Available as technical report TR99-018, Department of Computer Science, UNC Chapel Hill

Fast Proximity Queries with Swept Sphere Volumes *

Eric Larsen Stefan Gottschalk Ming C. Lin Dinesh Manocha
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{larsen,stefan,lin,dm}@cs.unc.edu
http://www.cs.unc.edu/~geom/SSV

Abstract: *We present novel algorithms for fast proximity queries using swept sphere volumes. The set of proximity queries includes collision detection and both exact and approximate separation distance computation. We introduce a new family of bounding volumes that correspond to a core primitive shape grown outward by some offset. The set of core primitive shapes includes a point, line, and rectangle. This family of bounding volumes provides varying tightness of fit to the underlying geometry. Furthermore, we describe efficient and accurate algorithms to perform different queries using these bounding volumes. We present a novel analysis of proximity queries that highlights the relationship between collision detection and distance computation. We also present traversal techniques for accelerating distance queries. 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*

*Supported in part by ARO Contract DAAH04-96-1-0257, NSF Career Award CCR-9625217, NSF grants EIA-9806027 and DMI-9900157, ONR Young Investigator Award and Intel.

have achieved significant speedups on many benchmarks.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling **Key Words and Phrases:** collision detection, distance computation, coherence, computational geometry, physically-based modeling, swept volumes

1 Introduction

Many applications of computer graphics or computer simulated environments need to determine spatial or proximity relationship between two geometric objects. The common set of proximity queries include

- **Collision detection** – given two or more objects, determine if a geometric contact has occurred between them.
- **Separation Distance Computation** – if two objects are disjoint, find the minimum Euclidean distance between them.
- **Approximate Distance** – given an error tolerance value, compute the separation distance to within the precision of the specified tolerance.

Such queries frequently arise in applications involving dynamics simulation, virtual prototyping, computer animation, surgical simulation, robot motion planning, simulation-based design, haptic rendering, molecular modeling, and computer games. Some applications, such as haptic rendering, have stringent performance requirements. They need to perform these queries in less than a millisecond on large models composed of hundreds of thousands of polygons.

Algorithms for such queries have been extensively studied in the literature. While a number of specialized algorithms have been designed to handle a pair of a special 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), discretely-oriented polytopes (k-DOPs), etc. The efficiency of a hierarchy is affected by the choice of a BV type. More precisely, there is a trade-off between the "tightness of fit" and the speed of operations between two such volumes. It is clear that no single bounding volume type gives the optimal performance for all proximity queries in all scenarios. For example, hierarchies based on spheres perform well on collision queries between well-separated objects. But when the objects are in close proximity, hierarchies based on tighter fitting bounding volumes, e.g. k-DOPs or OBBs, may result in superior performance.

Ideally, one may like to combine multiple BV types in a single hierarchy. However, there are a number of issues in designing hierarchies of mixed bounding volumes, such as selecting a particular BV type at a given node in the tree for a particular query. We also need to perform the geometric queries between them efficiently and accurately. If we use hybrid hierarchies composed of n different bounding volumes, we may have to implement $O(n^2)$ different tests to check them for overlap or compute distance between them.

Many applications such as dynamics simulation, haptic rendering and path planning make use of collision detection, distance computation and approximate distance queries. Because of issues related to software engineering and memory usage, we would like to use one BVH to perform all these queries efficiently.

1.1 Main Contributions

In this paper, we present hierarchies of swept sphere volumes for fast proximity queries. The BVs at the nodes of a BVH belong to a family of three different *swept sphere volumes*. They correspond to a sphere (also known as *point swept sphere or PSS*), and more complex volumes obtained by sweeping along either an arbitrarily oriented line (*line swept sphere or LSS*) or along an arbitrarily oriented rectangle (*rectangle swept sphere or RSS*). While it is least expensive to perform the proximity queries on PSSs, the RSSs provide the tightest fit to the underlying geometry. We present specialized, efficient and accurate algorithms to compute distance between any combination of these BVs.

Much of our study has involved applying swept sphere volumes to distance computation. We present some observations about the distance problem that may help to explain the costly performance of distance queries relative to collision queries, as well as how the choice of BV impacts distance queries. Specifically, we illustrate that each distance query between geometric models can be answered by performing collision tests between BVs and computing exact distance between the primitives. Furthermore, we demonstrate the effectiveness of utilizing coherence or priority directed search to enhance the performance of distance queries.

Combining these approaches, we present an algorithmic framework that uses swept sphere volumes as BVs and can be used for fast collision detection, separation distance as well as approximate distance computation. Some of the advantages of our approach include:

1. Good efficiency and accuracy in performing various proximity queries using just one type of BV.

2. The flexibility of including one or more of the three BV types in a BVH.

3. A potentially useful tradeoff between tightness of fit to underlying geometries and efficiency of BV operations.

4. An efficient approach for distance queries that takes advantage of coherence between successive frames and uses priority directed search.

5. As compared to BVHs composed of spheres, AABBs and OBBs, our distance computation algorithms have achieved significant speedups on different benchmarks.

Our benchmarks include examples of virtual prototyping, dynamics simulation, and motion planning on complex models composed of tens of thousands of polygons.

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 various proximity queries and the cost analysis of proximity queries. Section 4 introduces the family of swept sphere volumes, and presents algorithms for building BVHs and performing proximity queries. In Section 5, we present a novel analysis of proximity queries and show how to transform distance computation to collision detection using BVHs Section 6 describes acceleration techniques for the distance queries. In Section 7, we summarize the performance of our algorithms based on hierarchies of swept sphere volumes and compare their performance against the traditional BV types on different benchmarks.

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. Besides collision detection and distance computation, there is considerable work on BVHs for ray-tracing, visibility culling, boolean set operation, and intersection tests. 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 Hierarchical Data Structures

Hierarchical data structures, such as hierarchical spatial partitions and BVHs have been widely used to design efficient algorithms. Typical hierarchies include k-d trees and octrees, R-trees and their variants, cone trees, BSPs [NAT90] and their extensions to multi-space partitions [WG91], and spatial representations based on space-time bounds or four-dimensional tests [AANJ94, Hub93]. The set of BVs for BVHs include spheres [Hub93, Qui94], axis-aligned bounding boxes [BKSS90, HKM95], oriented bounding boxes [GLM96, BCG+96], approximation hierarchies based on S-bounds [Cam91], spherical shells [KPLM98], and k-DOPs [HKM96, KHM+98]. There is also literature on the use of spatial partitioning structures and BVHs to accelerate ray-tracing. Check out Arvo and Kirk [AK89] for a survey.

2.3 Distance Computation

Given two convex polytopes, algorithms to compute distance between them have been proposed by Gilbert *et al.* [GJK88], Lin and Canny [LC91], Cameron [Cam97], and Mirtich [Mir98]. The last three algorithms are incremental and 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. For parametric or implicit surfaces whose motion can be expressed as a closed form function of time, Snyder *et al.* [Sea93] have presented algorithms based on interval arithmetic to compute distance between them.

2.4 Hybrid Combinations

Several researchers have proposed hybrid combinations to accelerate proximity queries. Some of the public domain collision detection systems like SOLID [Sys97] and V-COLLIDE [HLC+97] make use of AABBs for N-body tests and a combination of AABBs or OBBs for pairwise primitive tests. Recently, Gregory *et al.* and Kim *et al.* [GLGT98, KGS98] used combinations of spatial partitioning data structures and BVs for faster collision tests.

