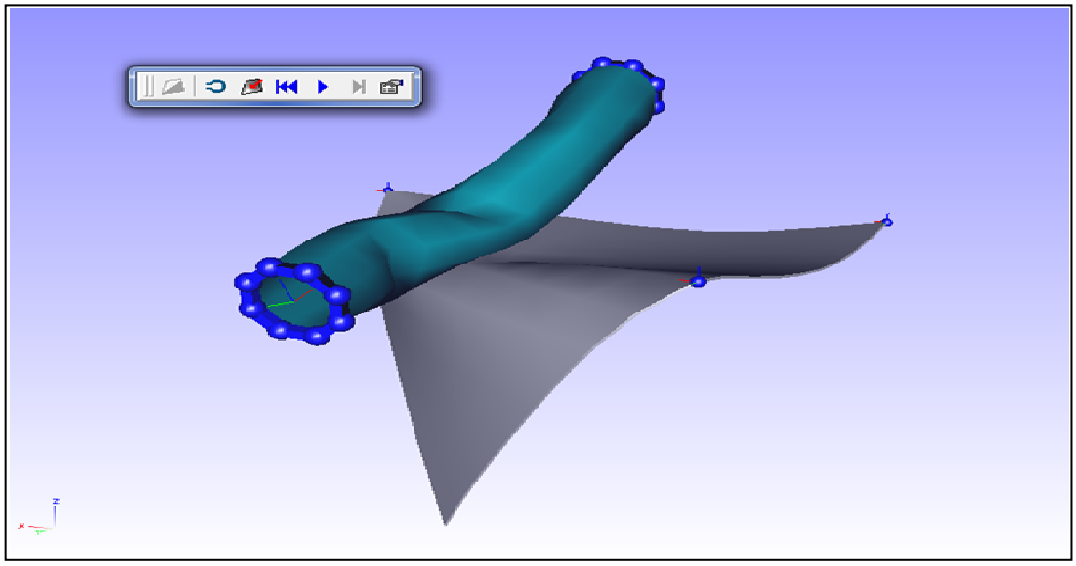
CHALMERS



Adaptive Bounding Volume Hierarchies For Deformable Surface Models

Master of Science Thesis in Complex Adaptive Systems

FADI BITAR

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2011

Adaptive Bounding Volume Hierarchies For Deformable Surface Models

FADI BITAR

Examiner: ULF ASSARSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden November 2011

Abstract

This master's thesis explores a new mechanism for maintaining Bounding-Volume Hierarchies (BVH) of deformable surface models.

Typical algorithms found in the literature are based on refitting only a portion of the BVH, leaving sometimes a large portion of the Bounding Volumes (BVs) inaccurately representing the parts of the object they should enclose. The algorithm proposed in this thesis allows the BVH's quality to degrade as the model it represents deforms, while guaranteeing that every point in the model is contained within the BVH at all times, and thus maintaining the accuracy of any collision detection or distance measurement queries performed on the model. Through a tunable asynchronous refitting of the individual bounding volumes, the algorithm offers a computationally efficient, low memory cost solution to the accurate simulation of deformable surface models in real environments.

The decision criteria for the refitting of the BVs along with the parameters of these criteria are optimized through a Genetic Algorithm search. The resulting algorithm is shown to outperform the most commonly referred to BVH-based algorithm referred to in the literature.

Acknowledgements

I would like to thank all the people who were involved in this project and helped one way or another. First and foremost the department of Geometry and Motion Planning at the Fraunhofer-Chalmers Center (FCC) where the work for this thesis was carried out, my supervisor at FCC Evan Shellshear for his continued efforts and support, my examiner at Chalmers University of Technology Ulf Assarsson for his help with brainstorming, and Sebastian Tafuri at FCC for his much appreciated help and technical expertise in the final days of the thesis.

Table of Contents

List of Figures

I. Introduction

The widespread use of simulation and animation in industry such as in virtual manufacturing or computer-aided design software solutions requires increasingly efficient and powerful mechanisms for handling the interaction of complex-shaped objects with each other. One of the major bottlenecks in such simulations is the detection of collisions among different objects, or the collision of a flexible object with itself. Moreover it is often desirable, as is the case in this thesis, to maintain an accurate distance measurement between the different simulated objects, and therefore it is essential for any proposed solution to efficiently handle both distance measurement and collision detection queries.

In simulation, objects are often represented as triangulated meshes. Given that such meshes often range between thousands and millions of vertices, performing distance tests between every single pair of triangles results in $O(n^2)$ complexity per frame for n triangles, which is clearly inefficient.

One of the most prominent approaches is to construct a Bounding-Volume Hierarchy (BVH) to approximate the shape of the triangulated mesh at iteratively finer levels. The result is a hierarchy of Bounding Volumes (BVs) where the root node encloses all triangles in the mesh, and every leaf node encloses a single triangle. Distance measurement and collision detection queries are then performed by traversing the hierarchy, culling away large portions of the object that do not need to be tested.

BVHs became popular in their use with rigid models, where the objects' only movements in space are combinations of translations and rotations. In this case, the BVHs for each object can be constructed before the start of the simulation, and then undergo the same transformations as the objects they represent. In most implementations this often implies computing only the new position of the root node of every BVH, which makes this approach highly efficient.

However in the context of flexible models, a BVH needs to be updated as the object deforms. Since it is highly inefficient to rebuild the entire bounding volume hierarchy at each frame, an algorithm that can efficiently maintain the BVH such that it minimizes the cost of subsequent distance measurement and collision detection queries is highly desirable.

Other approaches that have been used in the literature include spatial hashing, stochastic collision detection using particle swarm optimization, kinetic data structures, and other techniques that will be detailed in later sections.

This thesis covers the research undergone for the design and implementation of a novel algorithm for adaptive BVHs that can efficiently be used with flexible models. The

outcome of this thesis will lead to the improvement of the software libraries used at the core of Fraunhofer-Chalmers Center (FCC)'s interactive virtual assembly planning and robotics simulation software solution.

II. Previous Work

It is often the case for collision detection algorithms to have separate mechanisms for what is referred to as Broad-phase and Narrow-phase collision detection. The purpose of the broad phase is to cull away any objects that are certain not to be colliding, while the narrow-phase inspects two objects that might be colliding in an attempt to determine the exact location and time of the collision. Distance measurement queries on the other hand often require an accurate measurement of the shortest distance separating two complex shaped objects, regardless of its magnitude. Broad-phase techniques are therefore of lesser use in the context of this project. This section will hence focus on Narrow-phase techniques which could be potential solutions to the problem at hand.

Larsson and Akenine-Möller [Lar01] propose a BVH-based algorithm where only the nodes constituting the upper half of a BVH are refitted, while keeping other nodes unmaintained. In the event that the collision detection query reaches a node that is still unmaintained, the query is interrupted while the sub-tree stemming from that node is refitted. Later, the authors update their algorithm [Lar06] to track the “active” regions of a BVH, i.e. the BVs that have recently been traversed during a query, and only refit the active nodes, starting with the nodes at the bottom of the hierarchy.

Metzger, Kimmerle and Etzmuss [Met03] propose an “oriented inflation” mechanism to extend the BVs such that they still contain the triangles they enclose even after the deformation of an object. The inflation is based on a prediction mechanism which computes a velocity cone, prunes regions of low velocity, and inflates BVs according to the prediction of where particles will be in the next time step.

Sud et al [Sud06] argue that BVHs do not produce efficient enough culling when objects are in close proximity, and they propose the use of second-order discrete Voronoi diagrams. The closest neighbors of a triangle are its neighboring triangles in a Voronoi diagram, and hence computing the Voronoi diagram returns the closest neighbors of a triangle without resorting to any hierarchy traversal. The proposed method is shown to be more efficient in many-body simulations and self-collision detection than BVH-based algorithms when the number of object models involved is large, although suffers from overhead due to the Voronoi diagram computation which limits its performance for smaller environments.

