Abstract: We present new distance computation algorithms using hierarchies of rectangular swept spheres. Each bounding volume of the tree is described as the Minkowski sum of a rectangle and a sphere, and fits tightly to the underlying geometry. We present accurate and efficient algorithms to build the hierarchies and perform distance queries between the bounding volumes. We also present traversal techniques for accelerating distance queries using coherence and priority directed search. These algorithms have been used to perform proximity queries for applications including virtual prototyping, dynamic simulation, and motion planning on complex models. As compared to earlier algorithms based on bounding volume hierarchies for separation distance and approximate distance computation, our algorithms have achieved significant speedups on many benchmarks.

1 Introduction

The problem of computing minimum separation distance or tolerance verification arises frequently in robot motion planning, dynamic simulation, virtual prototyping, simulation-based design and haptic display. In many cases the performance of the overall application is greatly determined by the performance of distance computation algorithms. For example, applications such as haptic rendering have stringent performance requirements and need to perform distance queries in less than a millisecond on large models composed of tens of thousands of polygons [GLGT99]. Most randomized motion planners spend more than 90% of the overall time in distance computation routines [HKL98, Lat96].

Algorithms for proximity queries, such as collision detection and distance computation, have been extensively studied in the literature. While a number of specialized algorithms have been designed to handle a class of primitives, e.g. convex polytopes, spheres, or ellipsoids, the most general and versatile algorithms are based on bounding volume hierarchies (BVHs). Different BVHs are primarily categorized by the choice of bounding volume (BV) type at each node of the tree. Examples of BV types include spheres, axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs) and discretely-oriented polytopes (k-DOPs). The efficiency of a hierarchy is affected by the choice of a BV type. In particular, there is a trade-off between the speed of BV operations and the “tightness of fit” which affects the number of operations performed.

Most of the earlier work in this area was based on using simple BV types like spheres and AABBs [CLMP95, Hub93, BKSS90, HKM95, Qui94]. In the last few years, many researchers have investigated the use of tight fitting BVs for proximity queries. These include using of OBBs [GLM96, BCG96], spherical shells [KPLM98] and k-DOPs [HKM96, KHM98] for collision detection. For many close proximity configurations, these tight fitting BVs result in improved performance as compared to BVHs composed of spheres or AABBs. One of the commonly used algorithm for distance computation algorithm uses spheres as a BV type [Qui94]. Johnson and Cohen [JC98] have used OBBs for distance computation as well.

1.1 Main Contributions

In this paper, we present a new distance computation algorithm that uses hierarchies of Rectangle Swept Spheres (RSS). This BV corresponds to the set of points obtained by sweeping the center of a sphere over a 3D rectangle, i.e., the Minkowski Sum of an origin-centered sphere and an arbitrarily oriented rectangle. We present a specialized, efficient and accurate algorithm to compute the distance between two RSSs. Furthermore, we present acceleration techniques based on coherence or priority directed search to enhance the performance of distance and approximate distance queries. We demonstrate the performance of our algorithm on different applications. In practice, we achieve considerable speedups as compared to earlier algorithms and implementations of distance computation, based on bounding volume hierarchies.

*Supported in part by ARO Contract DAAD04-96-1-0257, NSF Career Award CCR-9625217, NSF grants EIA-9806027 and DMI-9900157, ONR Young Investigator Award and Intel.
This approach offers us a number of advantages. The RSSs provide a tight fit to the underlying geometry. Moreover, our specialized algorithm for distance computation runs quite fast as compared to computing distance between other tight fitting BVs. Furthermore, we show that as two objects move away from each other, BVHs based on tight fitting BVs result in better asymptotic performance.

1.2 Organization:
The rest of the paper is organized as follows. We survey related work on BVHs and proximity queries in Section 2. Section 3 gives an overview of using BVHs for distance queries. Section 4 introduces the Rectangle Swept Sphere, and presents algorithms for building BVHs and performing proximity queries. In Section 5, we present analysis tools and Section 6 describes acceleration techniques to speed up distance queries. In Section 7, we summarize the performance of our algorithms based on hierarchies of RSS volumes and compare the performance of our implementation with other distance computation algorithms.

2 Related Work
Proximity queries have been extensively investigated for decades by researchers in computer graphics, robotics, physically-based modeling, computational geometry, and computer animation. In this section, we briefly survey some of the related work.

