A Hybrid Bounding Volume Algorithm to Detect Collisions between Deformable Objects

F. A. Madera
Universidad Autónoma de Yucatán
Facultad de Matemáticas
Periférico Norte 13615, Mérida, Yucatán, México
mramirez@uady.mx

A. M. Day, S. D. Laycock
University of East Anglia
School of Computing Sciences
Norwich NR4 7TJ, UK
Andy.Day@uea.ac.uk, S.Laycock@uea.ac.uk

Abstract

An algorithm to detect collisions between both rigid and deformable objects is presented. The approach exploits benefits of a Bounding Volume Hierarchy (BVH) and a Feature-based method. The BVH decomposes the three dimensional polygonal objects into a hierarchy of spheres. The lowest level of the hierarchy is formed utilising spheres which bound 1-rings surrounding each vertex of the original mesh. Spatial coherence is exploited during construction to ensure that adjacent 1-rings are joined first. This promotes tighter bounding volumes as the objects deform. Experiments were carried out to analyse the performance of the method when varying the BVH construction to consider octrees and binary trees. To illustrate the enhancement the approach provides it has been compared against standard Sphere and Axis-aligned Bounding Volume Hierarchies.

1. Introduction

The widespread use of animation has resulted in a strong demand for accurate and believable collision detection. The contact points between two objects can usually be approximated at a small number of locations on a subset of the object’s primitives. When testing a virtual environment comprised of many objects, an efficient algorithm should first cull away objects which are deemed too distant to collide and then cull away distant primitives between close objects.

The use of the Bounding Volume Hierarchy in collision detection has been investigated for many years. Some work has explored the hierarchy construction phase to obtain a better approximation of the object’s shape [5, 1] whilst others focused on the hierarchy traversal phase to speed up the process [6, 10]. This latter phase contains two stages when considering deformable objects, the collision detection and the updating process. The updating part is required to ensure that the often pre-computed bounding volumes remain close approximations to the underlying geometry as the object deforms. To improve the collision detection tighter BVs can be used. However, this often causes more computation in the updating process. In contrast, to improve the updating process simpler BVs can be utilised which do not bound the underlying structures so tightly and therefore lead to potentially more levels in the BVH being tested for collision.

In this paper, we have focused on the efficiency of the updating process starting with spheres as the BVs. The approach proposed exhibits an efficient technique for updating the bounding volumes as deformation occurs, and it is derived from the algorithm presented in [9]. The original work suffered from insufficiently tight bounding volumes as the object deformed. This was particularly due to the way in which the BVH was constructed from the vertices of the mesh without taking proper regard for the topology of the object. To alleviate the shortcomings of the previous approach a new approach to the algorithm is taken.

In particular the contributions are as following.

• The object is divided into small clusters called 1-rings which are a set of triangles joined with a common vertex. These 1-rings form the lowest level of object decomposition used by the BVH and for this reason reduce the computations required in the hierarchy traversal process. The BV of a 1-ring is a sphere with a fixed centre indicated by the vertex joining the triangles. Utilising a vertex of the mesh negates the need to re-compute the centre in the updating process.

• We investigated the performance qualities of octrees and binary trees, experimenting with different objects.

• A comparison of our algorithm against the standard sphere (SBV) and Axis aligned bounding box (AABB) methods is presented.

The remainder of the paper is organised as follows: in Section 2 the proposed algorithm is described, experiments
are shown in Section 3, conclusions and plans for future work are given in Section 4.

2 The Distance Volume Hierarchy Algorithm

The DVH (Distance Volume Hierarchy) algorithm detects collisions between deformable objects by computing the squared Euclidean distance between the clusters defined in the objects and is derived from the DMV (Distance Middle Vertex) algorithm presented in [9]. The original work suffered from insufficiently tight bounding volumes as the object deformed. The reasons for this were largely due to the fact that the bounding volume hierarchy was constructed using an octree which worked with all the vertices of the original mesh without regard for how they were connected. This ensured the algorithm could work more easily with disjoint point sets as opposed to triangulated meshes but often as objects deformed large bounding volumes would result from the updating stage. A diagram of the algorithm is depicted in Figure 1.