3 Bounding Volume Hierarchies

In this section, we give an overview on bounding volume hierarchies (BVHs), the cost equation to measure their performance on proximity queries, and BVH-based algorithms for collision detection and 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 different proximity queries in the following manner:

Collision Detection: Two models are compared by recursively traversing their BVHs in tandem. Each recursive step tests whether BVs A and B, one from each hierarchy, overlap. If A and B do not overlap, the recursion branch is terminated. But if A and B do overlap, the enclosed primitives may overlap and the algorithm is applied recursively to their children. If A and B are both leaf nodes, the primitives within them are compared directly.

Separation Distance Computation: The structure of the query is very similar to the collision query. As the query proceeds, the smallest distance found from comparing primitives is maintained in a variable ϵ. At the start of the query, ϵ 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, ϵ. The call returns trivially if BVs A and B are farther than the current ϵ, since this precludes any primitive pairs within them being closer than ϵ. 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 ϵ, it updates ϵ.

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 ϵ from being reduced by more than the acceptable tolerance, that recursion branch is terminated.

3.1 Assumptions

The performance of BVHs on proximity queries is governed by a number of design parameters. These include techniques to build the trees, number of children each node can have, and the choice of BV type. An additional design choice is the *descent rule*. This is the policy for generating recursive calls when a comparison of two BVs does not prune the recursion branch. For instance, if BVs A and B failed to prune, one may recursively compare A with each of the children of B, B with each of the children of A, or each of the children of A with each the children of B. This choice does not affect the correctness of the algorithm, but may impact the performance.

We assume that all BVHs are binary trees and each leaf node consists of a single triangle. As part of a pre-process, all polygonal models are triangulated. We use a simple top-down scheme to construct the hierarchies and traverse them. Our descent rule is to recursively compare the "smaller" BV with the children of the "larger" BV, where relative size is determined by diameter. More details are given in Section 5.

3.2 Cost of Proximity Queries

Many researchers have provided metrics for evaluating the performance of BVHs. The commonly used cost equation is [GLM96, KHM+98]:

$$T = N_{bv} \times C_{bv} + N_p \times C_p, \tag{1}$$

where T is the total cost function for proximity queries, N_{bv} is the number of bounding volume pair operations, and C_{bv} is the total cost of a BV pair operation, including the cost of transforming each BV for use in a given configuration of the models, and other per BV-operation overhead. N_p is the number of primitive pairs tested for proximity, and C_p is the cost of testing a pair of primitives for proximity (e.g. overlaps or distance computation).

Typically for tight fitting bounding volumes, e.g., RSS, OBBs or k-DOPs (where k is high), N_{bv} and N_p are relatively low, whereas C_{bv} is high. In contrast, C_{bv} is low while N_{bv} and N_p may be higher for simple BV types like spheres and AABBs. Due to these opposing trends, no BV yields optimum performance for proximity queries in all situations. A major motivation in the design of hybrid hierarchies is to include simple shapes like PSS for fast overlap tests and tight fitting BVs like RSS to reduce the number of tests. The LSS may provide a trade-off between the PSS and the RSS, in terms of tightness of fit and the cost of overlap test.

4 Swept Sphere Volumes

In this section, we present a new family of BVs for fast proximity queries. These BVs correspond to geometric shapes composed of a core primitive shape grown outward by some offset. Any such shape is also the Minkowski sum or convolution of the core primitive shape and a sphere. In other words, the BV corresponds to the set of points swept out by the sphere as its center is moved over all the points of the core primitive shape. From here on, we will refer to the growth offset as a *radius*.

In this paper, we use a family of three BVs for proximity queries. These are:

- **Point Swept Sphere or PSS:** The core primitive shape is a point. The resulting BV corresponds to a sphere. We represent it using a point and a radius.
- **Line Swept Sphere or LSS:** The core primitive shape is a line segment. The resulting BV forms a cylinder with hemispherical caps at each end. We represent it using a line segment, the center and the radius.
- **Rectangle Swept Sphere or RSS:** The core primitive shape is an arbitrary rectangle in 3D. The resulting BV is like a rounded box. We represent it using the rectangle, its center, and a radius.

The idea of using any one of these shapes in proximity queries is not novel. For examples, spheres or PSS have been widely used for collision detection [Hub93] and distance computation [Qui94]. The LSS closely resembles a cylinder and many researchers have proposed using cylinders for proximity queries. Xiao and Zhang [XZ96] highlight all the

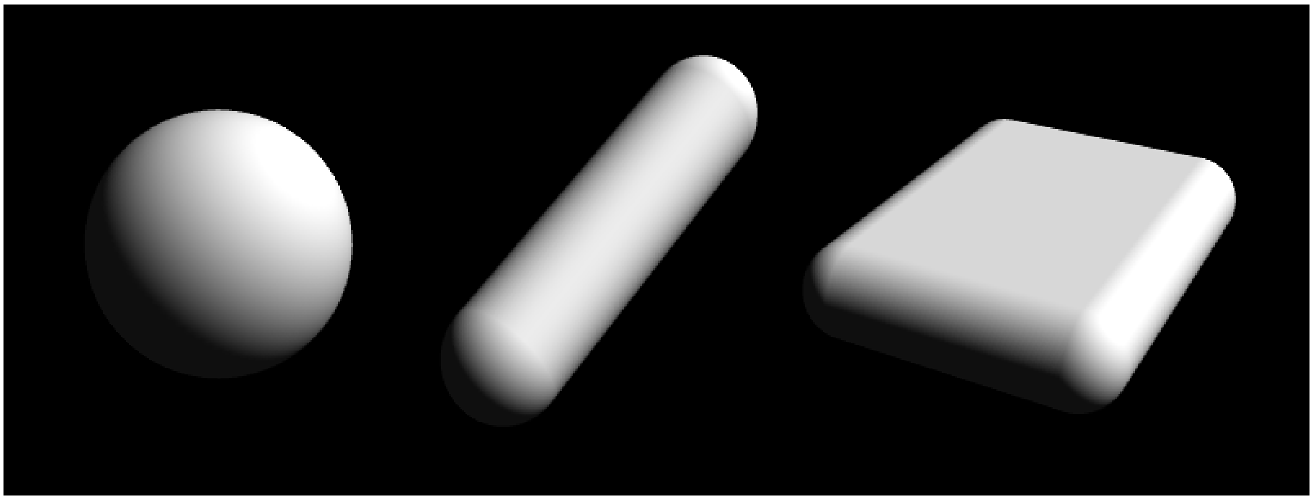

Figure 1: *A rendering of different swept-sphere volumes; a sphere is on the left, a line swept sphere in the center, and a rectangle swept sphere on the right.*

three shapes in describing a method for contact determination with location uncertainty. However, we present these shapes as a family of BVs and this has a number of advantages for proximity queries. These include:

1. Efficient Proximity Queries: It is relatively simple to perform different proximity queries. It involves computing the distance between the primitive core shapes and subtracting the offset radius of each BV. The same set of operations can be used for collision detection, distance computation or approximate distance computation between any pair of these BVs.

2. Varying Tightness of Fit: Different BVs provide a varying tightness of fit to the underlying geometry. The RSS is the tightest BV among them. The pruning power of RSS's is similar to that of OBBs proposed in [GLM96]. As a result, they provide local quadratic convergence to the underlying geometry. PSS has a linear convergence to the underlying geometry. The pruning power of LSS lies somewhere between that of a PSS and a RSS.