Other techniques used in the literature include deformable spanners [Gao04], special hashing [Tes03], second-order geometrical motion shape matching for believable motion [Mül05, Spi07], and Particle Swarm Optimization (PSO) [Tia06].

III. Algorithm Description

As discussed in the previous section, BVHs offer the most promising framework for efficient algorithms. Moreover in the context of simulations involving both rigid and flexible objects, given the fact that BVHs have proven to be highly efficient with rigid objects, it is highly desirable to employ a similar mechanism for flexible models so as to allow the implementation of generic Model-to-Model distance measurement and collision detection queries, where every model would be represented by a BVH regardless of whether it is rigid or flexible. This type of setting would largely decrease the complexity of the overlaying framework through which the queries are executed.

1. BVH overview

A BVH consists of a hierarchy of Bounding Volumes, where each leaf BV encloses a single triangle of the object model, while inner BVs serve as parents to two (or more, depending on the degree of the tree) BVs, containing the triangles enclosed in both child nodes, while not necessarily containing the child BVs themselves. The root node of the BVH is a BV that encloses all triangles of the object model.

BVHs are highly efficient structures for distance measurement and collision detection, as they inherit the search efficiency inherent to tree structures. A distance measurement query between two BVHs M_1 and M_2 first calculates the distance between the root nodes of the two models, then traverses the largest of the two nodes. The distances between the smaller root node and the child nodes of the larger root node are computed, and the child node whose distance is smallest replaces its parent in the recursion. Potentially half of the triangles of a BV are culled away at every recursion step in a distance measurement query.

Another way to look at BVHs is an iterative approximation of the shape of an object, where each level in the hierarchy represents a representation of the object model with finer detail, until the leaves are reached, beyond which a finer detail can only be found through the triangulated mesh representation of object model itself. The root of the BVH can be considered a rough approximation of the convex hull of the object whose shape is determined by the type of BV used in the implementation. The set of BVs at a given level in the hierarchy represent the shape of the object with a level of detail directly related to the depth at which these BVs lie in the hierarchy.

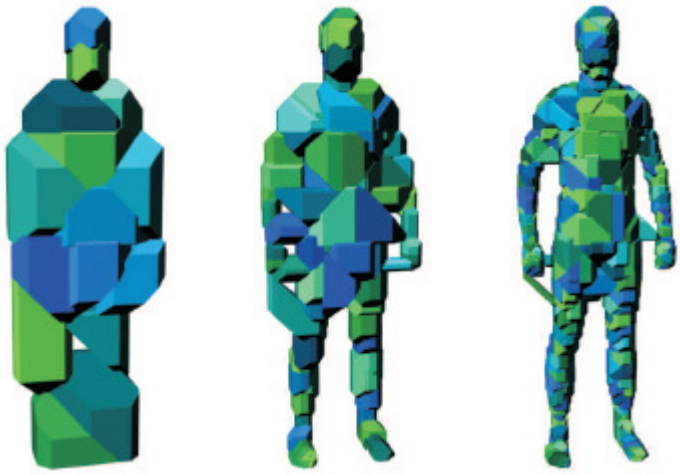


Figure 1: Shape approximation of a human model at three different hierarchy levels with increasing depth and level of detail from left to right

It is therefore highly efficient to perform collision detection using BVHs, as the lack of collision between two sets of BVs belonging to two hierarchy levels of two different object models implies the absence of collision between the objects themselves. Moreover, in the presence of collision between only a limited number of BVs, as opposed to the entire two levels of the hierarchy, all non-colliding BVs and their respective sub-trees can be culled away, resulting in a reduction in the number of tests to be performed at deeper levels in the hierarchy. Through this mechanism, the cost of a collision detection test can be reduced considerably compared to the brute-force approach which yields a computational complexity of $O(n^2)$ for object models made up of n triangles.

This concept of disregarding portions of the object model depending on the result of the queries on the BVs which enclose these portions is extended and used in the context of distance measurement queries. When querying for the shortest distance between two complex-shaped objects (or the two points which connect the two objects with the shortest distance), the BVs that are most likely to contain these points are considered first and their sub-trees are traversed.

It should be noted that all code in this thesis was developed using the Proximity Query Package (PQP), an open source package which is used, modified and improved at FCC. In the PQP model used in this thesis, BVHs have a binary tree structure.

2. Choice of Bounding Volumes

BVHs can be constructed using several types of Bounding Volumes (BVs). All BVs of a BVH are constructed similarly, and therefore a single BVH cannot be comprised of more than one type of BVs. It is generally the case that all BVs within a simulation belong to the same type. Since the choice of BV can have a significant impact on the performance of

the distance and collision queries, different BV types can be assessed and compared given the following criteria:

- *Tight fitness:* A BV should closely approximate the volume that the triangles it encloses occupy. Since the location of a BV is an approximation of that of the triangles it encloses, the tighter a BV is around its inner triangles, the less likely it is to result in a collision false positive during the hierarchy traversal.

- *Construction complexity:* Given a set of triangles, it is desirable to construct a BV that encloses these triangles while incurring the least computational cost possible for construction. In the context of rigid objects where the construction of the BVs takes place before the start of the simulation, this constraint can be relaxed, however with flexible objects, BVs often have to be recomputed during the course of the simulation, and therefore a computationally "cheap to construct" BV is highly desirable.

- *Memory footprint:* A binary-tree BVH of an object model would be comprised of as many BVs as twice the number of its triangles. As object models are often comprised of thousands of triangles, a BV type with a reduced memory footprint presents an advantage.

- *Distance calculation:* One of the key purposes of using BVHs is the reduction of the computational cost of distance measurement and collision detection queries. Given that collision detection queries are in essence distance measurement queries, it is desirable to use BVs that offer low complexity BV-to-BV distance queries during the hierarchy traversal.

Typical types of BVs used in the literature are the following:

- *Axis Aligned Bounding Boxes (AABBs):* This type of BV is aligned with the coordinate system of the BVH and consists of a three dimensional rectangle. It can be stored as the coordinates of a single point along with a height, width and length. Given a set of triangles, it is enough to determine the boundaries of the projection of the vertices of the triangles along the coordinate axes to build a BV containing these triangles. Moreover, distance calculations between two such BVs can be done in few operations. However, AABBs suffer from being the least tightly fit BVs covered in this section.

- *Oriented Bounding Boxes (OBBs):* OBBs are AABBs that are oriented along the axes of largest variation of the triangles they enclose. Their construction is similar to that of AABBs, except that it is usually preceded by Principle Component Analysis (PCA) to determine the new coordinate system along which the OBB will be oriented.

- *K-Discrete Oriented Polytopes (k-DOPs):* k-DOPs are OBBs with beveled edges. The parameter k determines the shape of the k-DOP, where 6-DOP is simply an OBB, while a 10-DOP has all of its vertical edges beveled, an 18-DOP has all edges beveled, a 26-DOP has all edges and corners beveled, and so on. Clearly, as k increases the complexity involved in the construction of the BV increases as well, while the fitness of the BV around the triangles it contains improves. Distance calculations between k-DOPs increase in complexity as k increases.

- *Minimum Bounding Spheres:* Many algorithms exist to compute the minimum bounding sphere for a set of points (the EPOS algorithm [Lar08] runs in $O(n)$ time). Given their storage as the coordinates of a single point and a radius, spheres have the smallest memory footprint of the BVs listed in this section, in addition to having the least complex distance measurement queries, which makes them quite widely used. However, spheres often provide a very loose fit, which incurs an increased number of unnecessary collision detection queries.

