To Appear in Proc. of ACM Siggraph'96

OBBTree: A Hierarchical Structure for Rapid Interference Detection [*]

S. Gottschalk   M. C. Lin[†]   D. Manocha
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{gottscha,lin,manocha}@cs.unc.edu
http://www.cs.unc.edu/~geom/OBB/OBBT.html

Abstract: We present a data structure and an algorithm for efficient and exact interference detection amongst complex models undergoing rigid motion. The algorithm is applicable to all general polygonal models. It pre-computes a hierarchical representation of models using tight-fitting oriented bounding box trees (OBBTrees). At runtime, the algorithm traverses two such trees and tests for overlaps between oriented bounding boxes based on a separating axis theorem, which takes less than 200 operations in practice. It has been implemented and we compare its performance with other hierarchical data structures. In particular, it can robustly and accurately detect all the contacts between large complex geometries composed of hundreds of thousands of polygons at interactive rates.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling
Additional Key Words and Phrases: hierarchical data structure, collision detection, shape approximation, contacts, physically-based modeling, virtual prototyping.

[*]Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, NSF grant CCR-9625217, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048, NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization NSF Prime contract No. 8920219 and a grant from Ford Motor company.

[†]Also with U.S. Army Research Office

1 Introduction

The problems of interference detection between two or more geometric models in static and dynamic environments are fundamental in computer graphics. They are also considered important in computational geometry, solid modeling, robotics, molecular modeling, manufacturing and computer-simulated environments. Generally speaking, we are interested in very efficient and, in many cases, real-time algorithms for applications with the following characterizations:

1. **Model Complexity:** The input models are composed of many hundreds of thousands of polygons.
2. **Unstructured Representation:** The input models are represented as collections of polygons with no topology information. Such models are also known as 'polygon soups' and their boundaries may have cracks, T-joints, or may have non-manifold geometry. No robust techniques are known for cleaning such models.
3. **Close Proximity:** In the actual applications, the models can come in close proximity of each other and can have multiple contacts.
4. **Accurate Contact Determination:** The applications need to know accurate contacts between the models (up to the resolution of the models and machine precision).

Many applications, like dynamic simulation, physically-based modeling, tolerance checking for virtual prototyping, and simulation-based design of large CAD models, have all these four characterizations. Currently, fast interference detection for such applications is a major bottleneck.

Main Contribution: We present efficient algorithms for accurate interference detection for such applications. They make no assumptions about model representation or the motion. The algorithms compute a hierarchical representation using *oriented bounding boxes (OBBs)*. An OBB is a rectangular bounding box at an arbitrary orientation in 3-space. The resulting hierarchical structure is referred to as an OBBTree. The idea of using OBBs is not new and many researchers have used them extensively to speed up ray tracing and interference detection computations. Our major contributions are:

1. New efficient algorithms for hierarchical representation of large models using tight-fitting OBBs.
2. Use of a 'separating axis' theorem to check two OBBs in space (with arbitrary orientation) for overlap. Based on this theorem, we can test two OBBs for overlap in about 100 operations on average. This test is about one order of magnitude faster compared to earlier algorithms for checking overlap between boxes.

3. Comparison with other hierarchical representations based on sphere trees and *axis-aligned bounding boxes (AABBs)*. We show that for many close proximity situations, OBBs are asymptotically much faster.

4. Robust and interactive implementation and demonstration. We have applied it to compute all contacts between very complex geometries at interactive rates.

The rest of the paper is organized in the following manner: We provide a comprehensive survey of interference detection methods in Section 2. A brief overview of the algorithm is given in Section 3. We describe algorithms for efficient computation of OBBTrees in Section 4. Section 5 presents the separating-axis theorem and shows how it can be used to compute overlaps between two OBBs very efficiently. We compare its performance with hierarchical representations composed of spheres and AABBs in Section 6. Section 7 discusses the implementation and performance of the algorithms on complex models. In Section 8, we discussion possible future extensions.