3. Hybrid Combinations: It is possible to construct hybrid trees where different nodes may correspond to different swept sphere volumes. For example, for some shapes (e.g. fat or symmetric objects) the spheres or PSS provide a good fit to the underlying geometry. Many CAD environments are composed of lots of pipes, and one can possibly use LSS as the BVs for such shapes. For configurations in which surfaces are close and parallel, one may prefer RSS as a choice for the BV. Furthermore, one can either use static or dynamic selection schemes to choose the appropriate BVs.

4. Low Storage Requirements: The PSS, LSS and RSS take 16, 32, and 48 bytes respectively to represent, assuming single-precision floating point arithmetic.

4.1 Building BVHs of Swept Sphere Volumes

In this section, we present algorithms for building BVHs composed of swept sphere volumes. It has two parts: enclosing a set of triangles by PSS, LSS or RSS and grouping of nested BVs into a single hierarchy.

Given a set of triangles, we use statistical techniques to compute different bounding volumes. Our approach is based on first and second order statistics summarizing the vertex coordinates, as used by [GLM96, BCG+96]. They are the mean, μ, and the covariance matrix, $\mathbf{C}$, respectively [DH73]. Let the vertices of the i'th triangle be the points $\mathbf{a}^i$, $\mathbf{b}^i$, and $\mathbf{c}^i$, then the mean and covariance matrix can be expressed in vector notation as:

$$\mu = \frac{1}{3n}\sum_{i=0}^{n}(\mathbf{a}^i + \mathbf{b}^i + \mathbf{c}^i),$$

$$\mathbf{C}_{jk} = \frac{1}{3n}\sum_{i=0}^{n}(\overline{\mathbf{a}}^i_j\overline{\mathbf{a}}^i_k + \overline{\mathbf{b}}^i_j\overline{\mathbf{b}}^i_k + \overline{\mathbf{c}}^i_j\overline{\mathbf{c}}^i_k), \qquad 1 \leq j,k \leq 3$$

where n is the number of triangles, and $\overline{\mathbf{a}}^i = \mathbf{a}^i - \mu$, $\overline{\mathbf{b}}^i = \mathbf{b}^i - \mu$, $\overline{\mathbf{c}}^i = \mathbf{c}^i - \mu$. The last equations represent subtracting the mean from each row of $\mathbf{a}^i$, $\mathbf{b}^i$ and $\mathbf{c}^i$. Our fitting algorithms use the eigenvectors of the covariance matrix, $\mathbf{C}$, to initially compute an OBB that encloses the underlying geometry. We compute different BVs as:

PSS: A number of efficient algorithms are known in computational geometry for computing the minimum enclosing sphere for a set of points [Wel91].

LSS: We use the largest dimension of the OBB as the central axis of the LSS. All the triangle vertices are projected onto a plane perpendicular to this axis, and a circle is computed that contains all of these projected vertices. This circle determines the position of our central axis and the radius of the LSS. Then we compute extremal vertices along the central axis and use them to compute the hemispherical caps for the LSS.

RSS: For fitting an RSS, the smallest of the three dimensions of the OBB becomes the rectangle normal direction. In most cases, this direction is most likely to be the 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 Hybrid Hierarchies

The main idea behind hybrid hierarchies is to use different BVs for the same query, making selections of BV type at either BVH building time or query time. For selection during a query, each BVH node can contain all 3 BV types, and an algorithm can dynamically select which two types of BVs (among all nine combinations) will be tested. Alternatively, a simple static selection criteria fits only one type of BV to each node in the BVH during hierarchy building. One strategy is to base the choice of a BV type on the relative dimensions of the axes of the OBB that is initially fit to the triangles. If the OBB has three axes of similar lengths, a PSS is used. If one axis length is large and the other two are relatively small, we use an LSS. In all the other cases, we use an RSS. In our current implementation, we have chosen a factor of two to decide when one dimension is larger than the other.

4.3 Proximity Tests

In this section, we present specialized algorithms to compute the distance between the primitive core shapes: points, lines, or rectangles. These are all convex shapes and one can

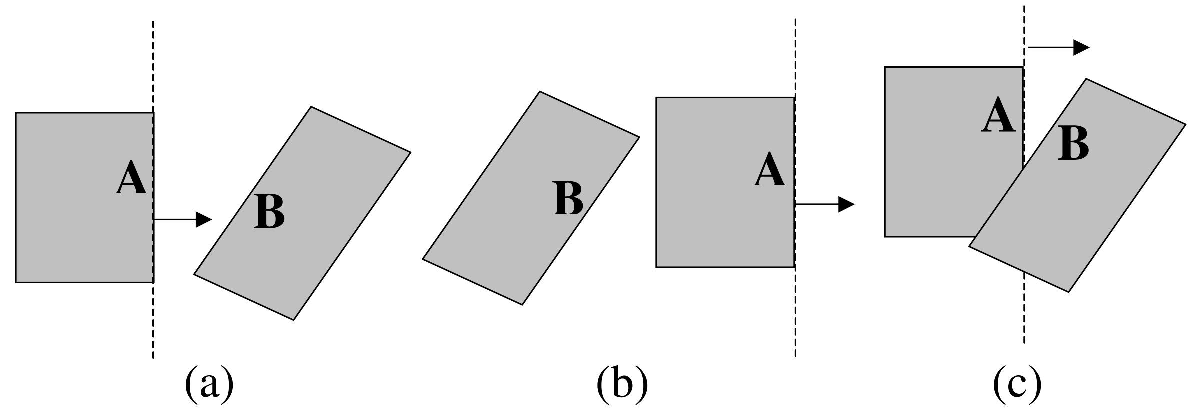

Figure 2: *Different relative configurations of two edges*

possibly use algorithms proposed in [GJK88, LC91, Mir98] to compute distances between them. However, we present specialized algorithms for three 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.
- **Ease of Implementation:** Given that we are using a family of three BV's, we will like to implement all the overlap tests using a similar set of operations.

Our algorithms utilize properties of external Voronoi regions for proximity queries, as proposed by Lin and Canny [LC91]. We initially present an algorithm for computing distance between two arbitrary rectangles in 3D. Later we use subroutines from this algorithm to handle distance computation between the other combinations of points, lines, and rectangles. Our algorithm first determines if the closest points between the rectangles lie on the boundary edges. If so, the distance between these points is computed and returned. If not, one of the two closest points must lie in the interior of a rectangle.

4.3.1 Closest Points on Edges

We use the external Voronoi regions of rectangles to verify that the closest points are found on the edges of the rectangles. The external Voronoi region of an edge E is defined as the region of space outside the rectangle in which all points are closer to E than to any other features of the rectangle. In our formulation, the vertices are not treated as separate features and therefore, the external Voronoi region of E is a half-space. Consequently adjacent edges have overlapping Voronoi regions. The region of overlap is the set of points closest to a corner vertex and thus equally distant to the adjacent edges.

