Sphere-Tree Construction using Dynamic Medial Axis Approximation

Gareth Bradshaw and Carol O'Sullivan
Image Synthesis Group, Dept. of Computer Science,
Trinity College, Dublin, IRELAND
{Gareth.Bradshaw, Carol.OSullivan}@cs.tcd.ie

Abstract

Collision handling is very computationally expensive, especially in large scale interactive animations. Hierarchical object representations play an important role in performing efficient collision handling. Many different geometric primitives have been used to construct these representations, which allow areas of interaction to be localised quickly. For time-critical algorithms, such as interruptible collision detection, there are distinct advantages to using hierarchies of spheres, known as sphere-trees. This paper presents a novel algorithm for the construction of sphere-trees. The algorithm presented approximates objects, both convex and non-convex, with a higher degree of fit than existing algorithms. In the lower levels of the representations, there is almost an order of magnitude decrease in the number of spheres required to represent the objects to a given accuracy.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Object hierarchies; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation G.1.2 [Mathematics of Computing]: Numerical Analysis—Approximation

Keywords: sphere-tree construction, object approximation, medial axis, level-of-detail collision detection

1 Introduction

Collision handling is a major bottleneck in any interactive simulation. Ensuring objects interact in the correct manner is very computationally intensive and much research has addressed the issues involved with trying to reduce the computational requirements. Researchers often utilise hybrid collision detection algorithms to tackle the problem in various phases. The initial phase of such an algorithm, known as the *broad* phase, aims to efficiently cull out pairs of objects which cannot possibly be interacting. A number of different techniques have been used to achieve this coarse grain detection. These include Sweep & Prune [Cohen et al. 1995; Ponamgi et al. 1997], global bounding volume tables [Palmer and Grimsdale 1995] and overlap tables [Wilson et al. 1998].

Having determined which objects are potentially interacting, the hybrid algorithm uses a finer grained algorithm to narrow in on the regions of the objects which are in contact. This *narrow phase* processing typically traverses hierarchical representations of the objects to hone in on the regions of interest. This reduces the amount of work that is required to perform the "exact" collision detection using such algorithms as Lin-Canny [Lin and Canny 1991; Lin 1993], V-Clip [Mirtich 1998] or Enhanced GJK [Gilbert et al. 1988; Rabbitz 1994; Cameron 1997].

Many different geometric primitives have been used for constructing the "Bounding Volume Hierarchies" (*BVH*) used to perform narrow phase processing. These include: Spheres [Quinlan 1994; Palmer and Grimsdale 1995; Hubbard 1995a; Hubbard 1995b; Hubbard 1995c; Hubbard 1996; O'Sullivan and Dingliana 1999], Axis Aligned Bounding Boxes (*AABB*) [van Den Bergen 1997], Oriented Bounding Boxes (*OBB*) [Gottschalk et al. 1996; Krishnan et al. 1998a], Discrete Oriented Polytopes (*k-DOP*) [Klosowski et al. 1998], Quantised Orientation Slabs with Primary Orientations (*QuOSPO*) [He 1999], Spherical Shells [Krishnan et al. 1998b] and Sphere Swept Volumes (*SSV*) [Larsen et al. 1999].

There is often a trade-off between the complexity of the bounding volume primitive and the tightness of fit that can be achieved. Simpler primitives, such as spheres and AABBs, are quite inexpensive to test for intersections. However, as they provide relatively poor approximations, large numbers are often required to approximate the objects effectively. More complex bounding volume primitives require more expensive intersection tests but, as they often provide tighter approximations, fewer primitives (and hence intersection tests) are required. The following equation has been used in [Gottschalk et al. 1996], [van Den Bergen 1997] and [Klosowski et al. 1998] to evaluate various types of bounding volume hierarchies :

$$T = N_u \times C_u + N_v \times C_v, \quad (1)$$

where :

T	is the total cost of collision detection between two objects,
N_u	is the number of primitives updated during the traversal,
C_u	is the cost of updating a primitive's position/orientation,
N_v	is the number of overlap tests performed,
C_v	is the cost of an overlap test between a pair of primitives.

Although spheres often do not provide particularly tight bounding volumes, there are a number of distinct advantages to their use. They are particularly advantageous when using the interruptible collision detection algorithm. This algorithm, introduced by Hubbard, uses a time-critical traversal of the sphere-trees which is terminated when the allotted time-slice has expired so as to maintain a consistent frame-rate [Hubbard 1995a; Hubbard 1995b; Hubbard 1995c; Hubbard 1996]. As the collisions may never be resolved down to the surface of the objects, the spheres can be used

to approximate the collisions and formulate the response, as done in [O'Sullivan and Dingliana 1999; Dingliana and O'Sullivan 2000]. Attractive properties include :

- *Rotational Invariance* : A sphere is invariant to the rotation of the objects and therefore its update cost is very small. No matter what type of motion the bodies are going through the spheres can be updated by simply translating their centers.
- *Efficient* : The cost of performing an overlap test between two spheres is extremely low, requiring only a few floating point multiplications and additions.
- *Suitability for Response* : While providing graceful degradation of the object approximation, each level of spheres in the hierarchy provides an approximation of the contact information necessary for collision response.

