal of graphics tools*, 3(2):1–14, 1998.

[Snyd97] J. Snyder and A. Barr. "Ray Tracing Complex Models Containing Surface Tessellations." *Computer Graphics (SIGGRAPH 87)*, 21(4): 119–128, 1987.

[Szir98] L. Szirmay-Kalos and G. Márton. "Worst-case versus average case complexity of ray-shooting." *Computing*, 61(2):103–131, 1998.

[Ulri00] T. Ulrich. "Loose Octress." In Mark DeLoura, ed., Game Programming Gems, Charles River Media, ISBN 1-58450-049-2, pp. 444–453, 2000.

[Wal01a] I. Wald, P. Slusallek, C. Benthin and M. Wagner. "Interactive Rendering with Coherent Ray Tracing." *Computer Graphics Forum*, 20(3):153–164, 2001.

[Wal01b] I. Wald and P. Slusallek "State of the Art in Interactive Ray Tracing." *Eurographics Conference 2001*, State of the Art Report, 2001.

[Wald02] I. Wald, C. Benthin and P. Slusallek. "A Simple and Practical Method for Interactive Ray Tracing of Dynamic Scenes." Technical Report, 2002.

[Ward99] G. Ward and M. Simmons "The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments." *ACM Transactions on Graphics*, 18(4):361–398, 1999.

[Whit80] T. Whitted. "An improved illumination method for shaded displays." *Communications of the ACM*, 23(6):343–349, 1980.