2 Previous Work

Interference and collision detection problems have been extensively studied in the literature. The simplest algorithms for collision detection are based on using bounding volumes and spatial decomposition techniques in a hierarchical manner. Typical examples of bounding volumes include axis-aligned boxes (of which cubes are a special case) and spheres, and they are chosen for to the simplicity of finding collision between two such volumes. Hierarchical structures used for collision detection include cone trees, k-d trees and octrees [31], sphere trees [20, 28], R-trees and their variants [5], trees based on S-bounds [7] etc. Other spatial representations are based on BSP's [24] and its extensions to multi-space partitions [34], spatial representations based on space-time bounds or four-dimensional testing [1, 6, 8, 20] and many more. All of these hierarchical methods do very well in performing "rejection tests", whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check very large number of bounding volume pairs for potential contacts. In such cases, their performance slows down considerably and they become a major bottleneck in the simulation, as stated in [17].

In computational geometry, many theoretically efficient algorithms have been proposed for polyhedral objects. Most of them are either restricted to static environments, convex objects, or only polyhedral objects undergoing rigid motion [9]. However, their practical utility is not clear as many of them have not been implemented in practice. Other approaches are based on linear programming and computing closest pairs for convex polytopes [3, 10, 14, 21, 23, 33] and based on line-stabbing and convex differences for general polyhedral models [18, 26, 29]. Algorithms utilizing spatial and temporal coherence have been shown to be effective for large environments represented as union of convex polytopes [10, 21]. However, these algorithms and systems are restrictive in terms of application to general polygonal models with unstructured representations. Algorithms based on interval arithmetic and bounds on functions have been described in [12, 13, 19]. They are able to find all the contacts accurately. However, their practical utility is not clear at the moment. They are currently restricted to objects whose motion can be expressed as a closed form function of time, which is rarely the case in most applications. Furthermore, their performance is too slow for interactive applications.

OBBs have been extensively used to speed up ray-tracing and other interference computations [2]. In terms of application to large models, two main issues arise: how can we compute a tight-fitting OBB enclosing a model and how quickly can we test two such boxes for overlap? For polygonal models, the minimal volume enclosing bounding box can be computed in $O(n^3)$ time, where n is the number of vertices [25]. However, it is practical for only small models. Simple incremental algorithms of linear time complexity are known for computing a minimal enclosing ellipsoid for a set of points [36]. The axes of the minimal ellipsoid can be used to compute a tight-fitting OBB. However, the constant factor in front of the linear term for this algorithm is very high (almost 3×10^5) and thereby making it almost impractical to use for large models. As for ray-tracing, algorithms using structure editors [30] and modeling hierarchies [35] have been used to construct hierarchies of OBBs. However, they cannot be directly applied to compute tight-fitting OBBs for large unstructured models.

A simple algorithm for finding the overlap status of two OBBs tests all edges of one box for intersection with any of the faces of the other box, and vice-versa. Since OBBs are convex polytopes, algorithms based on linear programming [27] and closest features computation [14, 21] can be used as well. A general purpose interference detection test between OBBs and convex polyhedron is presented in [16]. Overall, efficient algorithms were not known for computing hierarchies of tight-fitting OBBs for large unstructured models, nor were efficient algorithms known for rapidly checking the overlap status of two such OBBTrees.

3 Hierarchical Methods & Cost Equation

In this section, we present a framework for evaluating hierarchical data structures for interference detection and give a brief overview of OBBTrees. The basic cost function was taken from [35], who used it for analyzing hierarchical methods for ray tracing. Given two large models and their hierarchical representation, the total cost function for interference detection can be formulated as the following cost equation:

$$T = N_v \times C_v + N_p \times C_p, \tag{1}$$

where

T: total cost function for interference detection,
N_v: number of bounding volume pair overlap tests
C_v: cost of testing a pair of bounding volumes for overlap,
N_p: is the number primitive pairs tested for interference,
C_p: cost of testing a pair of primitives for interference.