While many techniques have been used for the construction of sphere-trees, using the object's medial axis ("skeleton") has been shown to produce tight fitting sphere-trees. This paper presents a novel sphere-tree construction algorithm which extends the medial axis based algorithm to allow the medial axis to be constructed as required (full details can be found in [Bradshaw 2001]). Thus the medial axis always provides enough information for the construction of the approximation. The rest of this paper is organised as follows : Section 2 discusses the requirements of a good bounding volume hierarchy and gives a brief overview of some of the existing algorithms for the construction of sphere-trees. Section 3 overviews the process of sphere-tree construction within our frame-work and discusses how object sub-division is managed. Section 4 details the use of Voronoi diagrams for the approximation of the medial axis. Section 5 presents the adaptive medial axis approximation algorithm which extends the existing algorithm to allow the approximation to be updated as required. Section 6 presents a number of new algorithms for the generation of sphere sets from the medial axis approximation. Section 7 evaluates the new algorithms for the approximation of objects, the construction of sphere-trees and their use within interactive simulations. Finally, Section 8 presents conclusions and mentions some possible routes for future work.

2 Existing Algorithms

A number of algorithms have been used for the construction of sphere-trees. Any algorithm which constructs bounding volume hierarchies for collision detection must meet four basic requirements [Hubbard 1995b] :

- the hierarchy conservatively approximates the volume of the object, each level representing a tighter fit than its parent;
- for any node in the hierarchy, its children should cover the parts of the object covered by the parent node;
- the hierarchy should be created in a predictable automatic manner, not requiring user interaction;
- the bounding volumes within the hierarchy should fit the original model as tightly as possible, representing it to a high degree of accuracy.

For interactive simulations, the emphasis is on achieving high and consistent frame-rates. Therefore a major concern is how well the hierarchy facilitates this goal. As the collision detection algorithm may not fully resolve the collisions, approximate collision information must be available from each level of the BVH.

The simplest algorithm for construction of sphere-trees uses the octree data-structure. The sphere-tree is constructed by using recursive sub-division of the object's bounding cube. The regions which cover part of the object are then further sub-divided, continuing down to the required depth. The sphere-tree is then constructed by placing a sphere around each of the nodes of the octree. This method has been adopted in [Palmer and Grimsdale 1995; Hubbard 1995a; Hubbard 1995b; Hubbard 1996; O'Sullivan and Dingliana 1999]. The simplicity of the algorithm allows it to be quickly and easily implemented and for the sphere-trees to be updated when objects deform. However, the sphere-trees produced often fit the object quite poorly.

Quinlan also uses sphere-trees for collision detection. The sphere-trees are constructed by first covering the surface with a set of uniformly sized spheres which represent the leaf nodes of the hierarchy. This set of spheres is divided into two (roughly equal) sub-sets. Trees are constructed for each of the two sets and these trees are used as children of the root [Quinlan 1994]. Q-Splat uses a similar strategy for constructing sphere-trees for visibility culling and level-of-detail rendering [Rusinkiewicz and Levoy 2000].

Along with the octree method, Hubbard also explored two other methods for the construction of sphere-trees. He initially used *simulated annealing* for sphere-tree construction. Later he used the object's medial axis to guide where to put spheres. This method provides tight fitting sets of spheres from which the sphere-tree is constructed.

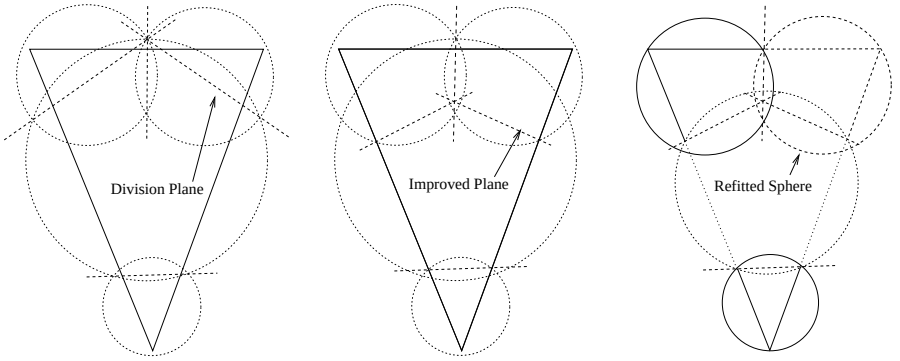
3 High-Level Construction Algorithm

This section gives an overview of the new sphere-tree construction algorithm. The algorithm consists of a number of layers. The top layer decomposes the construction into a number of sub-problems. It controls how the object is sub-divided into regions, each of which needs to be approximated with a set of spheres. These spheres will become the children of the sphere which was used to define the approximated region.