- *Rectangle-Swept Spheres (RSSs):* An RSS is a two-dimensional rectangle with a radius. The construction of an RSS around a set of triangles resembles that of an OBB, except only the two largest axes of variation are determined using PCA, followed by an additional step where the bounds of the rectangle and the radius of the RSS are determined. The use of RSSs attempts to balance the tightness of k-DOPs with the simplicity of sphere-to-sphere distance measurements.

For the purposes of this thesis, given the advantages of RSSs in terms of distance computation, tight fitness and memory usage, all BVs will be constructed as RSSs. The use of RSS was also advantageous from an implementation point of view since the existing library for rigid body simulations uses RSS-shaped BVs, and so the construction and distance test routines for this type of BV are already available within the library. Typically, an RSS BV is represented by a transformation matrix which can be used to compute the location and orientation of the coordinate system of the BV, the length and width of the base rectangle of the RSS, along with the radius of the RSS.

3. Choice of hierarchy construction strategy

Two main approaches exist to building a BVH: Top-down and Bottom-Up.

The top-down strategy starts off by constructing a BV around the entire object. A split axis is then chosen (using PCA, Projection Pursuit, etc. [Eri04]) along with a split point (mean of all projections along the split axis, midpoint of the range of projection along the split axis, etc. [Eri04]). The triangles are then split into two separate groups, for each of which a BV is constructed then split again. The hierarchy is complete when all leaf nodes contain single triangles. While the choices of split axis and split point constitute a topic of research on its own, it is usually the case that PCA along with the mean or midpoint of the projections of the vertices are used as the iterative splitting mechanism.

It should be noted that the top-down construction strategy is highly biased towards the initial configuration of the object, and does not take into account connectivity information of the triangles. As a result, the movement and deformation of a flexible object might trigger a necessary restructuring of the entire BVH if two triangles were initially adjacent but end up very far apart.

The bottom-up hierarchy construction strategy begins by constructing a BV around every single triangle, and then using connectivity information along with other heuristics, a parent BV is created for a set of neighboring BVs. Since binary-tree BVHs are typically used, it is generally the case that a parent BV is created for two neighboring BVs. Higher degree trees can also be used, in which case more BVs can be combined in the construction of a parent BV (e.g. eight neighboring BVs for an octree, and so on). The process is iterated until a root node has been created around all the triangles of the object model.

This approach, unlike its top-down counterpart, is not biased towards initial configuration and takes advantage of connectivity information in the hierarchy construction. Therefore, this approach removes the need to restructure the BVH in mid-simulation.

However, the literature offers very few solutions to bottom-up construction of hierarchies based on meshed triangulations. In fact, in most of the collision detection literature related to flexible objects, the bottom-up construction of the hierarchy is regarded as superior, but hardly ever used.

With that in mind, finding a good set of heuristics for building any BVH bottom-up and obtaining a well-balanced tree proved to be a more challenging task than initially thought. Nevertheless, a good deal of time was spent on that task, the details of which can be found in the Appendix. The remainder of this section will assume that the BVH has been constructed bottom-up according to the algorithm described in the appendix, and that the resulting tree is well-balanced.

Now that the choices for the initial construction of the BVH and the type of BV used have been justified, the mechanism making the BVH adaptive to flexible surfaces can be explained in detail.

4. Dynamic BV update

The challenge in devising a good updating/refitting schedule for the BVs of a BVH lies within the fact that the moment an object starts deforming, its BVH is no longer guaranteed to represent it correctly; and therefore any distance measurement or collision detection query performed on this BVH might not be accurate. In most of the literature, collision detection algorithms involving BVHs for flexible object models focus on obtaining a good guess of which regions of the BVH need to be refitted, such that any subsequent query on the BVH returns an accurate result.

In the Larsson and Akenine-Möller algorithm [Lar01][Lar06] all BVs in the upper half of the BVH are refitted at every step, while only the most recently visited nodes in the lower branches of the hierarchy are refitted. Moreover, if a node that has not been refitted is reached during a query, the processing of the query is stalled until the current node and its children are refitted. Gao et al [Gao04] devise an entire framework to predict which sub-tree of the BVH needs to be refitted.

The drawback of such methods is that during the course of the simulation the BVH is almost never entirely correct. From the moment the object starts deforming, the BVH only partially represents the object in its "active" regions, i.e. the nodes that are being traversed in the queries. In the context of multiple objects where an accurate pair-wise distance measure needs to be maintained, such algorithms are likely to label a large portion of the BVH's BVs as active, and therefore result in a large computational cost of refitting all active nodes. During the refitting of a node, the BV is recomputed based on the current location of the triangles it should enclose. If most nodes in the hierarchy are labeled as active then a large portion of the hierarchy is being recomputed.

The Metzger et al algorithm [Met03] keeps track of the velocity of motion of the triangles in order to predict their location during the next time step, and attempts to extend the BVs such that the probability of all BVs in the hierarchy still properly representing the triangles they enclose is maximized. Since the algorithm relies heavily on prediction, it provides no correctness guarantee for the BVH.

4.1. Choice and justification of criteria for the algorithm design

In this thesis an alternative algorithm is presented, and satisfies the following criteria:

- *The BVH is correct at all times:* Unlike methods that rely on refitting only a portion of the BVH, the algorithm presented in this thesis should guarantee that all BVs in the BVH correctly represent the triangles they enclose at any point in the simulation.

- *No change is needed to the query routines:* In order to reduce the cost of swapping memory during the simulation, and minimize the development effort needed to implement the algorithm within any BVH framework, the algorithm should use the same distance measurement and collision detection queries used for rigid models. The refitting of the BVs should never interrupt these queries.

- *The algorithm has a low computational cost:* Most importantly, the proposed algorithm should have an improved running time performance compared to other solutions proposed in the literature.

4.2. Data structure modifications

In order to satisfy the above specified criteria, the proposed algorithm relies on iteratively degrading the tight fitness of the BVs such that they are guaranteed to properly represent the triangles they enclose even in the worst case. A separate mechanism then controls the asynchronous refitting of the BVs that are deemed too degraded. Note that since the refitting process is asynchronous, the refitting of a node does not imply the refitting of any of its children.

The proposed algorithm can be sub-divided into three phases: the error-propagation phase, the BV refitting phase, and the query phase. The data structure used for the representation of triangles in deformable object models was augmented for the purposes of the error-propagation phase. Every triangle keeps track of two motion vectors:

- the largest displacement vector of any of its vertices during the last frame, referred to as “latest displacement vectors”.
- the largest displacement vector of any of its vertices since the last refitting of the leaf BV that encloses it, referred to as the “overall displacement vectors”.

At every frame, each triangle gets updated to its new location and updates its displacement vectors, which it forwards to its corresponding leaf BV. Using this information, the leaf BVs can be modified to contain the triangles in their new positions, and provide hints about the new location of the triangles further up the hierarchy.

The data structure used for the representation of the BVs has also been modified. An RSS BV is typically represented by length, width, and radius, in addition to a transformation matrix which determines the origin and orientation of the RSS. For the purpose of this algorithm, an additional radius measure was added to BV, whose value is equal to the original radius of the BV when it was last recomputed. This radius measure will be referred to as the “original radius”, while the radius measure that will be dynamically

modified throughout the simulation will be referred to as the “current radius”. The current and original radii are set equal whenever a BV is recomputed.

4.3. Algorithm details

The first step of the algorithm consists of building the BVH based on the connectivity information of the triangles (bottom-up construction). At this point all BVs are in their initial state and the current radius is equal to original radius for all BVs. Once the BVH construction is done, the simulation can start.

As mentioned earlier, the algorithm can be split into three phases: the error-propagation phase, the BV refitting phase, and the query phase. The error propagation phase proceeds as follows:

1. Update the position information for all vertices, while allowing triangles to keep track of the displacement vector of each of their vertices.

2. For each leaf BV, measure the largest norm $D_{overall}$ of the overall displacement vectors of the triangle it encloses.