To determine whether the closest points between the rectangles lie on their edges, all 16 pairs of edges (one edge from each rectangle) are examined. To verify whether an individual pair of edges contains the closest points of the rectangles, we use the following lemma:

Lemma 1 *A point* $\mathbf{a}$ *of edge* A *and a point* $\mathbf{b}$ *of edge* B *are the closest points of edges* A *and* B*, and* $\mathbf{a}$ *and* $\mathbf{b}$ *are the closest points for the rectangles.* $\Longleftrightarrow$ *The point* $\mathbf{a}$ *is in the external Voronoi region of* $\mathbf{B}$*, and* $\mathbf{b}$ *is in the external Voronoi region of* $\mathbf{A}$*.*

Hence for every edge pair (A, B), one can compute the closest points $\mathbf{a}$ and $\mathbf{b}$, and each such point can be checked whether it is contained in the external Voronoi region of the edge. However, computing the closest points on possibly all 16 edge pairs can be relatively expensive.

To speed up the computation, we check the relative configurations of two edges without explicitly computing the closest points. Consider when we are trying to determine whether $\mathbf{b}$, the closest point on edge B to A, is contained within A's Voronoi region. In some cases, this can be trivially answered. There are three cases:

- B is entirely inside the Voronoi region of A. Therefore, $\mathbf{b}$ must be in A's Voronoi region (Fig. 2(a)).

- B is entirely outside the Voronoi region of A. Thus, $\mathbf{b}$ cannot be in A's Voronoi region (Fig. 2(b)).

- Some points of B are inside and some are outside A's Voronoi region. In such cases, $\mathbf{b}$ may be inside or outside A's Voronoi region (Fig. 2(c)).

Our algorithm performs two such tests for each edge pair, taking advantage of the trivial acceptance and rejection permitted by the first two cases. The last case needs extra analysis for closest point test. When an edge pair passes both the tests, the algorithm computes $\mathbf{a}$ and $\mathbf{b}$, and returns distance between them as the distance between the two rectangles.

4.3.2 Other Cases

It is possible that no pair of edges will satisfy Lemma 1. In such cases, either the rectangles are overlapping or the closest point lies in its interior. To compute the distance in such cases, we use a variation of separating axis test proposed in [GLM96]. The main idea is to project each rectangle to a unit direction and compute the distance between resulting intervals. This distance along a direction is a lower bound to the actual distance.

When one of the closest points is in the interior of a rectangle, that rectangle's normal vector gives the direction along which the maximal separation occurs. To compute it, we consider the normal direction of each rectangle and find the separation along each of them and take the maximum. If both these distances are zero, the resulting rectangles are overlapping.

4.3.3 Other Distance Routines

The algorithms to compute distances between other primitives, e.g. points, lines and rectangles follow from this rectangle-rectangle distance test. In particular, a line-rectangle distance algorithm turns out to be a special case of this algorithm. All other queries: line-line, point-rectangle, point-line, point-point are all subroutines in the rectangle-rectangle distance test.

4.3.4 Improving the Accuracy

In our applications, a lower bound to the actual distance between the BV's is a conservative result. It doesn't affect the final outcome of the proximity query, but can increase the values of N_{bv} and N_p in the cost equation (1). In our implementation, we have made the algorithm conservative based on this realization. For instance, when division by a small number can cause the third case in the inclusion test (Fig. 2(c)) to fail. This corresponds to an ambiguity of the closest edges, which can be avoided by simply computing the separation in the final step.

4.3.5 Performance of Proximity Tests

We have implemented these overlap tests to compute distance between different core primitive shapes. The relative performance of different tests is shown in Table 4.3.5. These timings were produced by performing proximity queries without any triangle comparisons

Primitive Pairs	Relative Costs
Point-point	1
Point-line	1.25
Point-rectangle	1.53
Line-line	1.83
Line-rectangle	2.39
Rectangle-rectangle	3.10

Table 1: *Relative performance of distance tests between primitive core shapes*

and dividing the absolute time by the number of BV tests. Thus, they indicate the relative cost of each operation including some per operation system overhead. We also compared performance with an implementation of Gilbert et al.'s algorithm [GJK88], which was not specifically optimized for rectangles in 3D. We tested a range of rectangle configurations by extracting a sequence of more than $700,000$ rectangle pairs tested in several hundred distance queries taken from two applications. We timed the distance computations of these rectangles using each method. We found our specialized algorithm is approximately four times faster.

5 Analyzing Proximity Queries

In the previous section, we introduced a family of swept sphere volumes. This section includes many of the observations made while applying these volumes to collision and distance queries. A number of concepts are introduced that are general to all BV types, including the bounding volume traversal tree (BVTT) as well as the concept of transforming a distance query to a collision query.

This discussion primarily benefits intuition on several aspects of distance query performance. We use ϵ in the following discussion to denote the current estimate on the distance between two models, and δ 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. We will introduce several variants of the BVTT that will be helpful in the

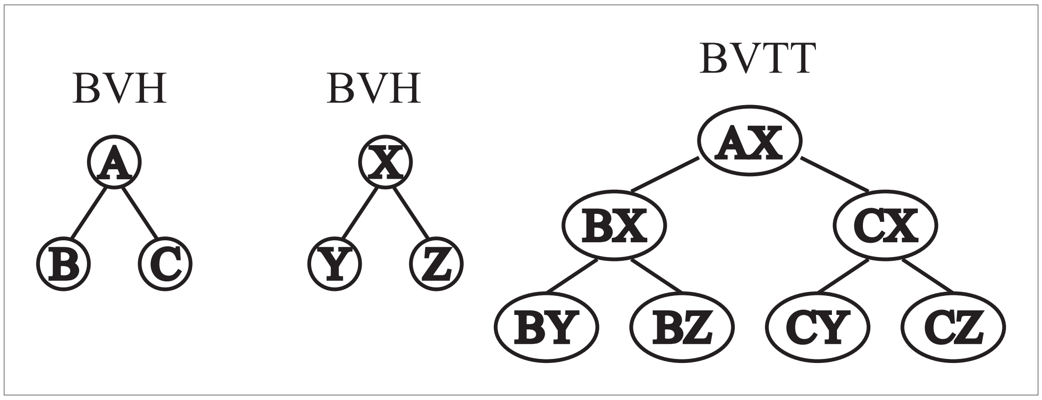

Figure 3: *An example BVTT derived from two BVHs*

discussion to follow.

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 else a test between a pair of nodes which did not overlap, which prunes their subtree from the BVTT.

A *maximal BVTT* results from a query in which no branches are pruned. The resulting tree has $O(n^2)$ nodes. 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.

One may look at the actual BVTT generated for a query as a search path through a subset of the maximal BVTT. For both distance and collision, the structure of this search is governed by the configuration of the models. However, for distance computation, the use of the distance estimate ϵ in pruning choices also influences the degree to which the maximal BVTT's structure is searched.

Recall that in order to prune the search at a BV pair test, ϵ must be smaller than the distance between the BVs. Thus the best pruning of the search will occur if ϵ is as small as possible. Clearly the smallest ϵ can be is δ, the true distance between the models. We can call a BVTT searched when ϵ is always equal to δ 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 ϵ can be reduced quickly, to limit exploration outside the minimal BVTT.

5.2 Transforming Distance Computation to Collision Detection

A collision query proceeds by testing two BVs for overlap, performing further recursive tests if they touch, but terminating the recursion branch if they are disjoint. A distance query has the same basic structure, except that the test between BVs measures the distance, and termination occurs only if the BVs are separated by more than the current distance estimate, ϵ. There is a natural correspondence between a distance query problem and a modified collision detection problem.

To explain the transformation, we use the notion of dilating a shape. *Dilation* of a shape S by ρ means to take its Minkowski sum with the origin-centered sphere of radius ρ. In the rest of this section, we use subscripts to denote dilations. Consider a bounding volume hierarchy H. Let the dilated BVH H_ρ be a BVH like H, except that each of its primitives and BVs have been dilated by distance ρ.