2.1 Computational Geometry
Many asymptotically efficient algorithms for collision detection and distance computation have been proposed by researchers in computational geometry. These include Dobkin-Kirkpatrick hierarchies [DK82], linear programming [Sei90] and algorithms for intersecting convex polytopes [Cha89].

2.2 Distance Computation
Given two convex polytopes, algorithms to compute distance between them have been proposed by Gilbert et al. [GIK88], Lin and Canny [LC91], Cameron [Cam97], and Mirtich [Mir98]. The last three algorithms perform incremental computation to exploit coherence between successive queries. Hamlin et al. [HKT92] present distance computation between a pair of spherically-extended polytopes (S-topes). The S-topes closely resemble the bounding volume shapes described in this paper. For general polygonal models, Quinlan [Qui94] proposed an algorithm using BVHs of spheres and also used them to compute approximate distance. Johnson and Cohen [JC98] used BVHs composed of oriented bounding boxes and also presented techniques to compute distance between NURBS primitives.

2.3 Spatial and Temporal Coherence
In many graphics applications and simulation environments, there is usually little movement of objects between successive frames or small time steps. Therefore, relative configurations between objects often do not change much. Many algorithms have been proposed to exploiting spatial and temporal coherence to further speed up the computation [Bar90, LC91, CLMP95, Cam97, KHM98, Mir98].

3 Distance Computation using Bounding Volume Hierarchies
In this section, we give an overview on bounding volume hierarchies (BVHs), and BVH-based algorithms for distance computation.

A bounding volume (BV) is used to bound or contain sets of geometric primitives, such as triangles, polygons, NURBS, etc. In a BVH, BVs are stored at the internal nodes of a tree structure. The root BV contains all the primitives of a model, and children BVs each contain separate partitions of the primitives enclosed by the parent. Leaf node BVs typically contain one primitive. In some variations, one may place several primitives at a leaf node, or use several volumes to contain a single primitive [Qui94]. The BVHs are used for distance queries in the following manner:

Separation Distance Computation: As the query proceeds, the smallest distance found from comparing primitives is maintained in a variable $\epsilon$. At the start of the query, $\epsilon$ is initialized to infinity, or to the distance between an arbitrary pair of primitives. Each recursive call with BVs $A$ and $B$ must determine if some primitive within $A$ and some primitive within $B$ are closer than, and therefore will modify, $\epsilon$. The call returns trivially if BVs $A$ and $B$ are farther than the current $\epsilon$, since this precludes any primitive pairs within them being closer than $\epsilon$. Otherwise the algorithm is applied recursively to its children. For leaf nodes it computes the exact distance between the primitives and if the new computed distance is less than $\epsilon$, it updates $\epsilon$.

Approximate distance computation: This supposes that a certain relative or absolute error in the distance computation is acceptable. The distance between BVs $A$ and $B$ gives a lower limit to the exact distances between their primitives, and if this bound prevents $\epsilon$ from being reduced by more than the acceptable tolerance, that recursion branch is terminated.

4 Rectangle Swept Spheres
In this section, we present Rectangle Swept Spheres (RSS) for fast distance queries. This BV corresponds to a volume covered by a sphere whose center is swept across the surface of a 3D rectangle. It may also be described as a
rectangle grown uniformly in all directions by some offset. From here on, we will refer to this growth offset, or radius of the swept sphere, as the RSS's radius.

### 4.1 Building BVHs of Rectangle Swept Spheres

In this section, we present algorithms for building BVHs composed of RSSs. It has two parts: enclosing a set of triangles by an RSS and grouping of nested BVs into a single hierarchy.

Given a set of triangles, we use statistical techniques to compute a bounding volume. Our approach is based on first and second order statistics summarizing the vertex coordinates, as used by [GLM96, BCG96]. They are the mean, \( \mu \), and the covariance matrix, \( \Sigma \), respectively [DH73]. Our fitting algorithms use the eigenvectors of the covariance matrix, \( \Sigma \), to initially compute an OBB that encloses the underlying geometry.