![Diagram of the DVH Algorithm](image)

**Figure 1.** The DVH Algorithm to detect collisions is formed by two main stages, the Hierarchy Construction in the pre-computation stage and the Hierarchy Traversal during runtime.

During the precomputation phase the hierarchy is constructed and at run-time the hierarchy is traversed. To deform the objects we used morphing by point specification [2]. In this way, we know the vertices $v$ and their associated displacement vectors $\vec{v}'$. Once the objects’ meshes have been deformed, the original bounding volumes constructed must be recalculated. The aim of this stage is to keep the spheres as tight as possible as deformation occurs. This depends on the bounding method used, Guibas et al. [4] presented two approaches namely the wrapped approach, which encloses the primitives of the object, and the layered approach, which encloses the child spheres. Though the wrapped hierarchy is more difficult to compute than the layered hierarchy, it is tighter fitting, and it can be maintained more easily when used for deformable models. For these reasons the wrapped approach is employed in this work.

2.1 Hierarchy Construction

The phase of hierarchy construction is computed as a pre-processing step, where the top-down method is easily implemented. First of all the 1-rings are created in accordance with the algorithm listed in Figure 2, subsequently these 1-rings are grouped into clusters to form the upper levels of the hierarchy. A cluster is illustrated in Figure 3 and contains a group of 1-rings. The algorithm bounds each cluster of geometry with a sphere, which will be utilised for the intersection test and will need to be updated at runtime as deformation occurs.

```plaintext
1-ring-Construction()
0. Queue = random triangle, T ∈ Object’s mesh
1. FOR a triangle T in Queue
2. FOR a vertex v in T
3. IF v is not marked
4. -mark v as the common vertex of a 1-ring
5. -Create a 1-ring
6. -push in Queue the triangles of the 1-ring
```

**Figure 2.** The 1-ring construction algorithm. The Queue is a data structure to store the triangles.

Once the set of 1-rings have been determined, the hierarchy can be built using either an octree or a binary tree. This involves partitioning the object into small pieces of geometry that contain a number of 1-rings. A hierarchy is illustrated in Figure 4.

The hierarchy can be constructed using either an octree or a binary tree working with the centre vertices of the 1-rings to form clusters. These trees are created using the spatial subdivision method that begins with an axis-aligned bounding box of the object and divides it into child nodes. The algorithm chooses the centre of the node to be the closest vertex to the average of the vertices contained in the node. The nodes are recursively subdivided until one 1-ring is included.
Figure 3. A cluster of three 1-rings is bounded with two circles. The blue circle takes a vertex of the mesh close to the real centre and its diameter is illustrated with the blue line. The red circle is the minimum circle that bounds the points.

Figure 4. A tree with its nodes labelled. 1-rings are contained in clusters. Clusters can be formed by 1-rings or by other clusters.

The next stage is to create the BVs. Consider the cluster \( F \) formed with \( m \) 1-rings, \( F = \{ R_1, R_2, ..., R_m \} \). In the wrapped approach, a parent node encloses its child 1-rings. The sphere of the cluster \( F \) is represented with centre \( C = \frac{1}{m} \sum_{j=1}^{m} v_j \), and radius \( r = \max \| C - v_j \|_2 \), where \( j \) indexes each of the vertices of the 1-rings in \( F \) to ensure the most extremal vertex is used to calculate the radius. The bounding volumes of the nodes are formed from the vertices of their corresponding child nodes (1-rings), where most vertices to be computed occur in higher levels.

2.2 Hierarchy Traversal

At runtime, the objects are taken in pairs to compare their hierarchies using a breadth-first search traversal. Initially, the spheres held in the first level of the hierarchies are used to determine which bounding volumes to test next. According to [3, 8], the total cost of intersecting two hierarchies, \( TC \), is given as:

\[
TC = N_V C_V + N_P C_P + N_U C_U, \tag{1}
\]

where \( N_V \) is the number of BV pairs tested for overlap, \( C_V \) is the cost of testing a pair of BVs for overlap, \( N_P \) is the number of primitive pairs tested, \( C_P \) is the cost of testing a primitive pair, \( N_U \) is the number of nodes that need to be updated, and \( C_U \) is the cost of updating such a node.