The root node of the sphere-tree is the smallest sphere that will enclose the object, which can be found with a minimum enclosing ball algorithm [Weltz 1991; White (www)]. The first level of spheres, the children of the root sphere, are constructed by calling a sphere generation algorithm which generates the required number of spheres to cover the object. These spheres are then used to segment the object into a number of regions. Each region defines the areas of the object that must be covered by a set of children spheres. These spheres form the next level of the hierarchy - as children of the sphere that defined the region.

3.1 Object Segmentation

For each node in the sphere-tree, sphere generation algorithms are used to construct a set of children spheres. In order to ensure that the entire object is approximated, the set of children spheres must cover the region covered by their parent. Thus, to produce the children spheres, we need to be able to determine the regions of the object covered by each sphere in an arbitrary set of spheres. The simplest method to achieve this would be to simply use any part of the object (or its surface) which is contained within the parent sphere. However, spheres generated from the medial axis tend to contain large areas of overlap inside the object. This is particularly true when trying to achieve tight fitting sets of spheres. Thus there will be large areas of the object which are shared between sets of children spheres. Rather than representing the common regions in multiple sub-trees, they are divided between the sub-trees so that each region will only need to be covered by one of the sets of children.



(a) Initial Divisions (b) Improved Divisions (c) Resulting Regions

Figure 1: Dividing the object into distinct regions using dividing planes.

Rather than working directly with the underlying model, the surface is represented by an arbitrarily large set of sample points. Thus to segment the object into regions we simply choose the sub-set of points that represent the surface within that area. For points covered by many spheres it is necessary to assign the point to one of the sets of children. This is achieved by dividing the overlapping region between the spheres. For each pair of overlapping spheres a dividing plane is constructed. The points contained within the intersection of the two spheres are divided based on which side of the plane they lie on.

Figure 1 shows how a triangular object is divided into four regions. The dividing planes, shown in Figure 1(a), are constructed so that the plane passes through the points of intersection of the spheres (circles in 2D). In this example the region associated with the large central sphere is much bigger than the regions associated with the other spheres. Thus, the children of this sphere will potentially have a looser fit than the other child sets. It is very desirable that the regions divide the object as evenly as possible without affecting the fit of the current set of spheres. Thus, the dividing plane between a pair of spheres is moved to try to get each to cover a region of equal size, Figure 1(b). Once we have established the regions to be covered by each of the spheres, new spheres can be created. These new spheres are now only required to cover the parts of the surface which are not covered by any other spheres, allowing a tighter sphere to be fitted, Figure 1(c).

4 Medial Axis Approximation

Constructing the medial axis for a polyhedral model is a complicated and computationally expensive task. For the purposes of constructing sphere-trees an approximate medial axis suffices. This section describes the algorithm for constructing a medial approximation using a Voronoi diagram.

A Voronoi diagram is constructed from a set of forming points. Each cell within the Voronoi diagram represents the region of space which is closer to its forming point than any of the other forming points. When the set of points is distributed over the surface of the object, a sub-set of the faces that lie between the cells forms an approximation of the medial axis [Hubbard 1995b]. We use Hubbard's algorithm, based on work described in [Bowyer 1981] and [Inagaki et al. 1992], for constructing the Voronoi diagram.

The algorithm for the construction of the Voronoi diagram is an iterative one. Each of the forming points, which are points distributed across the surface of the object, is added to the diagram in turn. As the points are added, new cells are constructed and the existing cells are updated so that each one still represents the region of space closest to its forming point. The algorithm is initialised with a Voronoi diagram containing 4 forming points which form a bounding tetrahedron for the object. The Voronoi vertices (which make up the cells) consist of one true vertex, constructed from the forming points, and 4 dummy vertices located at infinity.

When a new point is added to the Voronoi diagram, a new cell must be formed. This cell represents the region of space closer to the new forming point than to any of the existing forming points. To update the Voronoi diagram, the vertices which are closer to the new forming point need to be removed and new vertices created. Vertices which are approximately equidistant from both their forming points and the new point are considered *marginally deletable*. These are problematic as it is difficult to determine if they should be deleted or not. Hubbard finds the set of deletable vertices by considering the vertices in order of removability. The search starts with the most deletable vertex and maintains a priority queue of the possibly deletable vertices, which is initialised with the positively valued neighbours of the starting vertex. Each vertex in the queue is considered in order of removability. If the deletion of a vertex will invalidate the Voronoi diagram, i.e. it cannot be added to the deletable set, it is held back for later consideration. When a deletable vertex is found, its positively valued neighbours are added to the priority queue, and the vertices that were being held back are returned to the queue for further consideration.

Having found the set of deletable vertices, a new vertex is created for each pair of neighbouring vertices V_d and V_u, where V_d is a deletable vertex and V_u is an undeletable vertex. These vertices will share three forming points, which together with the new point will create the new vertex. The circumcenter of the tetrahedron formed by these four points gives the vertex's location. As with all Voronoi vertices, the new vertex V_n has four neighbours, V_u and three other new vertices. These other neighbours are the new vertices with which it shares 3 forming points.

