The Visual Computer

Efficient collision detection for models deformed by morphing

Thomas Larsson[1],
Tomas Akenine-Möller[2]

[1] Department of Computer Science and Engineering, Mälardalen University, Sweden
E-mail: thomas.larsson@mdh.se
[2] Department of Computer Engineering, Chalmers Institute of Technology, Sweden

Published online: 5 February 2003
© Springer-Verlag 2003

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

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) [22]. This means that we can expect more detailed and realistic deformed 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 prebuilt 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 Sects. 3 and 4, the details of our collision-detection method and the promising results we have achieved are discussed. In Sect. 5, further optimisation techniques are presented. Finally, our conclusions and some opportunities for future work are given.

2 Previous work

The collision-detection problem for bodies undergoing arbitrary deformation is an open research area [15]. Some initial efforts are described in the literature. For example, Smith et al. suggested a very general algorithm [17, 25] based on recomputing 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 [2, 3, 11] and for cloth simulation [23, 26–28].

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-volume trees. Some interesting collision-detection algorithms have used spheres [12–14, 24], axis-aligned bounding boxes [5, 29], oriented bounding boxes [8], discrete-orientation polytopes [18, 30], quantized orientation slabs with primary orientations [10], and convex-surface decompositions [6] 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 [5], 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 [19]. 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, such as 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.

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 preprocessing pass. Bounding-volume hierarchies are often built by using a top-down [18], bottom-up [4] or incremental-insertion heuristic [7]. 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 [9]. 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) [8]. 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* [5]. 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 the 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-volume trees is described and evaluated for models deformed by morphing.

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 [31].

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 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 Fig. 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 [16].

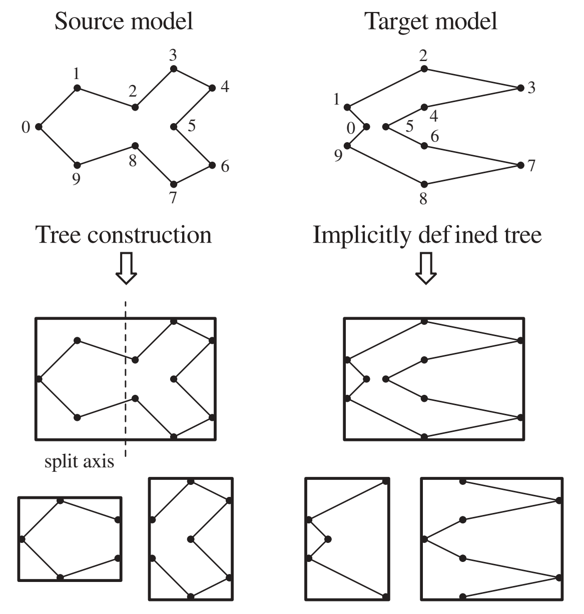


Fig. 1. Creating a morphing-aware bounding volume hierarchy. A source and target model and their vertex mapping, defined by the corresponding vertex labels, are shown at the *top*. Only the mesh of the source model is used to partition the geometric primitives into a bounding-volume tree, which is illustrated 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

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 associated 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

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

Any vertex p_m of the morphing body can then be defined as a convex combination of its corre-

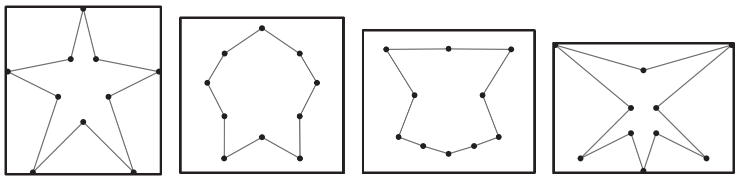


Fig. 2. A morphing star polygon and its AABB in 2D. The source model and target model are shown to the *left* and *right* respectively, and two of the inbetween models are shown between them. Although not optimal, the blended AABB always covers the morphing model. How large the excess volume of the blended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

sponding reference vertices p_{im} and weights w_i; that is,

$$p_m = \sum_{i=1}^{n} p_{im} w_i. \tag{2}$$

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 [8].