The hierarchy traversal commences from the root of the two objects’ hierarchies and tests the separated BVs for overlap. The BVs tested are traversed in a top-down manner, until the leaves are reached. In non-balanced trees, the comparison between a leaf BV and an inner BV is a frequent occurrence. In this case, the inner BV’s children are compared with the leaf BV, until both leaves have been reached. This will lead to the desired result of a comparison between the leaf BVs of the two hierarchies. A 1-ring is formed from several polygons, enabling a more detailed test between the individual polygons to be performed as required.

The sphere is one of the cheapest bounding volumes when considering the computation time required to determine intersection. However, the coarser the spheres the higher the number of overlaps presented, that means that tighter bounding volumes should be sought in order to reduce the number of overlaps. As the objects deform, the bounding volumes will eventually no longer fit the underlying geometry correctly. Nodes are updated by recomputing their radii. Unlike other approaches such as [6, 10, 7], the centres of the spheres do not need to be recomputed because they have been defined in the preprocessing stage as vertices of the original mesh.

Figure 5. Experiment 1. Forty bunny objects containing 16K triangles each. Objects are moving and deforming during 200 frames of a simulation.
3. Experiments

Three experiments were set up to determine the computation times required for the collision detection algorithm. Objects are deformed using morphing by point specification [2], but the approach assumes no knowledge of the deformation method.

The graphs presented in Figures 8, 9 and 10 depict the total time of the collision detection algorithm: the updating process and the intersection test. Each experiment was run utilising three types of bounding volumes: DVH, SBV, and AABB. Table 1 lists the number of triangles and the number of 1-rings in each object, whilst Table 2 describes the number of levels and clusters in the trees for each object.

In Figures 8, 9 and 10, the computation times as defined by Equation (1) are illustrated based on a simulation of 200 steps.

<table>
<thead>
<tr>
<th>Object</th>
<th>No. of triangles</th>
<th>No. of 1-rings</th>
</tr>
</thead>
<tbody>
<tr>
<td>bunny</td>
<td>16K</td>
<td>3,363</td>
</tr>
<tr>
<td>dinosaur</td>
<td>84K</td>
<td>17,905</td>
</tr>
<tr>
<td>rabbit</td>
<td>100K</td>
<td>21,199</td>
</tr>
<tr>
<td>armadillo</td>
<td>350K</td>
<td>72,140</td>
</tr>
</tbody>
</table>

Table 1. The number of triangles and 1-rings in the objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>levels binary</th>
<th>clusters binary</th>
<th>levels octree</th>
<th>clusters octree</th>
</tr>
</thead>
<tbody>
<tr>
<td>bunny</td>
<td>14</td>
<td>1,421</td>
<td>6</td>
<td>1,097</td>
</tr>
<tr>
<td>dinosaur</td>
<td>20</td>
<td>7,847</td>
<td>8</td>
<td>7,188</td>
</tr>
<tr>
<td>rabbit</td>
<td>18</td>
<td>9,283</td>
<td>7</td>
<td>8,121</td>
</tr>
<tr>
<td>armadillo</td>
<td>23</td>
<td>33,086</td>
<td>9</td>
<td>31,184</td>
</tr>
</tbody>
</table>

Table 2. The number of clusters and levels in the object’s hierarchies.

The first scene tested the algorithm using 40 objects of the same instance, a bunny of 16K polygons, as Figure 5 shows. The objects are undergoing rigid body transformations in addition to deformations. When a pair of objects collide, the translation and rotations are modified such that the objects separate. The deformation is global, affecting the whole hierarchy in such a way that most of the bounding volumes need to be updated. The total time of the octree is less than the total time of the binary tree for the AABB and the SBV. This difference is because the updating process for the binary tree is more expensive. The total time of the DVH remains equal. The intersection test requires more time for the DVH algorithm when objects are close. Since the AABB is a BV bordered by points independent of the mesh, the BV should follow the object by using the transformation matrix to hold the BV in correct alignment. Due to the fact that the deformations are global, this causes the updating process to recompute the majority of the nodes in the hierarchy. Note that the AABB is the most expensive method when the binary tree construction is applied, as shown in Figure 8. However, when using the octree, the AABB takes less time when objects are closer, since more collisions are obtained.