Given this cost function, various hierarchical data structures are characterized by:

Choice of Bounding Volume: The choice is governed by two conflicting constraints:

1. It should fit the original model as tightly as possible (to lower N_v and N_p).

2. Testing two such volumes for overlap should be as fast as possible (to lower C_v).

Simple primitives like spheres and AABBs do very well with respect to the second constraint. But they cannot fit some primitives like long-thin oriented polygons tightly. On the other hand, minimal ellipsoids and OBBs provide tight fits, but checking for overlap between them is relatively expensive.

Hierarchical Decomposition: Given a large model, the tree of bounding volumes may be constructed bottom-up or top-down. Furthermore, different techniques are known for decomposing or partitioning a bounding volume into two or more sub-volumes. The leaf-nodes may correspond to different primitives. For general polyhedral models, they may be represented as collection of few triangles or convex polytopes. The decomposition also affects the values of N_v and N_p in (1).

It is clear that no hierarchical representation gives the best performance all the times. Furthermore, given two models, the total cost of interference detection varies considerably with relative placement of the models. In particular, when two models are far apart, hierarchical representations based on spheres and AABBs work well in practice. However, when two models are in close proximity with multiple number of closest features, the number of pair-wise bounding volume tests, N_v increases, sometimes also leading to an increase in the number pair-wise primitive contact tests, N_p.

For a given model, N_v and N_p for OBBTreestend to be smaller as compared to those of trees using spheres or AABBs as bounding volumes. At the same time, the best known earlier algorithms for finding contact status of two OBBs were almost two orders of magnitude slower than checking two spheres or two AABBs for overlap. We present efficient algorithms for computing tight fitting OBBs given a set of polygons, for constructing a hierarchy of OBBs, and for testing two OBBs for contact. Our algorithms are able to compute tight-fitting hierarchies effectively and the overlap test between two OBBs is one order of magnitude faster than best known earlier methods. Given sufficiently large models, our interference detection algorithm based on OBBTrees much faster as compared to using sphere trees or AABBs.

4 Building an OBBTree

In this section we describe algorithms for building an OBBTree. The tree construction has two components: first is the placement of a tight fitting OBB around a collection of polygons, and second is the grouping of nested OBB's into a tree hierarchy.

We want to approximate the collection of polygons with an OBB of similar dimensions and orientation. We triangulate all polygons composed of more than three edges. The OBB computation algorithm makes use of first and second order statistics summarizing the vertex coordinates. They are the mean, μ, and the covariance matrix, $\mathbf{C}$, respectively [11]. If the vertices of the i'th triangle are the points $\mathbf{p}^i$, $\mathbf{q}^i$, and $\mathbf{r}^i$, then the mean and covariance matrix can be expressed in vector notation as:

$$\mu = \frac{1}{3n}\sum_{i=0}^{n}(\mathbf{p}^i + \mathbf{q}^i + \mathbf{r}^i),$$

$$\mathbf{C}_{jk} = \frac{1}{3n}\sum_{i=0}^{n}(\overline{\mathbf{p}}^i_j\overline{\mathbf{p}}^i_k + \overline{\mathbf{q}}^i_j\overline{\mathbf{q}}^i_k + \overline{\mathbf{r}}^i_j\overline{\mathbf{r}}^i_k), \qquad 1 \le j,k \le 3$$

where n is the number of triangles, $\overline{\mathbf{p}}^i = \mathbf{p}^i - \mu$, $\overline{\mathbf{q}}^i = \mathbf{q}^i - \mu$, and $\overline{\mathbf{r}}^i = \mathbf{r}^i - \mu$. Each of them is a 3×1 vector, e.g. $\overline{\mathbf{p}}^i = (\overline{\mathbf{p}}^i_1, \overline{\mathbf{p}}^i_2, \overline{\mathbf{p}}^i_3)^T$ and $\mathbf{C}_{jk}$ are the elements of the 3 by 3 covariance matrix.