3. Test the condition:
   $current\ radius - original\ radius < D_{overall}$
   If it is satisfied, set:
   $current\ radius = original\ radius + D_{overall}$

   Note that this operation guarantees that the BV will contain the triangle even in the worst case, where the largest displacement occurs in a direction normal to the surface of the RSS on a vertex that was positioned on the surface of the RSS.
   Note that using the largest norm of the overall displacement vectors is a mechanism to reduce the inflation (i.e. incrementing of the current radius) of the leaf BVs if vertices are oscillating back and forth. The overall displacement vector in this case would not grow to be too large, hence keeping the leaf BVs relatively tightly fit around their corresponding triangles.

4. Propagate the error up the hierarchy. Every leaf BV propagates the largest norm D_{latest} of the latest displacement vectors of the triangle it encloses to its parent. Note that since the overall displacement vectors are reset when the leaf BVs are recomputed, the overall displacement vectors provide very little information to BVs that are higher up the hierarchy, and therefore the latest displacement vectors are used instead.

5. Every non-leaf BV increases its current radius by the maximum norm D_{max} propagated to it by its children. This operation also guarantees that non-leaf BVs contains all vertices of the triangles they enclose even in the worst case movement. After inflation, the BV propagates D_{max} to its parent. This step is repeated until the root is reached, and every BV in the hierarchy has been inflated by the maximum displacement propagated to it by its children.

The inflation of the root marks the end of the error propagation phase. At the end of this phase, the degradation of the quality of the BVs by loosening their tight fitness around the triangles they enclose guarantees that the BVH correctly represents the deformable object model.

Since the quality degradation introduces an increase in the probability of a BV being traversed during a distance measurement or collision detection query, it is also important to maintain the quality of the BVH by occasionally refitting some of its BVs. Refitting all BVs in the hierarchy is computationally inefficient, while refitting none takes its toll on the distance query time, since a BVH that has not been maintained in k frames contains all the locations of all vertices of the model during those k frames, and therefore a distance measurement query would end up traversing many unnecessary BV nodes before returning an accurate result. With that in mind, it is important to devise a good strategy for the refitting of the BVs while satisfying the requirements on the algorithm listed above.

During the BV refitting phase, each BV in the model is tested for a set of criteria which, if satisfied, would mark it as unfit. All unfit BVs are then recomputed. It is however important to choose an appropriate set of criteria against which to test the model's BVs, as to minimize the computational cost of the BV refitting phase, while maintaining an acceptable quality for the model such that distance measurement queries are performed efficiently. The choice of the set of criteria will be discussed in depth in the next section. After the BV refitting phase, the BVH correctly represents the object model, and can be used in distance measurement or collision detection queries similarly to a newly constructed BVH.

4.4. The BV fitness criteria tests

Different mechanisms can be used to determine if a BV is unfit. Typically, it would be of interest to measure how tightly fit the inflated BV is around the triangles that it encloses. While such methods might work well with BVs with a relatively small number of triangles, their computational cost increases drastically for BVs in the upper levels of the hierarchy. Such methods are also problematic as they rely on the efficiency of the BV construction algorithm for randomly-shaped geometries. A newly constructed BV might be labeled as unfit simply as a result of an inefficiency in the BV construction algorithm,

and so such a BV would be recomputed at every frame. Moreover, the task of determining the fitness of a BV around a randomly-shaped geometry remains a challenge in itself. It is therefore unrealistic to base the BV refitting decisions on such heuristics.

The following measures were developed and shown to capture the necessary information needed to develop a good decision-making mechanism for the BV refitting phase. Note that the final algorithm will use these three measures, or tests, as building blocks in the final BV fitness test.

4.4.1. Inflation Factor Test

As mentioned in earlier sections, during the inflation of the BVs, the algorithm keeps track of both the original radius of the RSS and the current radius of the inflated RSS. This can be used as a means to monitor the movement of triangles within the bounds of their corresponding BVs. A BV whose current radius is large compared to its original radius should in all probability be recomputed.

Calculating the factor by which the BV radius increases:

$$Inflation\ factor = \frac{current\ radius - original\ radius}{original\ radius}$$

If the inflation factor in BV radius is larger than a given threshold p_{max}, then the BV is marked to have passed this test.

4.4.2. Recent Traversal Test

This test, as the name indicates, marks BVs that have been traversed during the last distance measurement or collision detection query. The BVH keeps track of which BVs are being traversed during the queries of a given frame, and resets the list of traversed BVs at every frame.

4.4.3. Child Nodes Distances Test

The third and final test also uses information obtained during the distance measurement queries.

Consider a distance measurement query between two BVH models M_1 and M_2. The query has reached a stage where the distance between BV B_1 from model M_1 and BV B_2 from model M_2 has been computed. The distance query now proceeds down the hierarchy of

one of the models (for instance M_1) by measuring the distance between B_2 and the child nodes of B_1: A_1 and C_1. The distance measures will be used by the query to decide which node to traverse next.

Let D be the distance between B_1 and B_2:

$$D = Distance(B_1, B_2)$$

Let R' be B_1's current radius, and R, B_1's original radius.
Let D_a be the distance between A_1 and B_2, and D_c the distance between C_1 and B_2.

$$D_a = Distance(A_1, B_2)$$
$$D_c = Distance(C_1, B_2)$$

Now let α and β be defined as follows:

$$\alpha = D_a - D$$
$$\beta = D_c - D$$

The Child Nodes Distances Test proceeds as follows: if B_1 is inflated by at least a given factor, then measure α and β.
If both α and β are both larger than a portion k of the amount of inflation of B_1, then the BV has passed the test. The condition for passing the Child Nodes Distances Test can be summarized in the following logical statement:

$$\left(\frac{(R' - R)}{R} \geq \Upsilon \wedge \left(\alpha > k(R' - R) \wedge \beta > k(R' - R)\right)\right)$$

ϒ and k are positive real numbers, and k < 1.

The rationale behind this test is that if the inflation level of a BV is such that its original radius seems to be a better fit than the current radius, based on the estimated location of its child nodes, then the BV's quality is too degraded.

Unlike the Inflation Factor Test, the Child Nodes Distances Test is not performed on all BVs in the BVH, as the test requires information that is only available during the traversal of the BVH by the distance measurement queries. A BV that has passed the Child Nodes Distance Test has implicitly also passed the Recent Traversal Test. Additionally, the Recent Traversal Test and the Child Nodes Distances Test are sensitive to the outcome of the distance measurement queries, allowing them to guide the decision of refitting a BV, whereas the Inflation Factor Test only relies on information contained within the BV itself.

All three tests introduce no added complexity to the overall algorithm, or changes to the distance measurement queries. The choice of parameters governing all three tests and the use of the tests in the algorithms are discussed in the following section.

4.5. Defining an unfit BV

In order to optimize the tradeoff between the time spent rebuilding unfit BVs and the time lost on inefficient distance queries due to a poorly maintained BVH, all three tests described in the previous section can be combined, and their parameters tuned to give the best result.

Assuming every test returns a Boolean value, let A be outcome of the Inflation Factor Test for a given BV, B the outcome of the Recent Traversal Test, and C the outcome of the Child Nodes Distances Test. These three variables can be combined in a number of ways using logical operations, each combination using one or more variables.

Knowing that C is implicitly B∧C, all logical combinations of one or more of the three variables can be listed, then equivalent combinations are removed. This results in the following list of seven logical combinations, also referred to as "fitness criteria".

i. A: The BV has reached an inflation factor that is larger than the threshold value p_{max}

ii. B: The BV has been traversed during the last distance query

iii. A∧B: The BV has been traversed during the last distance query and has reached an inflation factor larger than p_{max}

iv. B∧C: The BV has been traversed during the last distance query and has passed the Child Nodes Distances Test

v. A∧B∧C: The BV has been traversed during the last distance query, has reached an inflation factor larger than p_{max}, and has passed the Child Nodes Distances Test.

vi. A∨(B∧C): The BV has reached an inflation factor larger than p_{max} or has been traversed during the last distance query and passed the Child Nodes Distances Test.

vii. (A∧B)∨(B∧C): The BV has been traversed during the last distance query and has either reached an inflation factor larger than p_{max} or passed the Child Nodes Distances Test.