The Voronoi vertices inside the object represent points on the approximate medial axis. As the medial axis is defined as the set of maximally sized spheres that fill the object [Blum and Nagel 1978], these locations are used to make a set of spheres which approximate the object. Using the distance from the vertex to its forming points as the sphere's radius results in it touching the surface in (at least) 3 places. The sphere placement algorithm does not need to give special consideration to degeneracies in the medial axis, such as zero length edges, as this just results in two spheres being located in the same position.

5 Adaptive Medial Axis Construction

The algorithm presented in the previous section approximates the medial axis of an object with a Voronoi diagram. There are, however, a number of difficulties associated with this process. When generating the Voronoi diagram, it is difficult to choose the set of surface samples correctly. Problems in choosing the set of surface points can result in the medial axis being pushed through the surface to create either a hole in the approximation or a bridge that joins two separate areas of the object.

Hubbard introduces the notion of "gap crossing" cells as a way of identifying and fixing these problems [Hubbard 1995b]. However, this scheme does not address the problems associated with constructing a sphere-tree from the medial axis approximation. It is impossible to determine *a priori* how to construct a medial axis, and hence the set of medial spheres, that will best suit the sphere-tree being constructed. We favour an adaptive medial axis construction algorithm, presented in this section. This algorithm handles both the gap crossing problems and allows the medial axis approximation to be updated so that there is always a sufficient number of spheres available to construct the sphere-tree and to fit the object to a high degree of accuracy.

The adaptive medial axis approximation algorithm is iterative in nature. It starts with an empty Voronoi diagram and iteratively improves the approximation by adding new sample points to the surface of the object. The first task of the algorithm is to ensure that the set of medial spheres, which is the set of spheres placed around the Voronoi vertices, covers the object's surface. The second phase of the adaptive algorithm is to adaptively improve the medial spheres.

5.1 Completing Coverage

In order to ensure that the set of medial spheres is complete, it must cover the entire surface. This is achieved by representing the object as a densely packed set of points distributed across its surface. The surface is considered to be covered when all these points are covered by spheres. This representation makes it simple to ensure that the model is completely covered. For those points which are not covered we need to create spheres which will cover them. As the set of spheres placed around a Voronoi cell's vertices will cover every part of the cell, the extra spheres can be chosen from those belonging to the cell that contains the uncovered point.

There are several criteria that could be used to choose the sphere to cover each point. The smallest sphere, or the sphere with the lowest error, would help to produce a tight fitting approximation, whereas choosing the largest sphere would also cover a number of other uncovered points. This would mean that fewer of the spheres in the approximation would be "coverage spheres". As the adaptive algorithm is able to replace poor fitting spheres, these spheres will subsequently be replaced if they affect the quality of the approximation.

5.2 Iterative Improvement

The second phase of the adaptive medial axis approximation algorithm is the improvement step. During each iteration of the algorithm, having marked the medial (and coverage) spheres, the sphere with the worst fit is replaced[1]. This is achieved by adding a new sample point to the Voronoi diagram. This point, q, is the point on the surface which is closest to the center of the sphere, c. As q will be closer to the vertex than its forming points, this results in the vertex being replaced with new ones, as illustrated in Figure 2.

[1] Each Voronoi vertex caches its sphere and the associated error to allow the worst fitting sphere to be chosen quickly.

6 Sphere Generation

The top level sphere-tree construction algorithm, presented in Section 3, expresses sphere-tree construction as a number of sub-problems, each requiring that a region of the object be approximated with a set of spheres. This section details the process of constructing sphere sets from the medial axis approximation.

The sphere generation algorithm maintains a Voronoi diagram which approximates the medial axis of the object. Prior to the construction of a set of spheres, the medial axis is updated so that it is of sufficient quality to allow a tight approximation. For this we require that the set of spheres covering the region to be approximated contains some multiple of the target number of spheres and that the worst sphere in the set has an error which is a fraction (typically $\frac{1}{2}$ to $\frac{1}{4}$) of the parent sphere's error.

Having updated the medial axis approximation, the set of spheres which approximates the required region is used to construct the spheres for the sphere-tree. This set of spheres is reduced so that it contains the required number, i.e. equal to the branching factor of the tree. We have explored a number of algorithms for performing this reduction. Our favoured approach is to use two different algorithms, detailed next, and choose the set of spheres which best fits the object.

6.1 Merge

The "merge" sphere reduction algorithm, loosely based on Hubbard's algorithm, reduces the sub-set of medial spheres by successively merging pairs of spheres. Each sphere is allowed to merge with a number of other spheres, its neighbours. Initially, each sphere is given a set of neighbours which correspond to the vertex's neighbours within the Voronoi diagram.