For fitting an RSS, the smallest of the three dimensions of the OBB becomes the rectangle normal direction. In most cases, this direction is likely to be perpendicular to a nearly flat cluster of triangles, and will allow the flat shape of the RSS to fit the geometry tightly. The other directions fix the orientation of the rectangle and the rectangle dimensions are grown appropriately to enclose all the geometry. The dimensions of the rectangle are initially determined so that they enclose triangles along the two side projections of the RSS. This may miss triangles near the corners of the rectangle. As a result, the rectangle corners are extended outward at a 45 degree angle until they enclose all the triangles.

We use a top-down strategy to create the nodes of our hierarchy. This means that the hierarchy is built from the root node downward. The triangles in each node of the tree, starting with the root that contains all of the triangles, are split into two subsets that become the children nodes of this node. Nodes are recursively subdivided unless they contain only a single triangle, which corresponds to a leaf node of the hierarchy. Our splitting rule is the same as used for an OBBTree [GLM96]. A splitting axis is chosen, and a plane orthogonal to the axis is used to partition the triangles into two sets, according to which side of the plane their center point lies on.

### 4.2 Computing the Distance Between RSS

In this section, we present a specialized algorithm to compute the distance between two RSS. One may find the distance between two RSS by computing the distance between the core rectangles, and subtracting the sum of their radii. Because the rectangles are convex shapes, one can possibly use algorithms proposed in [GJK88, LC91, Mir98] to compute distances between them. However, we present a specialized algorithm for two reasons:

- **Efficiency**: We will like to minimize the operation count as much as possible for these primitives.

- **Accuracy**: We will like the algorithm to work on all configurations of BV's and not be susceptible to numerical errors and degeneracies.

We make use of a lemma in designing the algorithm for distance computation. First, we define the slab of a pair of distinct points \( a \) and \( b \) as the region bounded by two parallel planes, one plane intersecting \( a \), the other plane intersecting \( b \), with normals \( \mathbf{n} = \mathbf{b} - \mathbf{a} \). For our use in this paper, the slab itself does not include the planes, only the enclosed region. Hence each of the *exterior halfspaces* of the slab includes the plane that borders it.

We say that a slab *divides* two point sets if and only if one point set lies completely in one exterior halfspace, and the other point set lies completely in the other exterior half-space.

Additionally, we define *closest points* of a pair of points \( A \) and \( B \) as any pair of points \((a, b)\) from the Cartesian product of \( A \) and \( B \) such that \(||\mathbf{b} - \mathbf{a}||\) equals the distance between \( A \) and \( B \).

**Lemma**: Consider a pair of points, \( a \) from point set \( A \), and \( b \) from point set \( B \). If \( a \) and \( b \) are coincident, or if \( a \) and \( b \) are distinct and their slab \( S \) divides \( A \) and \( B \), then \( a \) and \( b \) are closest points for \( A \) and \( B \). Furthermore, if \( A \) and \( B \) are convex sets, then the converse is also true.

**Proof** [LGLM99]:

---

Figure 1: *Determining the separation between two rectangles using the normals of their edges.*
4.2.1 Minimum Distance Between Rectangles

Assume that closest points of the rectangles both lie on edges of the rectangles. Then it is possible to find at least one valid closest point pair by computing a closest point pair from each pair of rectangle edges and testing it according to the above lemma. In this section, we present an efficient algorithm for performing this test.

If \( e \) and \( f \) are such candidate edge points on edges \( E \) and \( F \), respectively (Figure 1). We can test whether rectangle \( A \) lies outside their slab by testing whether \( f - e \cdot n \leq 0 \), where \( n \) is a vector perpendicular to edge \( E \) along the rectangle's face. A similar test will show whether \( B \) also lies outside the slab.

However, in practice we use a slightly modified approach which improves efficiency. First, define the exterior halfspace of edge \( E \) as the halfspace defined by any point on \( E \) and the \(-n\) direction (Figure 1). We choose to include the bordering plane as a part of the exterior halfspace. We can implicitly determine whether \( f - e \cdot n \leq 0 \) by whether \( f \) lies in the exterior halfspace of \( E \). We can likewise test the other dot product by determining whether \( e \) lies in the exterior halfspace of \( F \).