The value of A depends on the Inflation Factor Test parameter p_{max}, while the value of C depends on the Child Nodes Distances Test parameters k and γ. Since the values of these three parameters influence the running times of six of the seven fitness criteria listed, conducting a conclusive comparison of their performance when used to decide whether or not a BV should be refitted is near impossible. It is therefore possible to run a genetic algorithm that would explore the parameter space for each of the given fitness criteria and determine the set of parameters and logical combination that would optimize the running time of the simulation.

IV. Simulation Setup and Results

The optimization of the decision-making criteria for the refitting of the BVs has been carried out using GALib, an open-source genetic algorithm implementation in C++. For each of the fitness criteria listed in the previous section, a separate execution of the genetic algorithm was carried out in order to determine the set of parameters for which the simulation running time is optimal.

The parameters of the genetic algorithm are the following:

- Population Size: 30
- Number of Generations: 50
- Population overlap percentage: 50%
- Crossover probability: 90%
- Mutation Probability 1%
- Range of p_{max}: [0 10]
- Range of k : [0 1]
- Range of Υ: [0 1]

Each individual in the population is evaluated according to the simulation running time. For that purpose, a test simulation that uses different types of deformable models was set up, and proceeds as follows:

- Two deformable object models are read and built.
- At every frame, both models are deformed, and the BVH is updated through error propagation and BV refitting.
- In between frames, the distance between the two models is measured.
- The time spent on the update of the BVH and the distance query is measured for all frames then returned when both models have completed the set of deformations.
- The test is repeated five times for different combinations of object model pairs, and the total time spent on all five simulation scenarios (which does not include the time spent on building the BVH) is set as the score of the individual.

The five pairs of deformable object models used in the evaluation simulations are shown in the below figures:

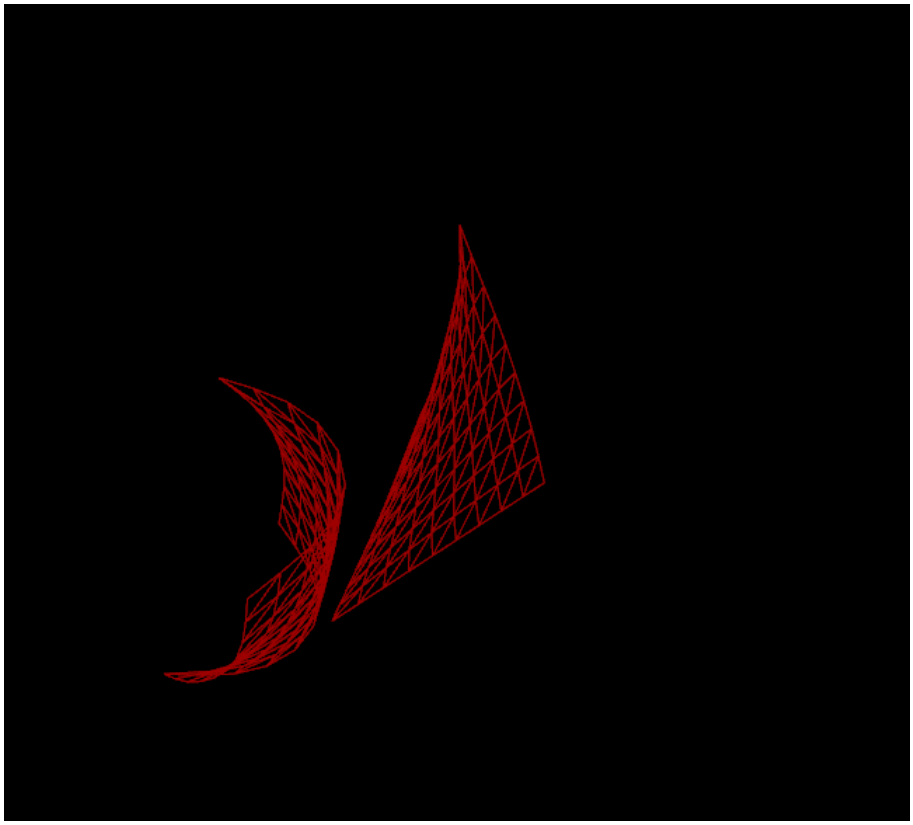


Figure 2: The two models used in the first simulation scenario: Two plates, one of which is getting twisted

Figure 3: The two models used in the second simulation scenario: A cable and a plate

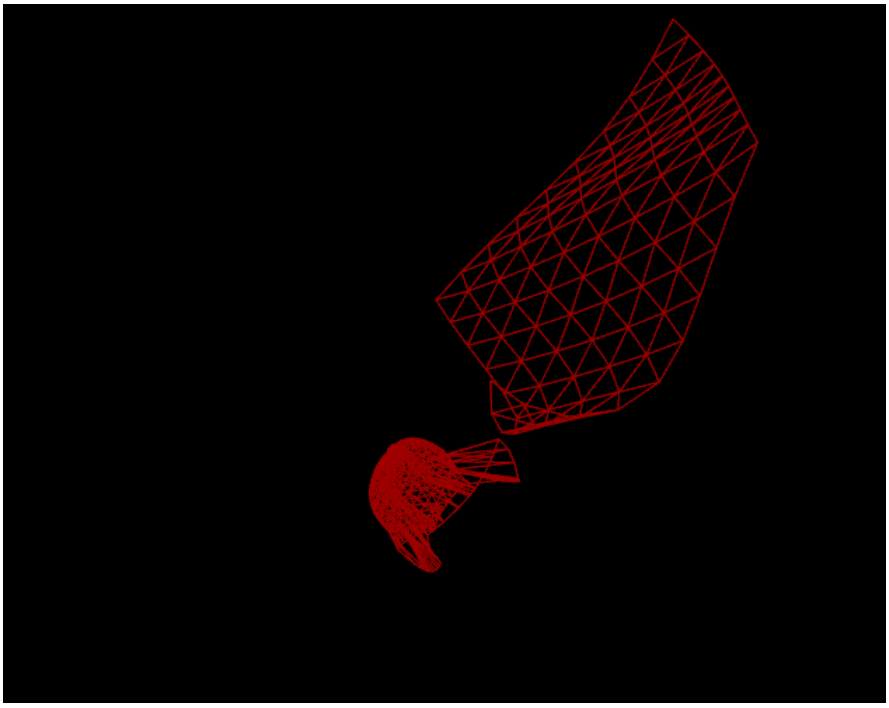


Figure 4: The two models used in the third simulation scenario: A helmet and a sheet folding on itself

Figure 5: The two models used in the fourth simulation scenario: A helmet and a cable