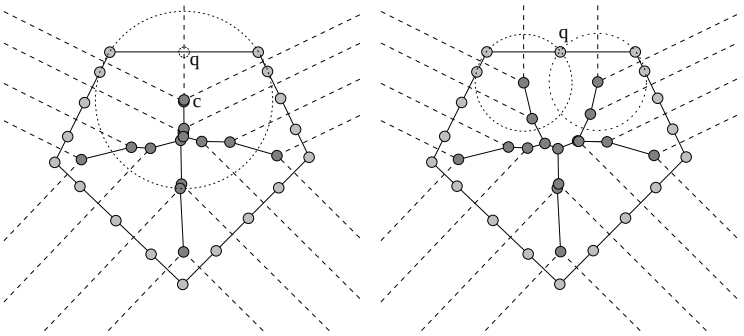
Each of the medial spheres covers a sub-set of the surface points. When a pair of spheres is merged, a new sphere is constructed which covers the union of the two sets of surface points. When constructing this sphere we wish to keep it close to the medial axis. Thus we use two approaches to constructing the bounding sphere. We first resize both the spheres to enclose both sets of points. Next, we create the minimum volume bounding sphere for the points, using White's algorithm [White (www)]. The final sphere is chosen to be the one with the lowest error.

Having combined a pair of spheres, the set of merges is updated. The neighbours of each of the merged spheres will become neighbours of each other, resulting in new potential merges. Any spheres that end up with no neighbours are given an artificial set of neighbours. This set of neighbours consists of any spheres which overlap the neighbourless sphere or all the remaining spheres if there are no overlaps. The reason for limiting the pairs of spheres that may be combined is to reduce the computational cost of the reduction process. When the number of spheres becomes sufficiently low, it becomes feasible to consider every pair for merging. This is typically done when we reach 2 or 3 times the target number of spheres.

Each iteration of the merge algorithm reduces the number of spheres by one, using a greedy algorithm. At each iteration, the merge which results in the lowest error is used. This does not consider the effects of the merge on future operations. We give special consideration to merges which actually reduce the error in the approximation. We refer to these as "beneficial merges". As an approximation is only as good as its worst error, we favour merges which improve the worst spheres in the approximation.

6.2 Expand

Although the merge algorithm is very general, it does not consider the global effect of merging two spheres and therefore can often produce sub-optimal sets of spheres. A much better strategy is



(a) Before (b) After

Figure 2: Addition of a new point to the Voronoi diagram to improve the approximation.

to choose a set of spheres that distribute the error evenly between them. By ensuring that all the spheres have the same error, the algorithm will have a better chance of achieving a tight fit. For a convex body, a sphere can be grown to hang over the surface by a given amount, e, using the following equation to calculate its radius (with q and c as defined in Section 5.2) :

$$r = e - \|q - c\| \tag{2}$$

Thus, the medial spheres are expanded to the desired stand-off distance, using Equation 2, and a sub-set of the spheres is selected. For non-convex bodies, the stand-off distance is an over-approximation of the error present in the sphere. A simple search algorithm, derived from binary search, is used to find the lowest value of e that results in a set that does not contain too many spheres.

The job of selecting the minimum number of expanded spheres that cover an object is a complicated one. Instead of looking for the globally optimum set we try to find a good minimal set of spheres, i.e. a set of spheres from which none of the spheres can be removed without exposing part of the surface. A greedy algorithm allows us to achieve this efficiently. In this algorithm the set of currently selected spheres is maintained and more spheres are selected until the region is completely covered. To decide which sphere to add, each sphere is ranked according to its potential to keep the set of spheres small. Two heuristics for ranking the spheres have been explored. The first is to simply rank the spheres by the area of previously uncovered surface contained within it. The second heuristic ranks the spheres by the number of other spheres which it makes redundant. The second heuristic tends to select spheres near to those previously selected and seems to work slightly better in practice.

7 Evaluation

This section compares sphere-trees generated using the new algorithm with existing medial axis based sphere-trees. Other techniques, including the octree method, have also been evaluated but the results are not included here as the medial axis method produced far tighter fitting hierarchies, as expected. A full discussion of these and other results can be found in [Bradshaw 2001].

The algorithms have been compared using a number of simple geometric shapes including a cube, an ellipsoid, a cylinder, a torus, a cone, an S-shaped object and a block with square cross sections. A number of commonly used complex models have also been used, including the Bunny[2], the Cow and the Dragon[3]. The Bunny and S-shape are shown in Figure 3.

There are a number of factors which must be considered when approximating an object with spheres. As stated in Section 2, the spheres should approximate the object's surface to a high degree of accuracy and should cover the entire object. The tightness of fit can be measured as the maximum distance from the surface of the spheres to the actual surface of the object. This represents an upper bound on the distance between two objects when they are falsely thought to be involved in a collision. The initial analysis is concerned with the geometric properties of sphere-trees and the later analysis considers the use of the resulting sphere-trees in an interruptible collision handling system.

7.1 Sphere-Tree Construction

Figure 4 compares the geometric fit achieved using the various sphere reduction algorithms. All tests were conducted with a tree branching factor of 8. All the algorithms used a set of $5000-10000$ surface points to check coverage. The original algorithm was used with a medial axis approximation containing *circa* 2500 spheres. For the adaptive algorithm, the medial axis initially contained 500 spheres and was dynamically refined so that each region had $\frac{1}{2}$ the error of the parent sphere and at least 100 spheres, i.e. the merge/expand algorithms started with 100 spheres from which the final 8 were to be produced.