The minimal BVTT for an exact distance query between two BVHs A and B is identical to the BVTT for the collision query between A_α and B_β where $\alpha + \beta = \delta$ and δ is the distance between the models A and B. Consider that for a distance query, a test between BVs or between primitives a and b prunes if the following condition is satisfied:

$$\mathrm{Dist}(a, b) \; > \; \epsilon. \tag{2}$$

The corresponding test in the dilated collision query prunes its recursion branch if

$$a_\alpha \cap b_\beta = \emptyset. \tag{3}$$

These two expressions are logically equivalent. Equation 2 is a fundamental test in a distance query, while equation 3 is an overlap test. Since the corresponding tests between the distance and dilated collision queries are identical, the same recursion branches will be terminated, and consequently the BVTTs are identical.

The arithmetic and logic required to perform the dilated collision query is exactly the same as that required by the distance query. So, the transformation from distance to collision does not make the task computationally simpler. However, it permits us to bring to bear on the distance computation problem some of the analytic results we know about collision queries. For example, Gottschalk *et al.* [GLM96] assert that tighter-fitting bounding volumes can enhance performance in certain situations. Also, theoretical bounds on the complexity of intersection problems concerning "fat objects" may apply [OvdS96, ZS99], since these dilated BVs and primitives are considered "fat".

5.3 Approximate Distance and Dilated Collision Detection

In an approximate distance query, the pruning condition is

$$\epsilon/(1+\mathcal{R}) < \mathrm{Dist}(A, B), \tag{4}$$

where R is the relative error tolerance. The minimal BVTT is produced when ϵ is always equal to δ, i.e., when each BV pair distance is compared against $\delta' = \delta/(1+\mathcal{R})$.

This corresponds to a collision query in which one of the models is dilated by δ'. Provided the original models did not already overlap, dilation by δ' will not bring the models into contact. Since the dilation depends on the relative error $\mathcal{R}$, one may see a range of performance between a distance and collision query by changing the value of $\mathcal{R}$. As $\mathcal{R} \to 0$, the query time for approximate distance computation increases. Quinlan [Qui94] had also noted this behavior empirically.

5.4 Observations concerning Dilated Collision Detection Problem

Consider that the collision detection problem resulting from this dilation always contains a contact, because we are dilating by the minimal distance between the original models. This has several important consequences. First, although one might think that distance computation is less expensive on well-separated objects, the dilation is always equal to the separation distance, so the dilation always brings two primitives back into contact. Increasing the distance merely increases the dilation, so that the dilated models remain in contact. Consequently, the complexity of the problem is not reduced by increasing separation – in fact, we will argue that the complexity is often increased.

Consider what happens as the BVs and primitives of a given BVH are dilated by ever greater amounts. In some circumstances, this increases the number of overlapping BVs between the dilated BVHs. We illustrate this concept in the plane in Figure 4. As the separation distance δ between parallel rows of primitives (i.e. the line segments) is increased, the BV on the left is dilated by ever greater amounts. As the dilation radius

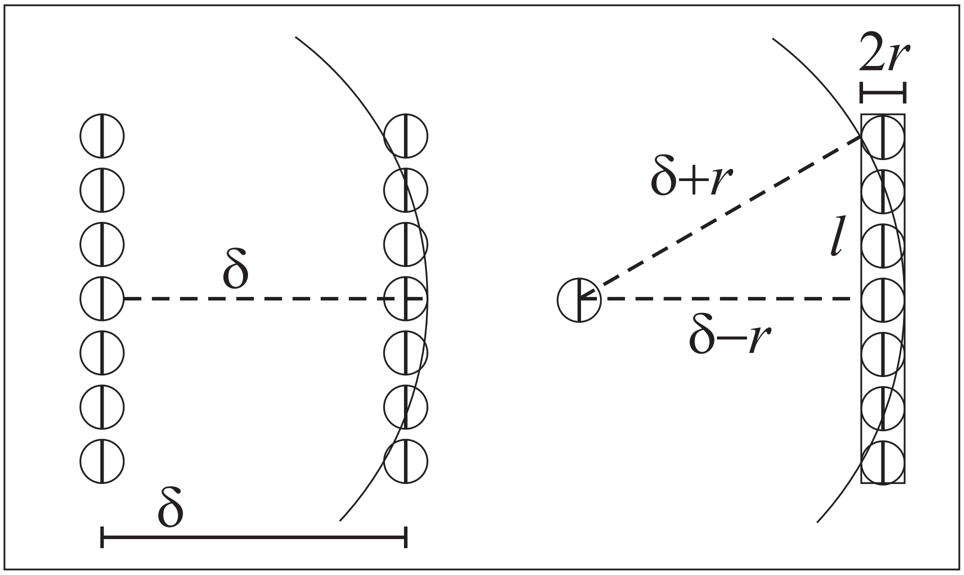

Figure 4: *This figure illustrates how number of BV tests in a distance query can increase as parallel surfaces are separated.*

increases, the dilated BV touches more of the BVs of the other BVH. Expressed more formally, consider the BVs of radius r, and let the primitives be separated by distance δ. The row of nodes in the right can be approximate by a rectangle of thickness $2r$. Observing the side lengths of the right triangle, we deduce that a swath of length $2l$ of the BVs on the right are touched by the dilated BV on the left, where

$$l = \sqrt{(\delta + r)^2 - (\delta - r)^2} = 2\sqrt{r\delta}$$

Since the BVs have radius r, there are $l/(2r)$ of them in the swath. So the number of BVs on the right touched by the dilated BV on the left is

$$n = l/(2r) = \sqrt{\delta/r}$$

Thus, as the separation δ between the parallel rows of primitives increases, the number of BVs on the right touched by each dilated BV on the left also increases. This example is intended to demonstrate the trend relating to a two dimensional problem. The situation is much more complicated for tessellated surfaces in space, but a similar trend can be observed : the number of BV tests, N_{bv}, for an exact distance query tends to increase as we separate approximately parallel surfaces.

Another observation is that the ideal distance query between models A and B is that its complexity begins to resemble an ordinary (not dilated) collision query between A and B as the separation distance δ approaches zero. This is trivially true when $\delta = 0$, in which case the dilation is zero. We use many of these observations in Section 7 to analyze our results on different benchmarks.

6 Distance Query Acceleration Techniques

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 closeby BV pairs, we may miss opportunities to prune the search tree, and thereby perform poorly. In this section we describe two simple techniques for improving the pruning of the BVTT by causing ϵ to approach δ 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 prioritize 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 ϵ approaches the δ 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 ϵ.

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 $O(n^2)$ pairs in the worst case, where n is the number of primitives in each model. 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 ϵ. If the motion between time steps is very small, then ϵ will be very close to the true minimum distance, δ, and only the minimal BVTT may be searched.

6.3 Combining Priority Search and Triangle Caching

Since triangle caching and priority directed search have similar impacts on the pruning, their effects are not additive: if triangle caching has already initialized ϵ close to δ, then our traversal will prune well regardless of which branches we take, and consequently priority directed search is rendered less effective. This has been demonstrated in Table 5 in Section 7.