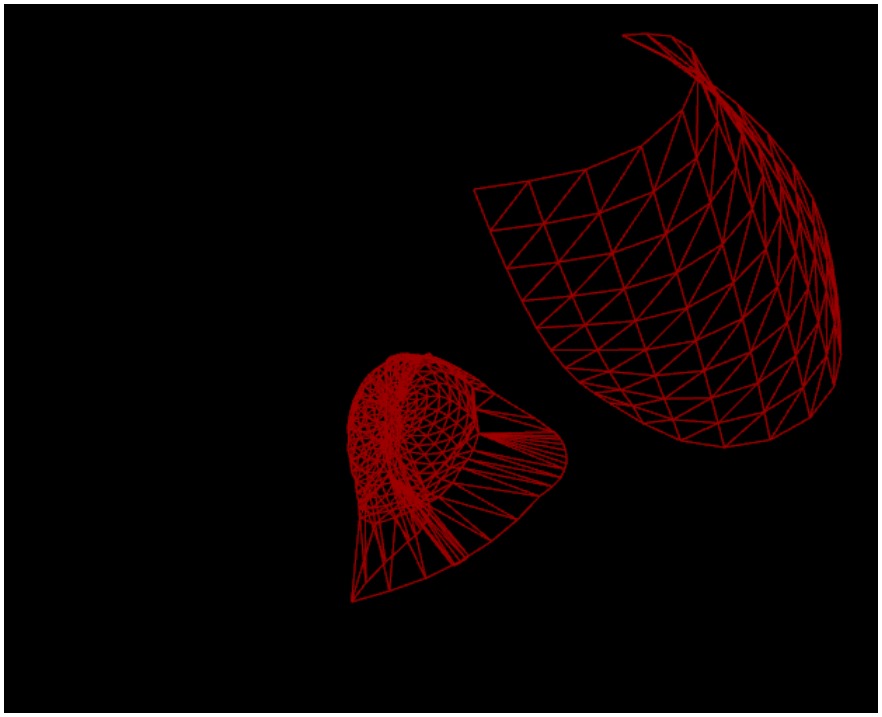


Figure 6: The two models used in the fifth simulation scenario: A helmet and a plate

The table below shows the number of vertices, triangles and deformation frames in each of the object models used in the simulations:

Object Model	Number of Vertices	Number of Triangles	Number of Frames
Plate	100	162	62
Twisting Plate	100	162	218
Cable	100	180	127
Helmet	385	710	20
Folding Sheet	100	162	29

Table 1: Object properties for the models used in the simulations

The helmet in the simulations is used as a generic deformable model with a custom-shaped geometry. Note that for each simulation scenario, the simulation lasts for the maximum number of deformation frames of either of its objects. For instance, the Helmet and Plate scenario lasts for 218 frames, in the first 62 of which both objects models deform simultaneously, while the helmet completes its deformation during the remainder of the deformation frames and the plate remains rigid.

The following table shows the best results for each fitness criteria along with the parameter values for which these results are achieved:

Fitness Criteria	p_{max}	k	Υ	Simulation time
A	1.884184	-	-	0.715
B	-	-	-	0.842
A∧B	1.498741	-	-	0.388
B∧C	-	0.333928	0.053803	5.12
A∨(B∧C)	3.383383	0.416571	0.111757	0.531
A∧B∧C	1.967193	0.221836	0.002197	4.557
(A∧B)∨(B∧C)	1.359579	0.891981	0.590829	0.332
Refit All	-	-	-	2.839
Refit None	-	-	-	12.485

Table 2: Comparison of results for all considered fitness criteria given the parameters yielding the best simulation times

As the results show, the best fitness criteria (A∧B)∨(B∧C), with the parameters listed in the same row. According to this result a BV is allowed to reach an inflation factor of ~0.59 before it is ever considered for refitting, and an inflation factor of ~1.36 still does not guarantee that the BV will be refitted, unless it is traversed§ during a distance measurement query. Comparing the winning criteria to A∧B, it can be noted that the added value of the "∨(B∧C)" in the winning criteria is small, and perhaps does not introduce a significant enough improvement to the A∧B criteria.

As an outcome to this simulation, the criteria (A∧B)∨(B∧C), A∧B and A∨(B∧C) are overall superior to other choices, with the first of the criteria outperforming the others.

V. Discussion

The algorithm presented in this thesis has been shown to be highly efficient. Ideally, it is desirable to compare its performance with that of prominent algorithms in the literature. An attempt at contacting Thomas Larsson to obtain his code for benchmarking purposes was not successful, as the code according to Larsson was not available in any form suitable for distribution.

The Larsson and Akenine-Möller algorithm [Lar06], which according to its authors outperforms the algorithm they have previously proposed [Lar01], marks all BVs traversed during the previous distance query as “active” then proceeds to rebuild all BVs marked as active starting with the BVs that are deepest in the hierarchy until the root node is reached.

In this scheme, nodes that are traversed during a distance measurement query which were not marked as active during the previous query do not accurately represent the triangles that they enclose and therefore need to be refitted. This operation interrupts the distance measurement query.

Additionally, in the Larsson and Akenine-Möller algorithm [Lar06], the root node is rebuilt at every frame, which can be computationally expensive for object models with a large number of vertices.

The Larsson and Akenine-Möller algorithm [Lar06] can be thought to be equivalent to the algorithm proposed in this thesis, where the fitness criteria is reduced to passing the Recent Traversal Test, without performing error-propagation. While the possibility of interrupting distance measurement queries to rebuild unfit BVs has not been implemented, it is reasonable to extrapolate the results of the simulation time of the proposed algorithm in this thesis under fitness criteria B to represent the running time of an implementation of the Larsson and Akenine-Möller algorithm [Lar06].

As the results in the above table clearly show, the proposed fitness criteria along with the error-propagation phase allow for much faster simulation times than a BV refitting schedule based solely on active nodes would, and therefore the conclusion can be made that the algorithm proposed in this thesis largely outperforms the Larsson and Akenine-Möller algorithm [Lar06], one of the most commonly used and benchmarked against algorithms in the literature.

VI. Conclusion and Future Work

This thesis covers the development and implementation of a new approach to updating BVHs. The proposed algorithm relaxes the necessity of rebuilding the BVs of the object model by introducing an initial phase of error-propagation which precedes the BV refitting phase, and guarantees all BVs in the hierarchy to correctly contain the triangles they enclose.

With the additional phase, the algorithm introduces a tradeoff between the quality of the BVH, or the tightness of its BVs around the triangles they enclose, and the time spent on refitting BVs. By iteratively degrading the quality of the BVs, it remains possible to query the BVH for distance measurements, without the need to interrupt the distance measurement or collision detection queries to refit BVs.

Unlike other algorithms available in the literature, the proposed algorithm does not rely on the concept of “active regions”, and therefore is expected to outperform other algorithms in the context of simulations involving a large number of objects, in which case large portions of the BVHs would be labeled as “active” by other algorithms and would therefore require recalculation, incurring a large computational cost.

Possible extensions to this thesis can be divided into exploratory and continuation work. In the interest of exploration, it would be interesting to benchmark the performance of the algorithm on BVHs whose tree structure is not binary. The performance of most algorithms in the literature seems to indicate that trees of degree 4 or 8 are more efficient. This would require changes in the BVH bottom-up construction algorithm, as the current implementation is suited for binary trees. Changes to the distance measurement and collision detection queries also should be implemented as the current queries assume the BVH has a binary tree structure, and are not generalized for any tree degree.

Other possible exploratory improvements would include the testing of other fitness heuristics, and possibly their combination with the ones covered in this thesis. The fitness criteria could be further tuned to improve the overall performance of the algorithm by either replacing some of the tests included in the fitness criteria, or by adding new tests to the set currently in use.

The algorithm should also be benchmarked against a largely varying input size, so as to get a clearer idea of its performance limitations, if there should be any. It should also be compared against the algorithm proposed by Sud et al [Sud06], which uses Discrete Voronoi Diagrams to perform proximity queries, and has been shown to outperform the algorithm initially proposed by Larsson and Akenine-Möller [Lar01], in turn outperformed by their revised algorithm [Lar06], which serves as the comparison basis in this thesis.

Finally, the continuation of this work should possibly be in the development of a self-collision detection algorithm. The framework set up in this thesis should facilitate this task, as it guarantees the BVH to be correct at all times. However, as the quality of the BVs of the BVH degrades, false detection can occur more frequently in the case of self-collision, and therefore the parameters of the fitness criteria might have to be tuned again, taking into account the performance of self-collision detection queries.