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 $n = (n_1, n_2, \dots, n_{k/2})$. 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:

$$B' = (s', e') = \left(\sum_{i=1}^{n} s_i w_i, \sum_{i=1}^{n} e_i w_i \right). \tag{3}$$

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 $0 \leq j \leq k/2$. 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,

$$s_{ij} \leq p_{im} \cdot n_i \leq e_{ij}. \tag{4}$$

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

$$s_{ij} w_i \leq (p_{im} \cdot n_i) w_i \leq e_{ij} w_i, \tag{5}$$

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:

$$\sum_{i=1}^{n} s_{ij} w_i \leq \sum_{i=1}^{n} (p_{im} \cdot n_i) w_i \leq \sum_{i=1}^{n} e_{ij} w_i. \tag{6}$$

Equation (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 Fig. 2. Clearly, this is an extremely efficient update or refit operation.

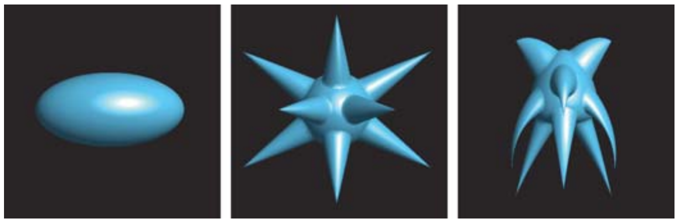


Fig. 3. The three reference models used to define the morphing models in the first experiment

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:

$$B' = (c', r') = \left(\sum_{i=1}^{n} c_i w_i, \sum_{i=1}^{n} r_i w_i \right). \tag{7}$$

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:

$$\| p_{im} - c_i \| \le r_i. \tag{8}$$

This also holds when all the terms in Eq. (8) are multiplied with w_i (since $w_i \ge 0$); that is,

$$\| p_{im} w_i - c_i w_i \| = \| p_{im} - c_i \| w_i \le r_i w_i. \tag{9}$$

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

$$\sum_{i=1}^{n} \| p_{im} - c_i \| w_i \le \sum_{i=1}^{n} r_i w_i. \tag{10}$$

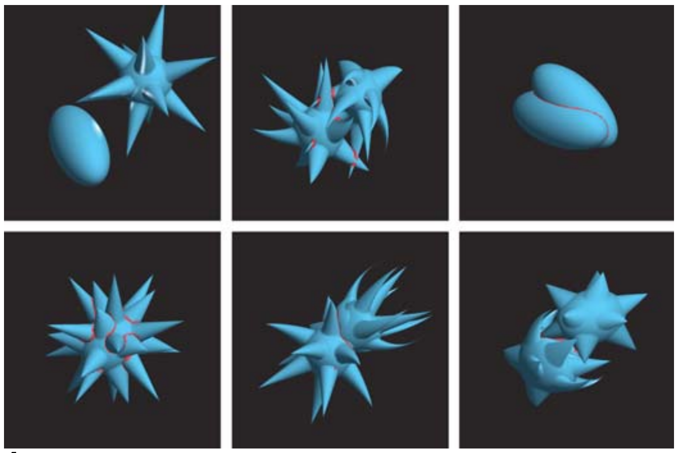
Equation (10) shows, as expected, that the blended sphere will always contain the blended geometry.

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.

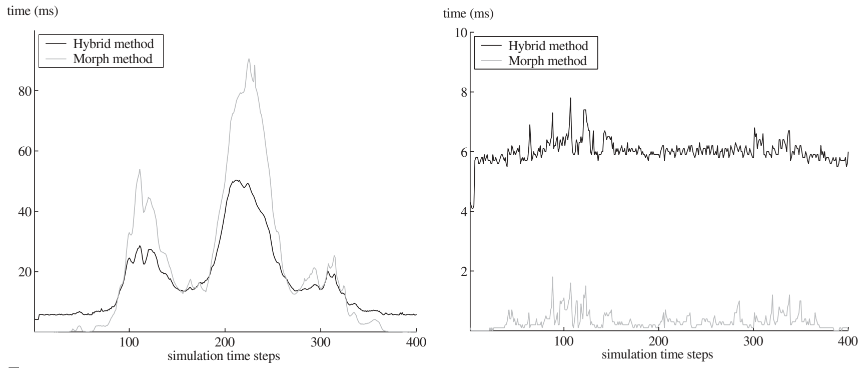
4 Results

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 Sect. 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 20 480 triangles. The three reference models used can be seen in Fig. 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 Fig. 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 Fig. 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 [19], 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