The advantage of the reformulation is that one may make each conclusion without explicitly computing the points \( e \) or \( f \). Consider, for example, trying to determine whether \( f \) is contained within \( E \)'s exterior halfspace. We distinguish three cases:

- \( F \) is entirely inside the exterior halfspace of \( E \). Therefore, \( f \) must be in \( E \)'s exterior halfspace (Fig. 2(a)).

- \( F \) is entirely outside the exterior halfspace of \( E \). Thus, \( f \) cannot be in \( E \)'s exterior halfspace (Fig. 2(b)).

- Some points of \( F \) are inside and some are outside \( E \)'s halfspace. In such cases, \( f \) may be inside or outside \( E \)'s halfspace (Fig. 2(c)).

It is easy to detect which of these cases is applicable. Only in the last case more work is needed to determine the location of \( f \). More details of that are given in [LGLM99].

If an edge pair test passes, only do we explicitly compute \( e \) and \( f \) and return the distance between them as the distance between the rectangles.

When no pair of edges passes an edge pair test, we can conclude that for any pair of closest points of the rectangles, one point must be interior to a rectangle face. Let us first assume that the rectangles do not overlap. Because the closest points are distinct, their slab will divide the rectangles. We can show that the slab corresponding to the closest points has a normal aligned with one of the two rectangle face normals [LGLM99]. This gives us only two possible slabs that may correspond to closest points.

Next we apply the second fact, that the thickness of any dividing slab is a lower bound to the distance between two point sets. This means that the larger thickness slab is the only one that can correspond to the closest points of the rectangles, and we can simply return the larger thickness as the distance between rectangles. Now suppose that the rectangles overlap. In this case there is no slab that will divide the rectangles, meaning that both of the separation intervals computed will be zero. Therefore, returning the larger separation as the distance works for overlapping rectangles as well.

4.3 Distance Computation between Primitives

Triangles are commonly used primitives in hierarchy-based proximity queries. We have developed our own specialized algorithm to compute the distance between triangles. The method we use is similar to that used for rectangles, and handles triangles which degenerate to a line segment or point. Initially we consider only non-degenerate triangles.

We begin the algorithm with a set of 9 edge pair tests. We explicitly compute for each edge pair, \((E, F)\), the closest points \( e \) and \( f \). If the points are not coincident, we need to show that their slab \( S \) divides the triangles to prove that these are closest points of the triangles. This technique has been extended to handle degenerate triangles, that may correspond to a line or a point. More details are given in [LGLM99].

4.3.1 Improving the Accuracy

In our application, a smaller than actual distance between the BV's is a conservative result. It does not affect the correctness of the proximity query, but can result in additional BV tests. Therefore, we have handled numerical inaccuracies by lower bounding the distance in some cases. For instance, when testing the edge pairs, several numerical errors may lead to a false conclusion that \( f \) lies within \( E \)'s
exterior halfspace. Whenever the two edges are nearly intersecting, their dot product can be very close to zero. Because such an error can result in a higher than actual distance being reported, we use tolerances to detect such cases and cause the edge pair test to fail. Thus, it is possible that the true closest points may be missed. But the final step of finding the maximal separation along the face normals gives the conservative lower bound to the actual distance that we are looking for.

4.3.2 Performance of RSS Distance Algorithm

We compared the performance of our implementation of distance computation between rectangles with an implementation of Gilbert et al.'s algorithm [GJK88] for general convex objects. It should be noted that this algorithm always returns the closest points, whereas our algorithm often computes only the distance between the rectangles. We tested several hundred thousand rectangle configurations from actual distance queries. We timed the distance computations of these rectangles using each method. We found our specialized algorithm is approximately four times faster.

5 Analyzing Distance Queries

In the previous section, we presented the distance computation algorithm using RSSs. This section introduces a number of concepts that are general to all BV types, including the bounding volume traversal tree (BVTT). These concepts are particularly useful in understanding the distance acceleration techniques we present in Section 6.

We use $\epsilon$ in the following discussion to denote the current estimate on the distance between two models, and $\delta$ to denote the minimal separation distance.

5.1 The Bounding Volume Test Tree

The bounding volume test tree (BVTT) represents the hierarchy of tests performed during a query. Each node in the BVTT corresponds to a single collision or distance test between a pair of BVs. For the query algorithms described in this paper, a BV test (a node in the BVTT) often leads to additional BV tests (its child nodes in the BVTT). The root node of the BVTT is the BV test between the roots of the BVHs. The leaf nodes of the BVTT are either a test between two BVH leaf nodes or a test between a pair of BVs that did not require descent to children BVs (e.g., a pair of BVs that did not overlap in a collision query).