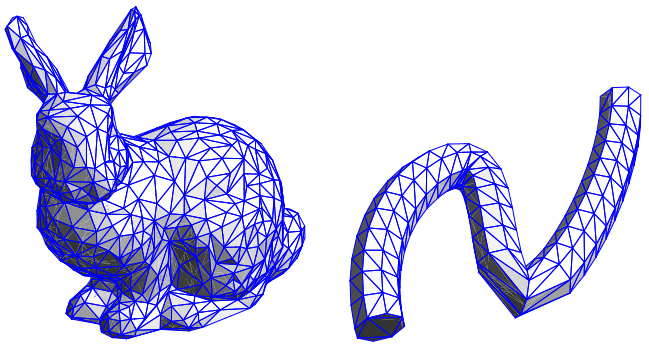
The new merge algorithm shows significant improvements over the original algorithm. For complex models, such as the Bunny, the 3^{rd} level spheres produced by the new merge algorithm exhibit about $\frac{1}{2}$ the error of those constructed with the original algorithm. The expand algorithm shows further improvements and using both sphere reduction algorithms in combination results in a further decrease in error. In fact, the worst case for the combined algorithm's level 2 spheres is roughly the same as that for the original algorithm's level 3 spheres (see Figure 4(a)). Thus the combined algorithm produces the same tightness of fit using around $\frac{1}{8}^{th}$ the number of spheres.

7.2 Simulation

In order to further evaluate the sphere-tree generation algorithm, the behaviour of the sphere-trees during simulation was tested. During

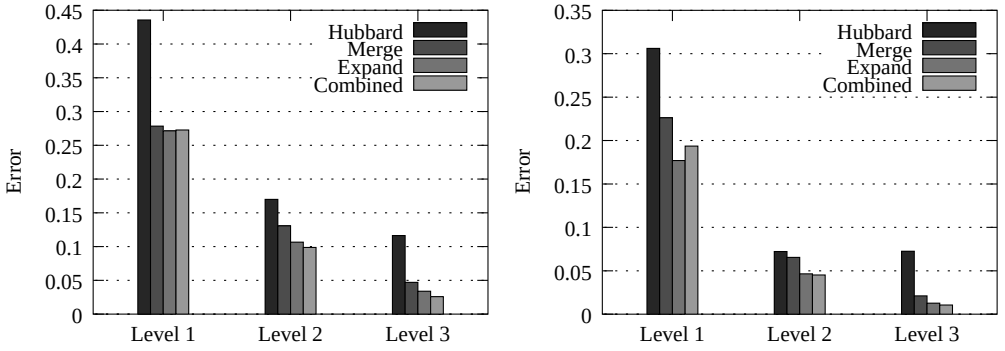
[2]Data from http://graphics.stanford.edu/data/3Dscanrep/

[3]Data from http://graphics.cs.uiuc.edu/∼garland/research/quadrics.html



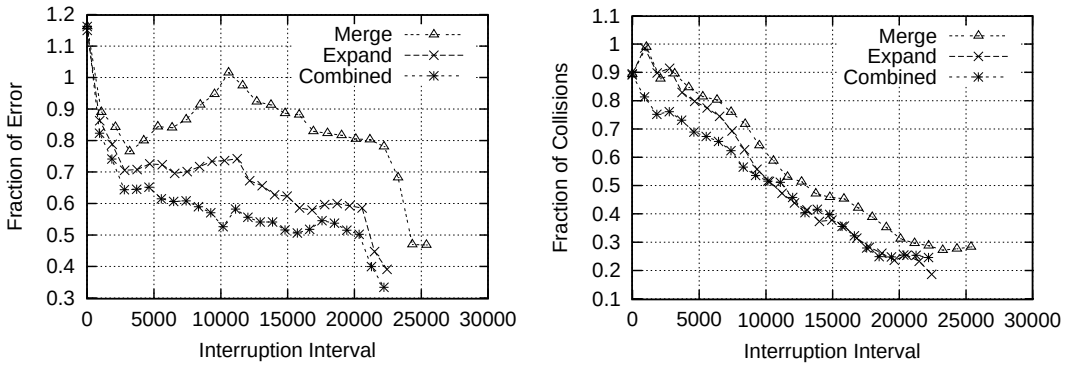
(a) Bunny (b) S-shape

Figure 3: Some of the models used for testing the algorithms.



(a) Bunny (b) S-Shape

Figure 4: Comparison of sphere-tree construction algorithms (worst error).



(a) Worst Error (b) Resulting Contacts

Figure 5: Comparison of sphere-trees at various interruption times for the Bunny (20 objects).

these simulations, objects were positioned (and oriented) randomly about a sphere and given a random velocity towards a central point. Each sphere-tree approximation of the object was evaluated during the simulations. At each time-step the colliding pairs were created by testing their bounding spheres. The sphere-trees for these pairs were then traversed as they would be in the interruptible collision detection algorithm. To avoid bias towards or against a particular sphere-tree, the motions of the objects and their response to collisions were computed using a commercial dynamics system which uses an "exact" collision detection algorithm and a complex friction dynamics model.