7 Implementation and Performance

In this section, we describe implementation of our algorithms and compare their performance with other bounding volumes on different benchmarks. We also analyze their performance and try to highlight situations where they may perform better. Overall, our collision detection algorithms perform favorably as compared with earlier approaches. The acceleration techniques described in Section 6 improve the performance of distance queries in most cases. Overall, our new algorithms for separation and approximate distance computation have achieved significant speedups on many benchmarks as compared to other BVs and traversal schemes.

7.1 Implementation

We have implemented all our algorithms as part of a general purpose framework for performing proximity queries using BVHs. The framework has been implemented in C++ and runs on top of PC's and SGI workstations. Each BV type is inherited from an abstract BV class, which defines a standard interface for all BV-centered operations, such as fitting the BV to a set of triangles, and performing distance and overlap operations on the BV. The desired BV type is specified at run time prior to building the hierarchy for an input model. Moreover, the priority based search method uses a binary heap for extraction of the closest pending BV pair.

7.2 Benchmarks

We used a number of benchmarks to measure the performance of swept sphere volumes and compare them with other BVs. The main benchmarks come from different applications. Our goal was to include some real-world benchmarks that consist of a variety of configurations between the models. These include models in close proximity (e.g. parallel close proximity, transverse contact etc.) as well as models moving away from each other. They have also been highlighted in the video. These include:

Engine Simulation for Virtual Prototyping: We used our algorithms to perform different proximity queries on an engine model. It is composed of moving pistons and we check for collisions and distances with the engine blocks. The path of moving pistons was pre-specified.

Randomized Path Planner: We used the randomized path planner [HKL+98] to plan the path of a wrinkled torus in a cave environment. The wrinkled torus has about 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 computes a trajectory. The randomized planner probes 2,880 random positions of the torus in the configuration space, computes distance to the nearby obstacle for each of these positions and based on that information computes a collision-free path. More details about the planner are given in [HKL+98]. All planners based on randomized approaches use all three proximity queries.

Dynamics Simulation of Falling Rings: Given a chain of 10 falling rings, we used the trajectories (for each ring) computed by a dynamics simulator and different algorithms to check for collisions and distances along these trajectories. Each ring is composed of 256 polygons. We performed 45 pairwise tests between all ring pairs at each step. We report the average time for collision and distance queries using different BVs.

Falling Tori: It consists of a lumpy torus, composed of 3,240 triangles, lying horizontally in space. The path corresponds to a smaller warped torus composed of 1,332 triangles falling onto it. It pivots until it has clear passage and falls through the center of the larger torus.

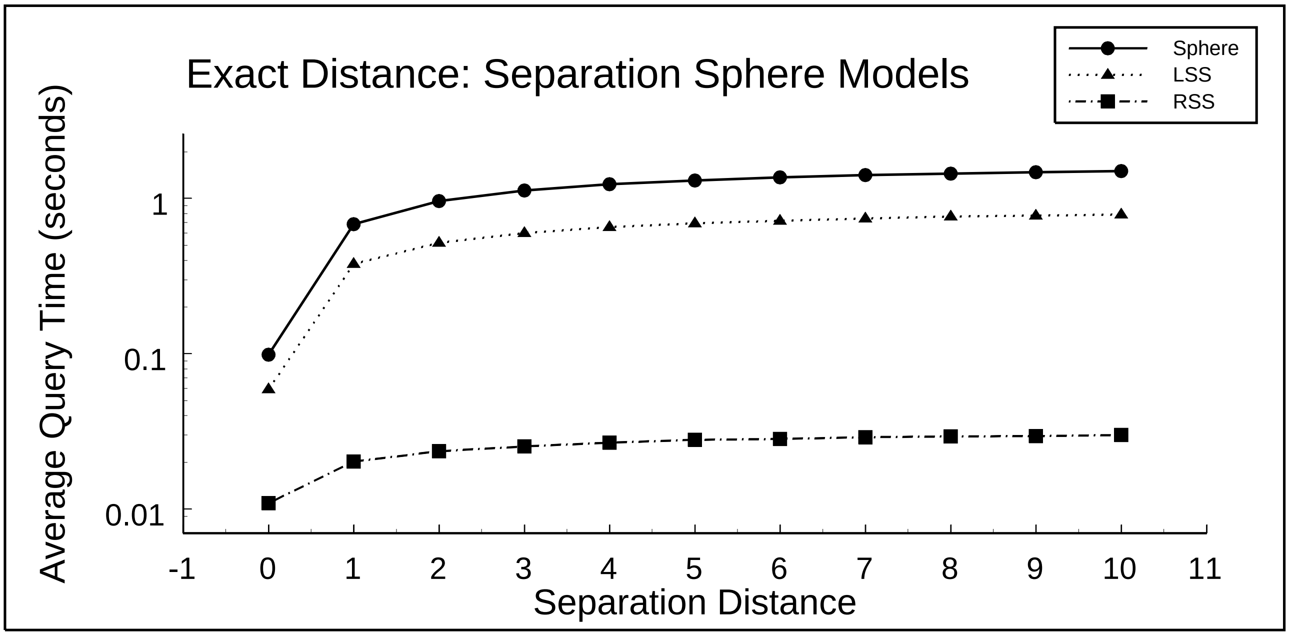

Figure 5: *This log-linear plot compares the efficiencies of exact distance queries using spheres, LSSs, and RSSs for our "separated sphere models" benchmark. It also shows that RSSs are significantly faster than the other two BV types in such situations.*

7.3 Relative Performance of Swept Sphere Volumes

We compared the relative performance of swept sphere volumes with different BV types using our "separated sphere models" benchmark, consisting of two 40K polygon sphere models of unit radius. At various separations of the models, 500 queries were made with the models given random orientations. The log-linear plot in Fig. 5 shows average query times. The plot shows that RSSs are significantly faster than spheres and LSSs. All BV types exhibit a trend of increasing expense with increasing distance, as predicted in our analysis of distance computation in Section 5, although we observe an apparent ceiling for the work for each BV type. This occurs because the benchmark involves surfaces which curve away from one another, whereas in the analysis we had made the simplifying assumption that the surfaces were parallel and unbounded.

We examined each of the BV types for approximate distance queries also using concentric sphere models of 40K triangles each (shown in Fig. 6). The outer sphere model