The eigenvectors of a symmetric matrix, such as $\mathbf{C}$, are mutually orthogonal. After normalizing them, they are used as a basis. We find the extremal vertices along each axis of this basis, and size the bounding box, oriented with the basis vectors, to bound those extremal vertices. Two of the three eigenvectors of the covariance matrix are the axes of maximum and of minimum variance, so they will tend to align the box with the geometry of a tube or a flat surface patch.

The basic failing of the above approach is that vertices on the interior of the model, which ought not influence the selection of a bounding box placement, can have an arbitrary impact on the eigenvectors. For example, a small but very dense planar patch of vertices in the interior of the model can cause the bounding box to align with it.

We improve the algorithm by using the convex hull of the vertices of the triangles. The convex hull is the smallest convex set containing all the points and efficient algorithms of $O(n \lg n)$ complexity and their robust implementations are available as public domain packages [4]. This is an improvement, but still suffers from a similar sampling problem: a small but very dense collection of nearly collinear vertices on the convex hull can cause the bounding box to align with that collection.

One solution is to sample the surface of the convex hull densely, taking the mean and covariance of the sample points. The uniform sampling of the convex hull surface normalizes for triangle size and distribution.

One can sample the convex hull "infinitely densely" by integrating over the surface of each triangle, and allowing each differential patch to contribute to the covariance matrix.

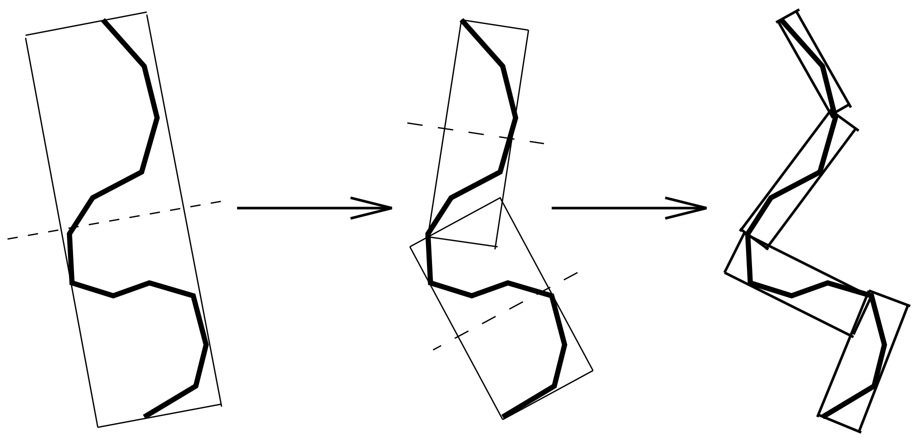

Figure 1: Building the OBBTree: recursively partition the bounded polygons and bound the resulting groups.

The resulting integral has a closed form solution. We let a point $\mathbf{x}^i$ in the i'th triangle be parameterized by s and t as in:

$$\mathbf{x}^i = \mathbf{p}^i + s(\mathbf{q}^i - \mathbf{p}^i) + t(\mathbf{r}^i - \mathbf{p}^i), \qquad s, t \in [0, 1]$$

The mean point of the convex hull is then

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{m^i}\int_0^1\int_0^{1-t}\mathbf{x}^i\ ds\ dt\right) = \frac{1}{6n}\sum_{i=1}^{n}\frac{1}{m^i}(\mathbf{p}^i + \mathbf{q}^i + \mathbf{r}^i)$$

where m^i = area of i'th triangle $= \frac{1}{2}|(\mathbf{q}^i - \mathbf{p}^i) \times (\mathbf{r}^i - \mathbf{p}^i)|$. The elements of the covariance matrix $\mathbf{C}$ have the following closed-form,