A maximal BVTT results from a query in which no branches are pruned. The resulting tree has $O(mn)$ nodes, where $m$ and $n$ are the respective number of primitives in each model. This may never occur in practice, but it is a useful concept. Note that if the BVHs are fixed over all queries, and a descent rule based on BV diameter is used, the structure of the maximal BVTT does not vary with the configurations of the models.

In order to prune the search at a BV pair test, $\epsilon$ must be smaller than the distance between the BVs. Thus the best pruning of the search will occur if $\epsilon$ is as small as possible. Clearly the smallest $\epsilon$ can be is $\delta$, the true distance between the models. We can call a BVTT searched when $\epsilon$ is always equal to $\delta$ a minimal BVTT. The minimal BVTT provides a lower bound to the set of nodes that must be searched in a distance query, given the assumptions we have made. In an actual query, one hopes that $\epsilon$ can be reduced quickly, to limit exploration outside the minimal BVTT.

6 Accelerating Distance Queries

A simple traversal of the BVTT would be a depth-first search, in which we recursively process a left subtree before processing the right subtree. As mentioned in Section 5, if we visit more distant BV pairs before visiting the closely BV pairs, we may miss opportunities to prune the search tree, and thereby increase the running time. In this section we describe two simple techniques for improving the pruning of the BVTT by causing $\epsilon$ to approach $\delta$ quickly. These two techniques are called priority directed search and triangle caching.

6.1 Priority Directed Search

Instead of traversing the BVTT as a strictly depth-first or breadth-first search [JC98], we use a priority queue to schedule which of the pending tests to perform next. We assign priority to the pending BVTT visits according to the distance: the closest pending BV pair is given a higher priority and visited next.

The goal is to guide the search process so that $\epsilon$ approaches the $\delta$ value after traversing as few BV’s as possible. In this way the search will proceed towards primitives in each BVH that will possibly result in a lower value of $\epsilon$.

However, there are several sources of overhead in using a priority queue. One of them is the space required to store all the candidate BV pairs. The priority queue can have up to $O(mn)$ pairs in the worst case, where $m$ and $n$ are the respective numbers of primitives of the models. Additional overhead comes from insertion, deletion and minimum finding operations in the priority queue. One way to ameliorate this problem is to use a fixed size priority queue. When the queue is full, a recursive call is made on that queue’s closest BV pair. This recursive invocation creates its own new queue, which it uses until its subtree is completely processed. Limiting the queue size limits the worst case storage requirements and performance of the algorithm.
6.2 Triangle Caching

Another approach to improve the performance is to utilize coherence between successive frames. In some applications, the distance only changes slightly between queries. To take advantage of this, the closest triangle pair from the previous query can be recorded, and the distance between them in the next query can be used to initialize $\epsilon$. If the motion between time steps is very small, then $\epsilon$ will be very close to the true minimum distance, $\delta$, and only the minimal BVTT may be searched.

7 Implementation and Performance

In this section, we describe an implementation of our algorithms and compare its performance with other BVH-based public domain libraries for distance queries.

7.1 Implementation

We have implemented all of our algorithms in a C++ library. They are available as part of a general purpose proximity query package called PQP. The floating point type used throughout the library can be set with a "typedef" before compilation.

We have represented the RSS in our implementation with a 3x3 rotation matrix giving the orientation of the rectangle (the first two columns give the directions of the two sides), a vector giving the position of a rectangle corner, the two side lengths of the rectangle, and the radius of the RSS. This is a total of 15 real numbers, requiring 60 and 120 bytes for single and double precision floats respectively. The triangle caching technique is used by default, since it requires little overhead. During a distance query, a model simply stores a pointer to the triangle which was shown closest to the other model input to the query.

7.2 Benchmarks

We used several benchmarks to measure the performance of our library and compare it with other proximity queries libraries.

Cave environment: Path Generation and Verification.