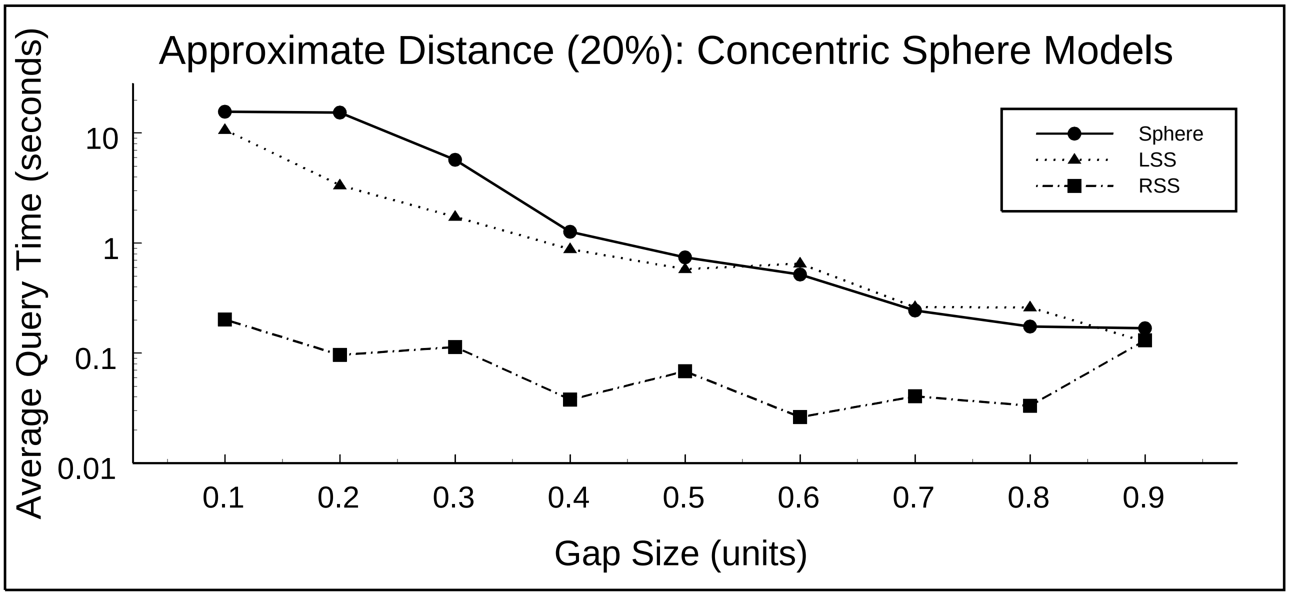

Figure 6: *This log-linear plot shows performance of each BV type for approximate distance queries between 40k polygon concentric sphere models. We observe that all BVs do more work as gap size decreases, but work for RSS increases more slowly than the others.*

was unit radius, the inner sphere model had radius ranging from 0.9 down to 0.1 (so the gap size between the spheres ranged from 0.1 to 0.9). For each gap size, we performed 60 queries, giving the models random orientation prior to each query. The average query times for each gap size are plotted for each BV type. We note that the RSSs significantly outperform the other two types when the gap size is very small, but for very large gaps the performance differences are small. We believe that when the gap is small, the problem resembles a parallel close proximity situation, and that the tighter fit of the RSSs become a significant factor in performance.

7.4 Choice of BV Type

In Tables 2, 8 and 8, we have highlighted the performance of different bounding volumes for collision detection, separation distance computation and approximate distance computation (with 10% relative error), respectively. The set of BVs include spheres, LSS, RSS, AABBs, OBBs and hybrid combination. The hybrid refers to the static selection strategy described in Section 4. We make the following interpretations from these benchmarks.

1. LSS are suitable for certain applications: In Tables 2 through 8, LSSs perform queries faster than any other BV type for the dynamics simulation benchmark. The ring models in that simulation are particularly well suited for LSSs: a section of a ring is long and thin and an LSS bounds it tightly. LSSs prune about as well as RSSs and OBBs for this scenario, but due to their faster BV test speed (lesser C_{bv}), they yield faster queries. This is an example where a simpler BV type fits the model data nearly as tightly as a more complex one. Thereby, it allows the simpler BV to yield faster queries by virtue of a faster BV query test.

2. AABBs suitable for certain applications The engine model in the virtual prototyping benchmark contains long thin triangles which are aligned with the world coordinate axes. AABBs bound these triangles tightly, their simple structure allows for fast overlap and distance tests. Thus, AABBs outperform all other types in all tests with this benchmark. In general, we can expect AABBs to perform better on models with axis-aligned components than on models with non-aligned components. Note that due to the relatively coarse tessellation of the models, the higher order convergence of OBBs and RSSs do not benefit their queries, since pruning does not occur above the leaf level.

3. Spheres unsuitable for certain applications In Tables 2-8, spheres perform poorly on the path planning application in all queries. This is because of long triangles in the stalagmites and stalagtites of the cave model which are poorly fitted by spheres. This causes the query to perform many BV overlap tests, degrading performance. Sphere BVs are not suitable for models containing long thin triangles using our algorithms.

4. RSSs most suitable for distance, OBBs not suitable The pruning power of RSSs and OBBs are comparable, so queries using these types perform a similar number of BV distance tests. However, since the OBB distance test is much more expensive than the RSS distance test, distance queries using RSSs complete much faster than distance queries using OBBs. This is seen clearly in Tables 2-8.

5. RSSs competitive for collision As mentioned before, the pruning power of RSSs is comparable to that of OBBs. The RSS overlap test is only slightly more expensive than the OBB overlap test, and consequently collision queries using RSSs tend to have comparable speed to collision queries using OBBs. This can be seen in Table 2, where RSSs outperform OBBs in half the benchmarks.

7.5 Performance of Acceleration Techniques

In Table 5, we have compared the performance of different traversal strategies on the path planning benchmark. The algorithm computes the separation distance and uses BVHs composed of RSS only. We have shown performance of five different traversal strategies for path computation and verification. For priority directed search, we used a queue size of up to 200 BV pairs. The query times are the average over a large sequence. The ideal proximity query results in the minimal BVTT for that particular configuration (as explained in Section 5). While computing a path, the planner selects the configurations randomly. As a result, there is little spatial coherence between adjacent probe sequences and we see less speedup due to triangle caching than due to priority directed search. Notice that the combined speedup due to priority directed search (PDS) and triangle caching (TC) is limited, as explained in Section 5. Overall, we are able to achieve more than an order of speedup due to these acceleration techniques on these benchmarks.

7.6 Comparison with Other Proximity Query Systems

A number of public domain systems are available for collision detection and distance computation based on BVHs. We did not directly compare the average query times for different benchmarks, shown in Tables 2-8, with the public domain systems. We wanted to ensure that we incur the same system overhead in terms of comparing the performance of different BVs. This includes same strategies for representing the data structures (e.g. transformation matrices in model space), traversal routines, fitting different bounding volumes and the overlap tests. However, we have incorporated most of the BVs used by these systems into our framework and compared their performance with swept sphere volumes. These include OBBs, used by RAPID and V-COLLIDE [GLM96, HLC+97], AABBs, used by SOLID [Sys97] and spheres, used for distance computation by Quinlan [Qui94]. However, Quinlan's implementation differs from ours in its approach. In his BVHs, several leaf node spheres may be used to tile each primitive. The BVH is then built to bound subsets of these tiling spheres, not the primitives of the model. A parameter "r_max" is used to control the maximum leaf node size and consequently the amount of leaf node tiling can greatly affect its performance. We did not incorporate tiling of primitives in our system and therefore, did not directly compare the query times. Johnson and Cohen [JC98] used OBBs and breadth-first search based traversal schemes for distance computation. Their overall performance is comparable to that of Quinlan's and they report up to two times speedup on large models. But no public domain implementation is available for their algorithm. Other candidates for comparisons include k-DOPs for collision detection [KHM+98]. However, we are not aware of any public domain implementation based on k-DOPs.

8 Conclusion and Future Work

In this paper, we have introduced a new family of BVs based on swept sphere volumes and used them to perform different proximity queries. We also provided a novel analysis of proximity queries and highlight the relationship between distance computation and collision detection. Based on this analysis, we are able to provide a unified algorithmic framework that performs all these queries efficiently. We also presented two new acceleration techniques based on priority directed search and triangle caching, which can lead to significant speedups in distance computation on some cases. We compared the performance of our algorithms with other BVHs and traversal strategies on a number of real-world benchmarks.