$$\begin{aligned}\mathbf{C}_{jk} &= \frac{1}{24n}\sum_{i=1}^{n} m^i[(\overline{\mathbf{p}}^i_j + \overline{\mathbf{q}}^i_j + \overline{\mathbf{r}}^i_j)(\overline{\mathbf{p}}^i_k + \overline{\mathbf{q}}^i_k + \overline{\mathbf{r}}^i_k) \\ &\quad + \overline{\mathbf{p}}^i_j\overline{\mathbf{p}}^i_k + \overline{\mathbf{q}}^i_j\overline{\mathbf{q}}^i_k + \overline{\mathbf{r}}^i_j\overline{\mathbf{r}}^i_k], \qquad 1 \le j, k \le 3\end{aligned}$$

where $\overline{\mathbf{p}}^i = \mathbf{p}^i - \mu$, $\overline{\mathbf{q}}^i = \mathbf{q}^i - \mu$, and $\overline{\mathbf{r}}^i = \mathbf{r}^i - \mu$.

Given an algorithm to compute tight-fitting OBBs around a group of polygons, we need to represent them hierarchically. Most methods for building hierarchies fall into two categories: bottom-up and top-down. Bottom-up methods begin with a bounding volume for each polygon and merge volumes into larger volumes until the tree is complete. Top-down methods begin with a group of all polygons, and recursively subdivide until all leaf nodes are indivisible. In our current implementation, we have used a simple top-down approach.

Our subdivision rule is to split the longest axis of a box with a plane orthogonal to one of its axes, partitioning the polygons according to which side of the plane their center point lies on (a 2-D analog is shown in Figure 1). The subdivision coordinate along that axis was chosen to be that of the mean point, μ, of the vertices. If the longest axis cannot not be subdivided, the second longest axis is chosen. Otherwise, the shortest one is used. If the group of polygons cannot be partitioned along any axis by this criterion, then the group is considered indivisible.

If we choose the partition coordinate based on where the median center point lies, then we obtain balanced trees. This arguably results in optimal worst-case hierarchies for collision detection. It is, however, extremely difficult to evaluate average-case behavior, as performance of collision detection algorithms is sensitive to specific scenarios, and no single algorithm performs optimally in all cases.

Given a model with n triangles, the overall time to build the tree is $O(n \lg^2 n)$ if we use convex hull, and $O(n \lg n)$ if we don't. The recursion is similar to that of quicksort. Processing fitting a box to a group of n triangles partitioning them into two subgroups takes $O(n \lg n)$ with convex hull and $O(n)$ without it. Applying the process recursively creates a tree with leaf nodes $O(\lg n)$ levels deep.

5 Fast Overlap Test for OBBs

Given OBBTrees of two objects, the interference algorithm typically spends most of its time testing pairs of OBBs for overlap. A simple algorithm for testing the overlap status for two OBB's performs 144 edge-face tests. In practice, it is an expensive test. Other algorithms based on linear programming and closest features computation exist. In this section, we present a new algorithm to test such boxes for overlap.

One trivial test for disjointness is to project the boxes onto some axis (not necessarily a coordinate axis) in space. This is an 'axial projection.' Under this projection, each box forms an interval on the axis. If the intervals don't overlap, then the axis is called a 'separating axis' for the boxes, and the boxes must then be disjoint. If the intervals do overlap, then the boxes may or may not be disjoint – further tests may be required.

How many such tests are sufficient to determine the contact status of two OBBs? We know that two disjoint convex polytopes in 3-space can always be separated by a plane which is parallel to a face of either polytope, or parallel to an edge from each polytope. A consequence of this is that two convex polytopes are disjoint iff there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope. A proof of this basic theorem is given in [15]. Each box has 3 unique face orientations, and 3 unique edge directions. This leads to 15 potential separating axes to test (3 faces from one box, 3 faces from the other box, and 9 pairwise combinations of edges). If the polytopes are disjoint, then a separating axis exists, and one of the 15 axes mentioned above will be a separating axis. If the polytopes are overlapping, then clearly no separating axis exists. So, testing the 15 given axes is a sufficient test for determining overlap status of two OBBs.