We used a randomized path planner [HKL+98] to plan the path of a wrinkled torus in a cave environment. The wrinkled torus has 20,000 polygons and the cave environment is composed of 50,970 polygons. Given the initial and final position of the torus, the path planner computed a trajectory by probing 2,884 random positions of the torus in the configuration space, computing distance to the environment for each of these positions. Our benchmark consists of the set of 2,884 sampled positions. With our user-specified parameters, the randomized path planner returns a set of 129 steps along the collision-free path of the torus in the cave environment.

Tori Environment: Path Generation and Verification

We again use the randomized path planner, but this time to plan a collision-free path of a small warped torus (1,332 polygons) through the center of a large knobby torus (3,240 polygons). We restrict the configuration space such that the small torus so that it must pass through the center of the large torus to reach its final configuration. In this case, the planner probes 18,850 random positions to find the collision free path, although we restrict our benchmark to the first 1000 steps. The planner returns 236 steps along the collision-free path of the small torus through the center of the larger torus.

7.3 Performance and Comparison

We have applied our distance computation algorithm to a number of applications including path planning, virtual prototyping and dynamic simulation. We also compared its performance with other bounding volumes including spheres, AABBS and OBBs. In most cases, our distance computation algorithm based on RSSs outperforms other those based on other BVs [LGLM99]. In this section, we compare the performance of our system with Quinlan’s distance computation library based on sphere trees [Qui94]. To the best of our knowledge, Quinlan’s system has been widely used in a number of applications.

In addition to using different bounding volume types, these packages also differ in their approaches to hierarchy construction. Our system is currently limited to one leaf BV per primitive, while Quinlan’s library can use more than one leaf sphere to bound a single primitive. The covering of a primitive with spheres is governed by the choice of an $r_{max}$ value which is the maximum allowed radius of a leaf node sphere. While $r_{max}$ can be chosen independently for each model or even for each primitive, we simply chose a single $r_{max}$ value for both of the hierarchies in an application. We then measured the average query performance across several settings of $r_{max}$. We also made one modification to the code written by Quinlan, which was to insert the same triangle distance test as was used in our library, in order to eliminate inconsistencies between the results of the two libraries. Note that relative performance results can be influenced by the cost or accuracy of the primitive test. We performed exact and 0.1 relative error distance queries on all four benchmarks, using a Pentium-II 400MHz Linux machine with 128 MB of RAM. A graph for each benchmark plots the average query speedup of our library over Quinlan’s library across several selected values of $r_{max}$, for both exact and approximate distance queries.

One observes a wide range of performance of Quinlan’s library in our applications using different values of $r_{max}$. Performance levels off once $r_{max}$ exceeds a particular value - at this value, all primitives of both models can be covered by a single sphere, and for any $r_{max}$ above that value, the same two hierarchies are built for the applica-
Figure 3: Cave Planner Benchmark – the horizontal axis corresponds to the \( r_{\text{max}} \) value selected, while the vertical axis corresponds to the average query speedup using our RSS-based system over Quinlan’s. These timings are generated without priority directed search or triangle caching. They compare the relative performance of different bounding volumes and corresponding distance tests.

Reduction of \( r_{\text{max}} \) below this point at first improves the performance of Quinlan’s library. However, performance of Quinlan’s library diminishes if \( r_{\text{max}} \) is made too small. We are probably seeing cache effects as the number of spheres in each hierarchy starts to strain available memory. In several of the graphs, reducing the lowest \( r_{\text{max}} \) value by half caused an out of memory exception.

Our system allows the user to turn on priority directed search by supplying a priority queue size to potentially improve performance. In general, it improves the performance of applications without spatial coherence between queries (e.g. path generation using randomized path planner). In these cases, we fix our queue size at an arbitrary 150, which we have seen to give a performance benefit in the appropriate applications.

Our system has also been used for robot motion planning [PHLM00].

8 Conclusions

In this paper, we have presented a new distance computation algorithm between general polygonal models. The algorithm makes use of hierarchies of RSSs. We have presented efficient and accurate algorithms to compute the hierarchies and use them for distance computation. We have also presented acceleration techniques based on coherence and priority directed search. The resulting algorithm has been implemented and applied to a number of applications. We have also compared its performance with an earlier implementation that used sphere as a bounding volume. This algorithm has also been used for robot motion planning [PHLM00].

References


Appeared in Proc. of IEEE Int. Conf. on Robotics and Automation 2000