In terms of future work, there are many open issues. There are other candidates for core primitive shape. These include portions of a sphere or other higher order surfaces. Such BVs will provide a tighter fit to the underlying geometry. We would also like to classify scenarios or applications where different BVs (or family of BVs) work well. Based on this understanding, we need to explore hybrid hierarchies in more detail and design better static and dynamic selection schemes. We would also like to develop efficient algorithms and analyze relationships for other proximity queries such as penetration depth computation as well. Other families of BVs should to be explored for hybrid hierarchies as well. For example, the k-DOPs [KHM+98] are a good candidate. By varying the value of k, one can generate a large family of BVs.

References

[AANJ94] A.Garica-Alonso, N.Serrano, and J.Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 13(3):36–43, 1994.

[AK89] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In *An Introduction to Ray Tracing*, pages 201–262, 1989.

[Bar90] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

[BCG+96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3d. In *Proc. of Eurographics'96*, 1996.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

[Cam91] S. Cameron. Approximation hierarchies and s-bounds. In *Proceedings. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 129–137, Austin, TX, 1991.

[Cam97] S. Cameron. Enhancing gjk: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[Cha89] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 586–591, 1989.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[DK82] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. In *Proc. 9th Internat. Colloq. Automata Lang. Program.*, volume 140 of *Lecture Notes in Computer Science*, pages 154–165. Springer-Verlag, 1982.

[GJK88] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automation*, vol RA-4:193–203, 1988.

[GLGT98] A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. Real-time collision detection for haptic interaction using a 3-dof force feedback device. Technical report, Department of Computer Science, University of North Carolina, 1998. A preliminary version of this paper will appear in the Proceedings of VR'99.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[HKL+98] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.

[HKM95] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conference on Computational Geometry*, 1995.

[HKM96] M. Held, J. Klosowski, and Joseph S. B. Mitchell. Real-time collision detection for motion simulation within com plex environments. In *Proc. ACM SIGGRAPH'96 Visual Proceedings*, page 151, 1996.

[HKT92] G. Hamlin, R. Kelley, and J. Tornero. Efficient distance calculation using the spherically-extended polytope (s-tope) model. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2502–2507, 1992.

[HLC+97] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: Accelerated collision detection for vrml. In *Proc. of VRML Conference*, pages 119–125, 1997.

[Hub93] P. M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[JC98] D. Johnson and E. Cohen. A framework for efficient minimum distance computation. *IEEE Conference on Robotics and Automation*, pages 3678–3683, 1998.

[KGS98] D. Kim, L. Guibas, and S. Shin. Fast collision detection among multiple moving spheres. *IEEE Trans. on Visualization and Computer Graphics*, 4(3):230–243, 1998.

[KHM+98] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Visualization and Computer Graphics*, 4(1):21–37, 1998.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Proc. of Third International Workshop on Algorithmic Foundations of Robotics*, pages 122–136, 1998.

[LC91] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[Mir98] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yield polyhedral modeling results. In *Proc. of ACM Siggraph*, pages 115–124, 1990.

[OvdS96] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21:629–656, 1996.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[Sea93] J. Snyder and et. al. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proceedings of ACM Siggraph*, pages 321–334, 1993.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[Sys97] SOLID Interference Detection System. http://www.win.tue.nl/cs/tt/gino/solid/, 1997.

[Wel91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). Technical Report B 91-09, Fachbereich Mathematik, Freie Universitat, Berlin, 1991.

[WG91] W.Bouma and G.Vanecek. Collision detection and analysis in a physically based simulation. *Proceedings Eurographics workshop on animation and simulation*, pages 191–203, 1991.

[XZ96] J. Xiao and L. Zhang. Towards obtaining all possible contacts — growing a polyhedron by its location uncertainty. *IEEE Trans. on Robotics and Automation*, 12(4):553–565, August 1996.

[ZS99] Y. Zhou and S. Suri. Analysis of a bounding box heuristic for object intersection. In *Proceedings of SODA'99*, 1999.

BV Type	Virtual Prototyping			Falling Tori			Path Planning			Dynamic Simulation		
	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)
Sphere	4.92	1.44	173.92	6.15	0.70	7.2947	70.5	30.9	449.06	23.1	8.68	119.43
AABB	0.61	0.05	19.200	5.64	0.38	9.4972	7.22	1.45	40.328	15.6	2.46	70.819
OBB	0.39	.002	31.883	0.99	0.01	4.9359	1.92	0.20	25.974	2.10	.082	20.604
LSS	1.89	.208	80.504	3.20	0.20	7.3875	4.20	1.02	34.097	1.97	.253	10.674
RSS	0.37	.002	36.650	0.75	0.01	4.6245	1.85	0.21	29.965	1.42	.081	18.094
Hybrid	0.43	.002	31.024	0.92	0.01	4.7934	1.97	0.21	29.134	1.55	.085	17.795

Table 2: *Performance of Collision Detection Algorithms based on different BV types.*

BV Type	Virtual Prototyping			Falling Tori			Path Planning			Dynamic Simulation		
	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)
Sphere	6.45	2.06	721.40	9.54	1.474	41.05	95.59	42.704	1577.869	83.49	33.67	938.063
AABB	1.58	0.31	170.75	9.25	1.00	37.764	9.56	2.11	103.69	73.9	16.5	616.34
OBB	1.76	0.12	1443.8	1.56	0.04	75.565	1.14	.042	92.443	12.2	0.91	625.36
LSS	3.01	0.49	289.46	5.36	0.51	23.947	4.1	.975	51.279	13.2	2.15	111.37
RSS	1.70	.11	213.30	1.50	0.051	9.02	1.70	.196	25.018	10.5	0.90	121.88
Hybrid	1.80	.12	220.07	1.70	0.05	9.47	1.92	.209	26.743	11.0	0.94	129.08

Table 3: *Performance of Distance Computation Algorithms based on different BV types.*

BV Type	Virtual Prototyping			Falling Tori			Path Planning			Dynamic Simulation		
	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)
Sphere	6.30	2.005	748.64	8.92	1.307	38.23	89.58	39.97	1494.2	45.7	17.4	609.01
AABB	1.47	0.28	167.27	8.58	.855	34.60	8.18	1.70	88.104	40.8	7.02	358.78
OBB	1.53	.072	1347.9	1.36	.018	66.995	0.93	0.01	78.835	5.43	.173	363.22
LSS	2.90	0.463	279.87	4.95	0.445	21.93	3.62	0.84	45.874	4.15	.506	42.603
RSS	1.47	.071	174.86	1.32	.028	7.73	1.54	0.164	23.026	3.74	.173	50.44
Hybrid	1.57	.075	180.24	1.50	.028	8.13	1.72	0.17	24.10	3.85	.179	51.724

Table 4: *Performance of Approximate Distance Computation Algorithms (10% Relative Error) based on different BV types.*

Traversal Technique	Path Planning Computation			Path Planning Verification		
	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)	N_{bv} $\times 10^6$	N_p $\times 10^6$	Av. Query Time (ms)
Depth First Search	25.93	5.514	219.55	23.54	5.279	294.95
Priority Directed Search (PDS)	1.389	0.162	20.61	0.783	0.099	9.771
Triangle Caching (TC)	11.385	2.353	101.911	0.736	0.086	9.841
PDS & TC	1.378	0.159	20.295	0.683	0.076	8.397
Ideal Distance Query	1.126	0.121	16.517	0.679	0.075	6.164

Table 5: *Comparison of different traversal schemes on the path planning benchmark. Average query times for RSS for separation distance computation are reported. Path computation involves querying 2880 random configurations for computing a path. Path verification consists of 487 steps.*