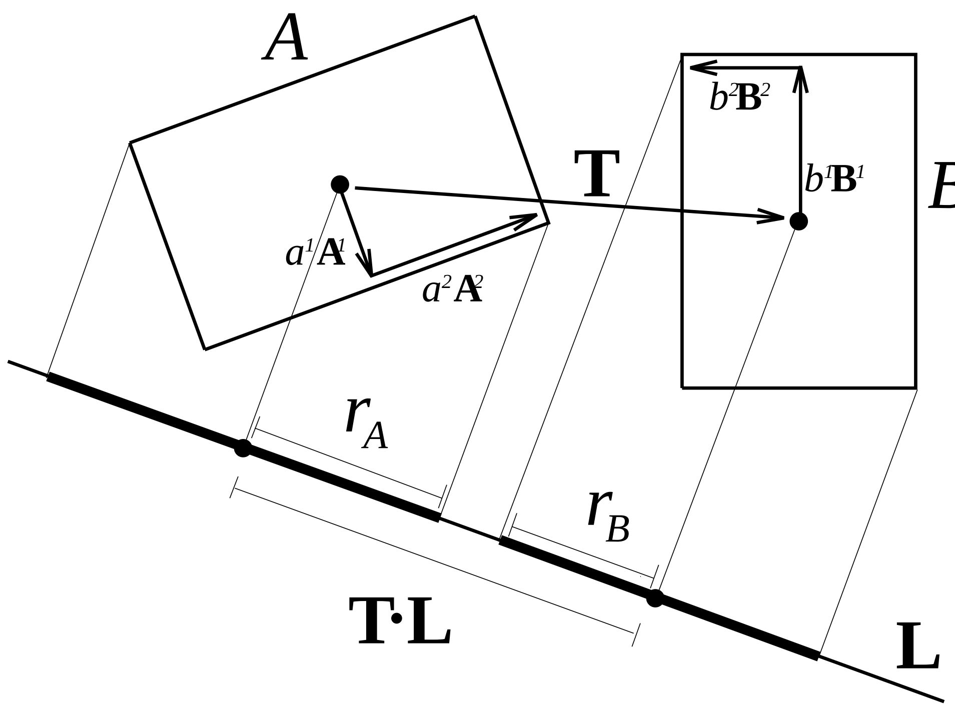

Figure 2: $\vec{L}$ is a separating axis for OBBs A and B because A and B become disjoint intervals under projection onto $\vec{L}$.

To perform the test, our strategy is to project the centers of the boxes onto the axis, and also to compute the radii of the intervals. If the distance between the box centers as projected onto the axis is greater than the sum of the radii, then the intervals (and the boxes as well) are disjoint. This is shown in 2D in Fig. 2.

We assume we are given two OBBs, A and B, with B placed relative to A by rotation $\vec{R}$ and translation $\vec{T}$. The half-dimensions (or 'radii') of A and B are a_i and b_i, where $i = 1, 2, 3$. We will denote the axes of A and B as the unit vectors $\vec{A}^i$ and $\vec{B}^i$, for $i = 1, 2, 3$. These will be referred to as the 6 box axes. Note that if we use the box axes of A as a basis, then the three columns of $\vec{R}$ are the same as the three $\vec{B}^i$ vectors.

The centers of each box projects onto the midpoint of its interval. By projecting the box radii onto the axis, and summing the length of their images, we obtain the radius of the interval. If the axis is parallel to the unit vector $\vec{L}$, then the radius of box A's interval is

$$r_A = \sum_i |a_i \vec{A}^i \cdot \vec{L}|$$

A similar expression is used for r_B.