The second experiment, illustrated in Figure 6, involved testing three objects, two dinosaurs comprised of 84K triangles each, and a rabbit model containing 100K triangles. The simulation was set up to deform the objects obtaining collisions between the three pairs. The larger deformations occur in the upper parts of the objects. The intersection and updating times were recorded for every frame of a 200 frame simulation, and the results are shown in Figure 9. The algorithm requires similar computation time for both trees, the DVH is approximately 15 ms faster than the SBV, and the SBV is approximately 18 ms faster than the AABB per frame.

The third experiment, illustrated in Figure 7, utilised a scene containing two objects of 350K triangles each. The deformation method is applied over each object and the collisions occur in the hands of both armadillos. The total time to detect collisions is shown in Figure 10. The binary tree takes more time than the octree, with the DVH being the fastest, followed by the SBV and finally the AABB. Comparing the binary tree against the octree, the AABB took 100 ms more, the SBV took 40 ms more, and the DVH keeps consistent. With regard to the intersection time, the SBV was the most costly, and the AABB was the least costly. Once more the notable difference lies in the updating
process where the DVH remains the fastest.

Figure 7. Experiment 3. Two objects of $350K$ triangles are deforming and colliding in the scene.

4. Conclusions and Future Work

A significant modification to the DMV algorithm is presented resulting in the DVH algorithm to efficiently compute collisions between deformable objects. The design and implementation issues were discussed as well as some experiments to validate its performance. This algorithm is suitable for the detection of collisions between deformable models due to the fact that the BVs are represented by some vertices of the mesh.

We have carried out three experiments regarding different types of objects, motions and deformations in order to generalise the results of the algorithm’s performance. The deformations applied are large such that the algorithm fully updates the whole hierarchy. This is in contrast to local deformations where only a small part of the hierarchy is affected and consequently only a small area must be recalculated. We considered two types of trees, but other trees or hybrid trees can be used. The number of primitives and cost of determining primitive intersections are replaced by the overlapping 1-ring tests utilised in our algorithm. According to our experiments, the octree construction method resulted in the best run-time efficiency, whilst the accuracy remained consistent. In general the DVH algorithm performs very well in comparison to the other approaches.

The use of a hybrid technique has been convenient in the design of the algorithm. This overcomes the problem of having to spend a lot of time updating the BVs as objects deform, and to reduce the number of primitives to be dealt with. In the case of the feature-based method, the algorithm helps to use non-convex objects in constant motion and deformation. Consequently, by taking the vertices of the mesh as the centres of the spheres, the updating process can be simplified and the total cost of intersecting two hierarchies, $T_{c}$, can be reduced, leading to an algorithm that is well suited to detect collisions between deformable objects.

As the collision detection is a geometric problem, other animation strategies can be used to deform the objects since the input parameters are specified by the locations of the object’s features that are constantly changing. The algorithm is flexible, it allows the use of another kind of tree, a clustering method to define clusters, and any other deformation strategy. In applications where the accuracy is not relevant, we can stop the process of collision before reaching the leaves.

References


Figure 8. Scene1. Total time of the collision detection algorithm for the three methods AABB, DVH, and SBV. This time includes the intersection test for all the bounding volumes of the hierarchy and the updating process. The octree is depicted in the left side and the binary tree in the right side.

Figure 9. Scene2. Total time of the collision detection algorithm for the three methods AABB, DVH, and SBV. This time includes the intersection test for all the bounding volumes of the hierarchy and the updating process. The octree is depicted in the left side and the binary tree in the right side.

Figure 10. Scene3. Total time of the collision detection algorithm for the three methods AABB, DVH, and SBV. This time includes the intersection test for all the bounding volumes of the hierarchy and the updating process. The octree is depicted in the left side and the binary tree in the right side.