4



5

Fig. 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

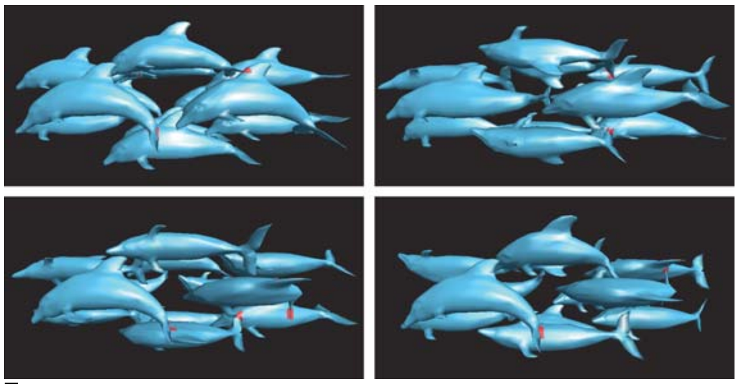
Fig. 5. Measured performance of the collision-detection algorithm in the first experiment when reporting all intersecting triangle pairs (*left*) and when only reporting the first-found triangle pair (*right*)

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.

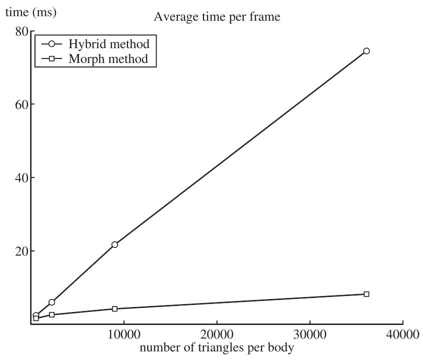
In the second experiment, twelve morphing dolphin models were used. Each dolphin was defined by three reference models, which are shown in Fig. 6. In our scenario, the dolphins swam closely together as a group in order to generate interesting intersections. Besides the ongoing morphing of the dolphins, 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 Fig. 7, images from some of the frames in this simulations are presented.

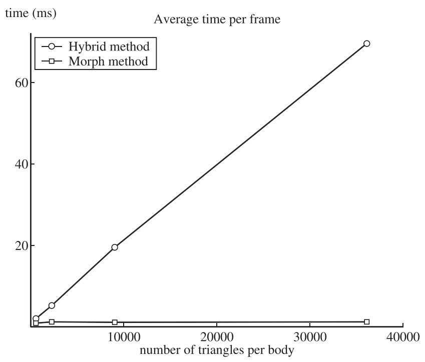
This simulation was repeated with varying triangle counts for the dolphin models, but we ensured that the shape of these different models were the same. In these repetitions, all the dolphin models had 564, 2256, 9024 and 36 096 triangles respectively. The results of the simulations are reported in Fig. 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.

6



7





8

Fig. 6. The three reference models used to define the morphing dolphin models in the second experiment

Fig. 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*

Fig. 8. Results from the second experiment when reporting all intersecting triangle pairs (*left*) and when only reporting the first-found intersecting triangle pair (*right*) for all intersecting dolphin pairs in the scene

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.

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 Fig. 2. We can improve the tightness of our blended boxes by inserting new reference bounding volumes between the ones produced 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 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 [19].

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 [6, 21] 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 [20] describing various approaches. More recently, an overview of available methods in the area of mesh morphing has also been published [1].

References

1. Alexa M (2001) Mesh morphing. In: State of the art reports, Eurographics 2001, Manchester, UK, 3–7 September 2001
2. Baraff D (1990) Curved surfaces and coherence for non-penetrating rigid body simulation. In: Proceedings of the 17th annual conference on computer graphics and interactive techniques. ACM Press, New York
3. Baraff D, Witkin A (1992) Dynamic simulation of non-penetrating flexible bodies. In: Proceedings of the 19th annual conference on computer graphics and interactive techniques. ACM Press, New York
4. Barequet G, Chazelle B, Guibas L, Mitchell J, Tal A (1996) BOXTREE: a hierarchical representation for surfaces in 3D. Comput Graph Forum 15(3):387–396