The sphere-tree traversals were interrupted at regular intervals and the approximate collisions evaluated. The error associated with each colliding pair was computed as the sum of the spheres' errors, i.e. the maximum distance between the sphere and the surface. This represents an upper-bound for the true separation between the objects. The new algorithms were compared to a reference sphere-tree, i.e. that constructed with Hubbard's original algorithm. For each interruption time, the average improvement was computed using all the frames of the animation (typically 1000+).

Figure 5 presents results for simulations containing 20 objects. The horizontal axis of each of the graphs shows the interruption interval. This is expressed in terms of the number of primitive operations performed, i.e. sphere updates and overlap tests. The amount of work done before interruption is computed using Equation 1. The values of C_u and C_v are 2 and 1 respectively. These values represent the relative number of floating point operations performed when updating a sphere's position (21 floating point operations) and testing two spheres for overlap (10 floating point operations). The first graph (a) in each set shows the fraction of error present in the approximations and the second (b) shows the relative number of colliding pairs produced by the different algorithms.

The new algorithms show a definite reduction in error. For complex models, such as the Bunny, the combined algorithm quickly falls to as little as 50% of the error resulting from existing algorithms (see Figure 5(a)). The amount of error continues to decrease to as low as 30%. In other simulations, using the simpler models, we have found this to be as low as 20%. The algorithms also show significant reductions in the numbers of pairs of colliding spheres, which result from the traversal. This provides a reduction in the amount of work which will need to be done by the later stages of the collision handling system, i.e. contact modelling and collision response. For the Bunny, the number of resulting collisions has decreased to as little as 20% (see Figure 5(b)).

8 Conclusions & Future Work

This paper has presented some novel work in the areas of medial axis approximation and sphere-tree construction. An adaptive medial axis approximation algorithm, which addresses many of the issues involved with using Voronoi diagrams for this purpose, was presented. The adaptive algorithm not only allows us to ensure that every part of the object is approximated but also allows the sphere-tree construction algorithm to refine the approximation as it moves down the sphere-tree. The approximate medial axis is then used to construct a set of spheres from which the sphere-tree is constructed.

A generic sphere-tree construction algorithm was presented which expresses the task as a number of sub-problems. Each of these sub-problems involves approximating a region of the object with a desired number of spheres. This allows many different algorithms, each of which generate the required number of spheres from the medial axis approximation, to be slotted into the top-level algorithm. The sphere reduction algorithm presented combines two of the algorithms we have explored. The merge algorithm is generic in nature and is suitable for most scenarios. The expand algorithm takes a novel approach to sphere reduction. This algorithm reduces the set of medial spheres by enlarging them and selecting a sub-set to cover the object. This produces spheres which distribute the error evenly between them and often results in tighter fitting sets of spheres.

The sphere-reduction algorithms were used for the construction of sphere-trees to approximate a number of different models. The new algorithms consistently provide improvement over the existing algorithms. In the lower levels of the sphere-trees, the new combined algorithm required almost an order of magnitude less spheres. The sphere-trees were also evaluated for use in interruptible simulations. The sphere-trees constructed with the new algorithms showed a large decrease in error and resulted in far fewer pairs of colliding spheres.

There are a number of interesting areas of research that can build upon this work. To date, we have concentrated on fitting tight hierarchies of spheres to rigid bodies. This can, of course, be used for articulated objects too. However, further work would be required to make the algorithms suitable for use with deformable objects. Also, many objects contain large flat areas which are not well approximated by spheres, e.g. buildings. For general purpose simulation it would be nice to be able to use other collision detection strategies for these areas while using sphere-trees where applicable. Sphere-trees have also recently been used for visibility culling and level-of-detail rendering [Rusinkiewicz and Levoy 2000; Rusinkiewicz and Levoy 2001]. It would be extremely interesting to combine this technique with level-of-detail collision detection so that the same sphere-trees could be used for both. Finally, another interesting research area would be the construction of more generic bounding volume hierarchies containing many different types of primitive, each one being used where it is best suited.

References

BLUM, H., AND NAGEL, R. 1978. Shape description using weighted symmetric axis features. *Pattern Recognition 10*, 167–180.

BOWYER, A. 1981. Computing Dirichlet tessellations. *The Computer Journal 24*, 2, 162–166.

BRADSHAW, G. 2001. *Bounding Volume Hierarchies for Level-of-Detail Collision Handling*. PhD thesis, Trinity College Dublin, IRELAND.