The placement of the axis is immaterial, so we assume it passes through the center of box A. The distance between the midpoints of the intervals is $|\vec{T} \cdot \vec{L}|$. intervals. So, the intervals are disjoint iff

$$|\vec{T} \cdot \vec{L}| > \sum_i |a_i \vec{A}^i \cdot \vec{L}| + \sum_i |b_i \vec{B}^i \cdot \vec{L}|$$

This simplifies when $\vec{L}$ is a box axis or cross product of box axes. For example, consider $\vec{L} = \vec{A}^1 \times \vec{B}^2$. The second term in the first summation is

$$\begin{aligned} |a_2 \vec{A}^2 \cdot (\vec{A}^1 \times \vec{B}^2)| &= |a_2 \vec{B}^2 \cdot (\vec{A}^2 \times \vec{A}^1)| \\ &= |a_2 \vec{B}^2 \cdot \vec{A}^3| \\ &= |a_2 \vec{B}^2_3| \\ &= a_2 |\vec{R}_{32}| \end{aligned}$$

The last step is due to the fact that the columns of the rotation matrix are also the axes of the frame of B. The original term consisted of a dot product and cross product, but reduced to a multiplication and an absolute value. Some terms reduce to zero and are eliminated. After simplifying all the terms, this axis test looks like:

$$|\vec{T}_3 \vec{R}_{22} - \vec{T}_2 \vec{R}_{32}| > a_2 |\vec{R}_{32}| + a_3 |\vec{R}_{22}| + b_1 |\vec{R}_{13}| + b_3 |\vec{R}_{11}|$$

All 15 axis tests simplify in similar fashion. Among all the tests, the absolute value of each element of $\vec{R}$ is used four times, so those expressions can be computed once before beginning the axis tests. The operation tally for all 15 axis tests are shown in Table 1. If any one of the expressions is satisfied, the boxes are known to be disjoint, and the remainder of the 15 axis tests are unnecessary. This permits early exit from the series of tests, so 200 operations is the absolute *worst case*, but often much fewer are needed.

Degenerate OBBs: When an OBB bounds only a single polygon, it will have zero thickness and become a rectangle. In cases where a box extent is known to be zero, the expressions for the tests can be further simplified. The operation counts for overlap tests are given in Table 1, including when one or both boxes degenerate into a rectangle. Further reductions are possible when a box degenerates to a line segment. Nine multiplies and ten additions are eliminated for every zero thickness.

OBBs with infinite extents: Also, when one or more extents are known to be infinite, as for a fat ray or plane, certain axis tests require a straight-forward modification. For the axis test given above, if a_2 is infinite, then the inequality cannot possibly be satisfied unless $\vec{R}_{32}$ is zero, in which case the test proceeds as normal but with the $a_2|\vec{R}_{32}|$ term removed. So the test becomes,

$$\begin{aligned} \vec{R}_{32} &= 0 \ \textbf{and} \\ |\vec{T}_3 \vec{R}_{22} - \vec{T}_2 \vec{R}_{32}| &> a_3 |\vec{R}_{22}| + b_1 |\vec{R}_{13}| + b_3 |\vec{R}_{11}| \end{aligned}$$

Operation	Box-Box	Box-Rect	Rect-Rect
compare	15	15	15
add/sub	60	50	40
mult	81	72	63
abs	24	24	24

Table 1: Operation Counts for Overlap Tests

In general, we can expect that $\vec{R}_{32}$ will not be zero, and using a short-circuit **and** will cause the more expensive inequality test to be skipped.

Comparisons: We have implemented the algorithm and compared its performance with other box overlap algorithms. The latter include an efficient implementation of closest features computation between convex polytopes [14] and a fast implementation of linear programming based on Seidel's algorithm [33]. Note that the last two implementations have been optimized for general convex polytopes, but not for boxes. All these algorithms are much faster than performing 144 edge-face intersections. We report the average time for checking overlap between two OBBs in Table 2. All the timings are in microseconds, computed on a HP 735/125 .