5. van den Bergen G (1997) Efficient collision detection of complex deformable models using AABB trees. J Graph Tools 2(4):1–14
6. Ehmann S, Lin MC (2001) Accurate and fast proximity queries between polyhedra using convex surface decomposition. Comput Graph Forum 20(3):500–510
7. Goldsmith J, Salmon J (1987) Automatic creation of object hierarchies for ray tracing. IEEE Comput Graph Appl 7(5):14–20
8. Gottschalk S, Lin MC, Manocha D (1996) OOBTree: a hierarchical structure for rapid interference detection. In:Proceedings of the 23rd annual conference on computer graphics and interactive techniques. ACM Press, New York
9. Gottschalk S (2000) Collision queries using oriented bounding boxes. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill
10. He T (1999) Fast collision detection using QuOSPO trees. In: 1999 Symposium on interactive 3D graphics, Atlanta, Ga., 26–28 April 1999. ACM Press, New York
11. von Herzen B, Barr A, Zatz H (1990) Geometric collisions for time dependent parametric surfaces. In: Proceedings of the 17th annual conference on computer graphics and interactive techniques. ACM Press, New York
12. Hubbard PM (1993) Interactive collision detection. In: IEEE 1993 symposium on research frontiers in virtual reality, San Jose, Calif., 23–26 October 1993. IEEE Computer Society Press, Los Alamitos
13. Hubbard PM (1995) Collision detection for interactive graphics applications. IEEE Trans Vis Comput Graph 1(3):218–230
14. Hubbard PM (1996) Approximating polyhedra with spheres for time-critical collision detection. ACM Trans Graph 15(3):179–210
15. Jiménez P, Thomas F, Torras C (2001) 3D collision detection: a survey. Comput Graph 25(2):269–285
16. Kent J, Carlson W, Parent R (1992) Shape transformation for polyhedral objects. In: Proceedings of the 19th annual conference on computer graphics and interactive techniques. ACM Press, New York
17. Kitamura Y, Smith A, Takemura H, Kishino F (1998) A real-time algorithm for accurate collision detection for deformable polyhedral objects. Presence 7(1):36–52
18. Klosowski JT, Held M, Mitchell JSB, Sowizral H, Zikan K (1998) Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Trans Vis Comput Graph 4(1):21–36
19. Larsson T, Akenine-Möller T (2001) Collision detection for continuously deforming bodies. In: Short presentations, Eurographics 2001, Manchester, UK, 3–7 September 2001
20. Lazarus F, Verroust A (1998) Three-dimensional metamorphosis: a survey. Vis Comput 14:373–389
21. Li T-Y, Chen J-S (1998) Incremental 3D collision detection with hierarchical data structures. In: Proceedings of the ACM symposium on virtual reality software and technology 1998. ACM Press, New York
22. Lindholm E, Kilgard M, Moreton H (2001) A user-programmable vertex engine. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques. ACM Press, New York
23. Liu J-D, Ko M-T, Chang R-C (1996) Collision avoidance in cloth animation. Vis Comput 12:234–243
24. Palmer I, Grimsdale R (1995) Collision detection for animation using sphere-trees. Comput Graph Forum 14(2):105–116
25. Smith A, Kitamura Y, Takemura H, Kishino F (1995) A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In: Virtual reality annual international symposium ’95, Research Triangle Park, N.C., 11–15 March 1995. IEEE Computer Society Press, Los Alamitos
26. Vassilev T, Spanlang B, Chrysanthou Y (2001) Fast cloth animation on walking avatars. Comput Graph Forum 20(3):261–267
27. Volino P, Thalmann NM (1994) Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. Comput Graph Forum 13(3):155–166
28. Volino P, Courchesne M, Thalmann NM (1995) Versatile and efficient techniques for simulating cloth and other deformable objects. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques. ACM Press, New York
29. Zachmann G, Felger W (1995) The BoxTree: enabling real-time and exact collision detection of arbitrary polyhedra. In: Proceedings of the workshop on simulation and interaction in virtual environments, Iowa City, Iowa, 13–15 July 1995. University of Iowa Computer Science Department, Iowa City, Iowa
30. Zachmann G (1998) Rapid collision detection by dynamically aligned DOP-trees. In: IEEE 1998 virtual reality annual international symposium, Atlanta, Ga., 14–18 March 1998. IEEE Computer Society Press, Los Alamitos
31. Zöckler M, Stalling D, Hege H (2000) Fast and intuitive generation of geometric shape transitions. Vis Comput 16:241–253

Photographs of the authors and their biographies are given on the next page.

THOMAS LARSSON is a lecturer and PhD candidate in the Department of Computer Science and Engineering at Mälardalen University, Sweden, where he also received his BS and MS in computer engineering in 1996 and 1999 respectively. His main research interests are collision detection, deformable bodies, real-time computer graphics algorithms and virtual reality.

TOMAS AKENINE-MÖLLER is an assistant professor at the Department of Computer Engineering at Chalmers University of Technology, Sweden. He has received a MS in computer science and engineering from Lund University of Technology, and a PhD in computer graphics from Chalmers University. He is the co-author with Eric Haines of the book *Real-Time Rendering*, and his main research interests are rapid and realistic real-time rendering, interactive ray tracing, spatial data structures, and algorithms for future graphics hardware.