References

[Aga04] Agarwal P., Guibas L., Nguyen A., Russel D., Zhang L., “Collision detection for deforming necklaces”, Computational Geometry: Theory and Applications, v.28 n.2-3, p.137-163, June 2004

[Bro01] Brown J., Sorkin S., Bruyns C., Latombe J., Montgomery K., and Stephanides M. 2001. “Real-Time Simulation of Deformable Objects: Tools and Application”, In Proceedings of Computer Animation 2001.

[Cur08] Curtis S., Tamstorf R., Manocha D., “Fast collision detection for deformable models using representative-triangles”, Proceedings of the 2008 symposium on Interactive 3D graphics and games, February 15-17, 2008, Redwood City, California

[Eri04] Ericson, C. 2004. “Real-Time Collision Detection”. Morgan Kauffmann.

[Gao04] Gao J., Guibas L., Nguyen A., “Deformable spanners and applications”, Proceedings of the twentieth annual symposium on Computational geometry, June 08-11, 2004, Brooklyn, New York, USA

[Gov05] Govindaraju N, Knott D., Jain N., Kabul I., Tamstorf R., Gayle R., Lin M., Manocha D., “Interactive collision detection between deformable models using chromatic decomposition”, ACM Transactions on Graphics (TOG), v.24 n.3, July 2005

[Hut07] Hutter M., and Fuhrmann A., 2007. “Optimized continuous collision detection for deformable triangle meshes”. In Proc. WSCG '07, 25--32.

[Lar01] Larsson T., and Akenine-Möller T., “Collision Detection for Continuously Deforming Bodies”, Eurographics 2001, A. Chalmers and T.-M. Rhyne, Eds., Eurographics, 325--333.

[Lar06] Larsson T. and Akenine-Möller T. 2005. “A dynamic bounding volume hierarchy for generalized collision detection”. Workshop on Virtual Reality Interaction and Physical Simulation. 91--100.

[Lar08] Larsson T.. “Fast and tight fitting bounding spheres”, In Proceedings of The Annual SIGRAD Conference, pages 27–30. Linköping University Electronic Press, November 2008.

[Lau06] Lauterbach C., Yoon S., Tuft D., and Manocha D. 2006. “RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs”. IEEE Symposium on Interactive Ray Tracing, 39--46.

[Luq05] Luque R., Comba J., and Freitas C., “Broad-phase collision detection using semi-adjusting bsp-trees”. In SI3D'05: Proceedings of the 2005 symposium on

Interactive 3D graphics and games, pages 179–186, New York, NY,USA, 2005. ACM Press.

[Mad09] Madera F.A., Day A.M., Laycock S.D., “A Hybrid Bounding Volume Algorithm to Detect Collisions between Deformable Objects”, Second International Conference on Advances in Computer-Human Interactions, 2009

[Mad10] Madera F.A., Day A.M., Laycock S.D., “Detecting Self-Collisions Using a Hybrid Bounding Volume Algorithm”, Third International Conference on Advances in Computer-Human Interactions, 2010

[Met03] Metzger J., Kimmerle S., and Etzmuss O. “Hierarchical techniques in collision detection for cloth animation”. Journal of WSCG 11, 2 (Feb. 2003), 322–329.

[Mir97] Mirtich, B., 1997. “Efficient algorithms for two-phase collision detection”.

[Mül05] Müller M. , Heidelberger B., Teschner M., Gross M., “Meshless deformations based on shape matching”, ACM Transactions on Graphics (TOG), v.24 n.3, July 2005

[Ove92] Overmars M.. “Point location in fat subdivisions”, Information Processing Letters, 44:261-265, 1992

[Pre99] Preis, R. “Linear time 1/2 -approximation algorithm for maximum weighted matching in general graphs”. Symp. on Theoretical Aspects in Computer Science (STACS), 259–269.

[Spi07] Spillmann J., Becker M., and Teschner M., “Efficient updates of bounding sphere hierarchies for geometrically deformable models”. J. Visual Communication and Image Representation, 18(2):101–108, 2007.

[Sud06] Sud A., Govindaraju N., Gayle R., Kabul I., and Manocha D., “Fast Proximity Computation among Deformable Models Using Discrete Voronoi Diagrams”, Proc. ACM SIGGRAPH, 2006.

[Tes03] Teschner M., Heidelberger B., Muller M., Pomeranets D., Gross M.. “Optimized Spatial Hashing for Collision Detection of Deformable Objects”, Proc. Vision, Modelling, Visualization, pp. 47–54, 2003.

[Tes05] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., and Volino P. 2005. Collision detection for deformable objects. Computer Graphics Forum 24, 1 (March), 61--81.

[Tia06] Tianzhu W., Wenhui L., Yi W., Zihou G., and Dongfeng H., 2006, “An Adaptive Stochastic Collision Detection Between Deformable Objects Using Particle Swarm Optimization”, Applications of Evolutionary Computing, Lecture notes in Computer Science, Volume 3907/2006

[Wal07] Wald I., Boulos S., Shirley P., “Ray tracing deformable scenes using dynamic bounding volume hierarchies”, ACM Transactions on Graphics (TOG), v.26 n.1, p.6-es, January 2007

[Xin07] Xinyu Z, Kim Y.J, “Interactive Collision Detection for Deformable Models using Streaming AABBs”, Visualization and Computer Graphics, 2007

Appendix: Bottom-up hierarchy construction algorithm

The construction of Bounding Volume Hierarchies from the bottom up presents many advantages when dealing with deformable object models. The lack of bias for the initial shape of the object removes the need for restructuring the BVH when two sets of triangles initially close together move away from each other. If no BVH restructuring is performed in this case, the BV containing both sets of triangles would grow unnecessarily, hence degrading the performance of any subsequent distance measurement or collision detection query on the deformable object model. The general procedure for the bottom-up construction of a BVH starts with the construction of leaf BVs around all triangles of the model, then proceeds to grouping BVs pair-wise (assuming a binary tree structure), until a root node is created, containing all triangles in the model. An important aspect of the algorithm is the choice of heuristic used to assess the quality of a possible grouping of BVs, as to optimize the division of triangles into BVs and the overall structure of the BVH.

Using triangle connectivity information during the BVH construction guarantees that triangles contained within the same BV remain within relative proximity. Therefore, the first requirement of the algorithm for the bottom-up construction of the BVH presented in this appendix is a triangulated mesh representation of the deformable object, where triangles can be queried for neighboring triangles.

For each triangle, a neighboring triangle can either be defined as a triangle with at least a single common vertex, or a triangle sharing a common edge. In the first case, a triangle can have any number of neighbors, while the second definition limits the neighbors to an upper bound of three triangles. The edge-based definition is advantageous in this case as triangles that share an edge are more likely to be close together, and therefore can be enclosed in a smaller BV.

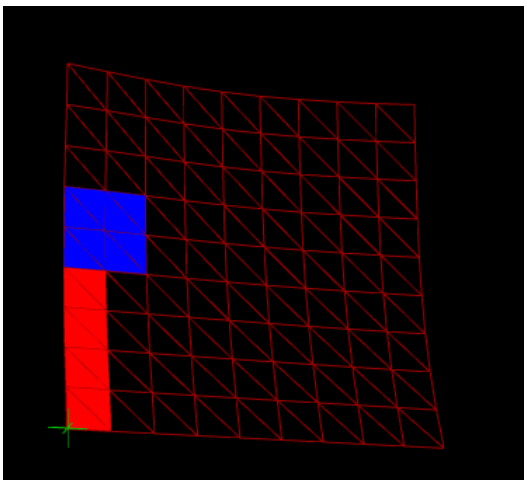


Figure 7: Example of a sub-optimal grouping between two sets of triangles highlighted in red and blue

It is important to note that in the construction of the BV, the choice of triangle groupings needs to be such that a tight BV can be constructed around it. Consider the case of two BVs containing two sets of triangles both in a long thin rectangular configuration. Now consider that the two sets share a vertex such that they are connected forming a right angle. Grouping these two sets of triangles would not be optimal, as a tight BV containing both sets of triangles is not likely to be found. Additionally, such a BV is likely to accidentally enclose triangles that it is not supposed to, which also affects the performance of distance measurement queries negatively.