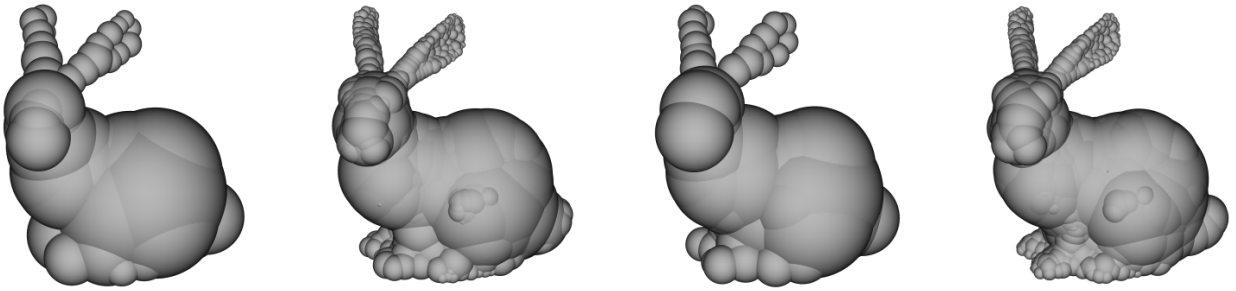
CAMERON, S. 1997. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of the Int. Conf. On Robotics and Automation*, 3112–3117.

COHEN, J., LIN, M., MANOCHA, D., AND PONAMGI, M. 1995. I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. In *Proceedings of ACM Int.3D Graphics Conference*, 189–196.

DINGLIANA, J., AND O'SULLIVAN, C. 2000. Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum, (Proceedings, Eurographics 2000) 19*, 3, 239–247.

GILBERT, E., JOHNSON, D., AND KEERTHI, S. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation 4*, 2, 193–203.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH '96*, 171–180.



(a) Merge - Level 2 (b) Merge - Level 3 (c) Combined - Level 2 (d) Combined - Level 3

Figure 6: Sphere-trees constructed for the Bunny.

HE, T. 1999. Fast collision detection using QuOSPO trees. In *Proceedings of the 1999 Symposium on Interactive 3D graphics*, 55–62.

HUBBARD, P. 1995. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics 1*, 3, 218–230.

HUBBARD, P. 1995. *Collision Detection for Interactive Graphics Applications.* PhD thesis, Dept. of Computer Science, Brown University.

HUBBARD, P. 1995. Real-time collision detection and time-critical computing. In *Workshop On Simulation and Interation in Virtual Environments*.

HUBBARD, P. 1996. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics 15*, 3, 179–210.

INAGAKI, H., SUGIHARA, K., AND N.SUGIE. 1992. Numerically robust incremental algorithm for constructing 3D Voronoi diagrams. In *Proceedings of the* 4^{th} *Canadian Conference on Computational Geometry*, 334–339.

KLOSOWSKI, J., HELD, M., MITCHELL, J., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE transactions on Visualization and Computer Graphics 4*, 1, 21–36.

KRISHNAN, S., GOPI, M., LIN, M., MANOCHA, D., AND PATTEKAR, A. 1998. Rapid and accurate contact determination between spline models using ShellTrees. In *Proceedings of Eurographics '98*, vol. 17(3), 315–326.

KRISHNAN, S., PATTEKAR, A., LIN, M., AND MANOCHA, D. 1998. Spherical shells: A higher order bounding volume for fast proximity queries. In *Proceedings of the 1998 Workshop on the Algorithmic Foundations of Robotics*, 122–136.

LARSEN, E., GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1999. Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Dept. of Computer Science, University of North Carolina.

LIN, M., AND CANNY, J. 1991. Efficient algorithms for incremental distance computation. In *Proc. IEEE Conference on Robotics and Automation*, 1008–1014.

LIN, M. 1993. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley.

MIRTICH, B. 1998. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics 17*, 3, 177–208.

O'SULLIVAN, C., AND DINGLIANA, J. 1999. Realtime collision detection and response using sphere-trees. In *Proceedings of the Spring Conference on Computer Graphics*, 83–92.

PALMER, I., AND GRIMSDALE, R. 1995. Collision detection for animation using sphere-trees. *Computer Graphics Forum 14*, 2, 105–116.

PONAMGI, M., MANOCHA, D., AND LIN, M. 1997. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics 2*, 1, 51–64.

QUINLAN, S. 1994. Efficient distance computation between non-convex objects. In *Proceedings International Conference on Robotics and Automation*, 3324–3329.

RABBITZ, R. 1994. Fast collision detection of moving convex polyhedra. In *Graphics Gems IV*, P. Heckbert, Ed. Academic Press, Cambridge, MA, 83–109.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 343–352.

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Streaming QSplat: A viewer for networked visualization of large, dense models. *2001 Symposium on Interactive 3D Graphics*.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4, 1–13.

WELTZ, E. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, H. Maurer, Ed. 359–370.

WHITE, D. (www). *Smallest Enclosing Ball of Points/Balls*. http://vision.ucsd.edu/ dwhite/ball.html.

WILSON, A., LARSEN, E., D.MANOCHA, AND LIN, M. 1998. IMMPACT: A system for interactive proximity queries on massive models. Tech. Rep. TR98-031, Dept. of Computer Science, University of North Carolina.