Considering for instance the grouping between the two sets of triangles highlighted in the above figure, such grouping is sub-optimal in the presence of sets of grouped triangles that share longer edges with either of the two sets of highlighted triangles. A possibly better grouping for the red set of triangles would be a set with a similar shape aligned along its side, such that their combined shape is rectangular, and such that the length is not much larger than the width. It is therefore important for the bottom-up construction mechanism to group sets of triangles in a way as to avoid sub-optimal sets such as the one shown above.

In order to tackle the bottom-up construction problem, a new class of objects was introduced, “Potential BVs”, each of which contains a BV object in addition to information needed during the bottom-up construction of the BVH, such as connectivity information of a Potential BV to other Potential BVs. As this information is no longer needed once the BVH is constructed, Potential BVs are objects that are only used during the construction of the BVH, while the BV objects they contain are saved in the final BVH.

The initial approach to the bottom-up construction consisted of first constructing the leaf BVs, then constructing Potential BVs which represent all possible groupings of the leaf BVs and inserting them in a minimum heap. The criteria for the comparison of the Potential BVs will be detailed further at a later stage. Iteratively, the algorithm would extract the minimum Potential BV object from the heap. If its child BVs are still not marked as grouped, the Potential BV is marked as a valid grouping, its child BVs are marked as grouped, therefore invalidating any other Potential BV using these BVs. Based on the connectivity information of the current Potential BV, new Potential BVs are constructed grouping it with each of its neighboring Potential BVs and inserted into the heap. This process is repeated until the heap is empty. The list of valid Potential BVs is then used to generate the BVH according to the PQP model.

Some of the heuristics used to compare Potential BVs during their insertion into the minimum heap are listed below. It should be noted that this approach did not produce good groupings despite several attempts at tuning, combining, and weighing the heuristics listed below:

- Smaller number of triangles enclosed by the BV
- Smaller number of neighboring BVs
- Smaller surface area of the BV
- BV whose rectangle’s aspect ratio is closest to a square

Since this approach failed to produce valid groupings, it was necessary to explore other approaches. One of the reasons this approach failed was its reliance on information extracted from the BV that can be built for the grouping, instead of information from the triangles within the grouping. After developing a graphical interface to represent the object models, it became clear that the algorithm in charge of building the BVs was not performing optimally, and that the BVs were not as tightly fit as they were thought to be, which rendered any decision based on information extracted from the BVs possibly sub-optimal and misguided.

The following figure shows the helmet model used in the simulation, where the white box represents the outer limits of the root RSS, and the blue and yellow boxes represent the outer limits of its children BVs. Note that the RSSs in this case are shown as OBBs for ease of graphical display. The actual limits of the RSSs lie within the rounded corners of these boxes.

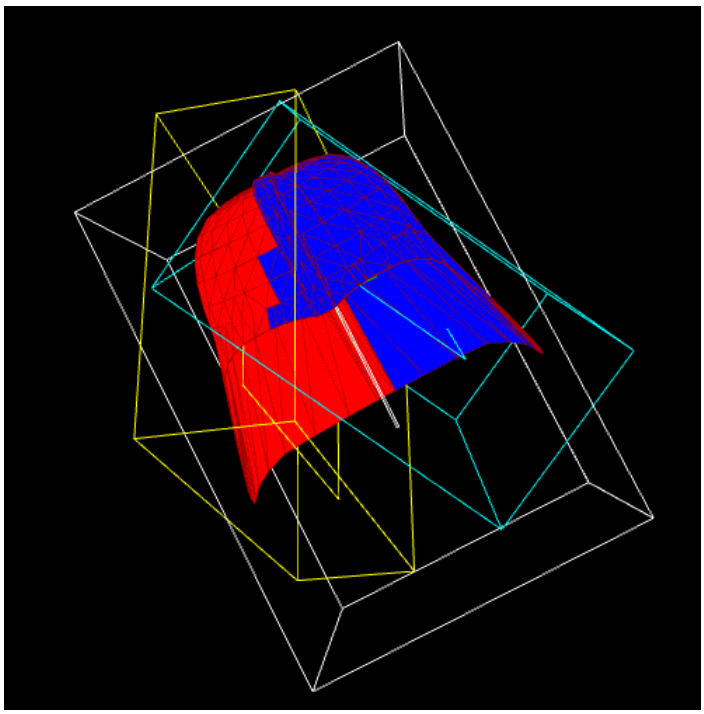


Figure 8: An example showing the tight fitness of the BVs as they are constructed around the helmet model

What the figure above clearly shows is that the BVs are not always tightly fit around the triangles they enclose. While the children of the root seem to be a good fit around their triangles (in red and blue respectively), the root node appears to be quite larger than it should be. This observation led to the conclusion that a method that is not based on information extracted from the BVs should be developed.

Based on the graph matching method described by Preis [Pre99], a new approach which, unlike the one described previously, explicitly builds the hierarchy one level at a time is developed and tested.

First, Potential BVs are constructed for all triangles in the object model. Based on this set of Potential BVs, the second level of the tree (from the bottom) is constructed by grouping the available Potential BVs two at a time.

The grouping of the Potential BVs starts with a given Potential BV B_1, and looks for the best fit neighboring Potential BV for it to be grouped with (the definition of "best fit" used in this implementation will be covered at a later stage). Let B_2 be the best fit neighbor of B_1. The algorithm then looks at the neighbors of B_2 for a neighbor that is a better fit grouping to B_2 than B_1 is. If such a neighbor B_3 is found, the algorithm goes through the neighbors of B_3, and so on, until no better fit is found. Since the exploration of the neighbors is done recursively, assuming B_2 and B_3 are matched, the algorithm then returns to B_1 to look for its best fit neighbor, then back to B_0. If all of B_0's neighbors have already been grouped, B_0 is marked as an isolated node.

When the grouping of all Potential BVs of a given level is done, and if the number of isolated nodes is higher than 1, an extra step is performed in order to find proper groupings that would leave out at most one isolated node at each level: Starting from an isolated node, a breadth-fist search is performed, traversing only the newly created groupings. The search begins with the neighboring Potential BVs of an isolated node, then moves on to Potential BVs neighboring the Potential BVs grouped to the set currently under exploration (not direct neighbors of the set of Potential BVs under exploration). The search ends when another isolated node is found, at which point the shorted path connecting the two isolated nodes is traced, also through grouped Potential BVs, and the groupings are swapped such that both isolated nodes become grouped. This search is repeated until at most one isolated node is left.

Once the final pairings of Potential BVs are found, the new Potential BVs are built according to the found pairings, then a new grouping process starts using the newly constructed Potential BVs. It should also be noted that isolated nodes of a given level are favored for matching at the next level, such that no node is left isolated on more than one level. The entire process ends when a root node has been created. The set of Potential BVs is then converted to a BVH using only BVs according to the PQP model.

Finally, the performance of the above described method depends on the measure of fitness comparison between pairings of Potential BVs. For that purpose, several measures were tested, most of which were similar in nature to the ones used in the previous method.

However a new measure proved to be successful at generating well-formed trees. In the evaluation of such measures, methods that generate square-like shaped pairings are preferred to others that generate odd-shaped pairings.

The measure used in this method, and that proved to generate good BV pairings consisted of measuring the total edge distance connecting the two sets of triangles. A pairing whose total edge connection is large is likely to be square-like in shape. This measure also allows for the

correction of oddly-shaped triangle sets, which are likely to be paired with corresponding sets producing a square-like shaped BV.