Sep. Axis Algorithm	Closest Features	Linear Programming
$5 - 7$ us	$45 - 105$ us	$180 - 230$ us

Table 2: Performance of Box Overlap Algorithms

6 OBB's vs. other Volumes

The primary motivation for using OBBs is that, by virtue of their variable orientation, they can bound geometry more tightly than AABBTrees and sphere trees. Therefore, we reason that, all else being the same, fewer levels of an OBBTree need to be be traversed to process a collision query for objects in close proximity. In this section we present an analysis of asymptotic performance of OBBTrees versus AABBTrees and sphere trees, and an experiment which supports our analysis.

In Fig. 9(at the end), we show the different levels of hierarchies for AABBTrees and OBBTrees while approximating a torus. The number of bounding volumes in each tree at each level is the same. The ϵ for OBBTrees is much smaller as compared to ϵ for the AABBTrees.

First, we define *tightness*, *diameter*, and *aspect ratio* of a bounding volume with respect to the geometry it covers. The *tightness*, τ, of a bounding volume, B, with respect to the geometry it covers, G, is B's Hausdorff distance from G. Formally, thinking of B and G as closed point sets, this is

$$\tau = \max_{b \in B} \min_{g \in G} \mathsf{dist}(b, g)$$

The diameter, d, of a bounding volume with respect to the bounded geometry is the maximum distance among all pairs of enclosed points on the bounded geometry,

$$d = \max_{g, h \in G} \mathsf{dist}(g, h)$$

The *aspect ratio*, ρ, of a bounding volume with respect to bounded geometry is $\rho = \tau / d$.

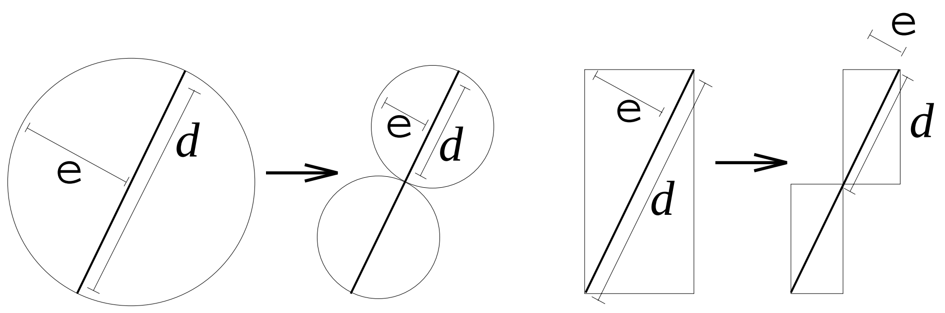

Figure 3: Aspect ratios of parent volumes are similar to those of children when bounding nearly flat geometry.

We argue that when bounded surfaces have low curvature, AABBTrees and sphere trees form fixed aspect ratio hierarchies, in the sense that the aspect ratio of a node in the hierarchy will have an aspect ratio similar to its children. This is illustrated in Fig. 3 for plane curves. If the bounded geometry is nearly flat, then the children will have shapes similar to the parents, but smaller. In Fig 3 for both spheres and AABBs, d and τ are halved as we go from parents to children, so $\rho = d/\tau$ is approximately the same for both parent and child. For fixed aspect ratio hierarchies, τ has linear dependence on d.

Note that the aspect ratio for AABBs is very dependent on the specific orientation of the bounded geometry – if the geometry is conveniently aligned, the aspect ratio can be close to 0, whereas if it is inconveniently aligned, ρ can be close to 1. But whatever the value, an AABB enclosing nearly flat geometry will have approximately the same ρ as its children.

Since an OBB aligns itself with the geometry, the aspect ratio of an OBB does not depend on the geometry's orientation in model space. Rather, it depends more on the local curvature of the geometry. For the sake of analysis, we are assuming nearly flat geometry. Suppose the bounded geometry has low constant curvature, as on the surface of

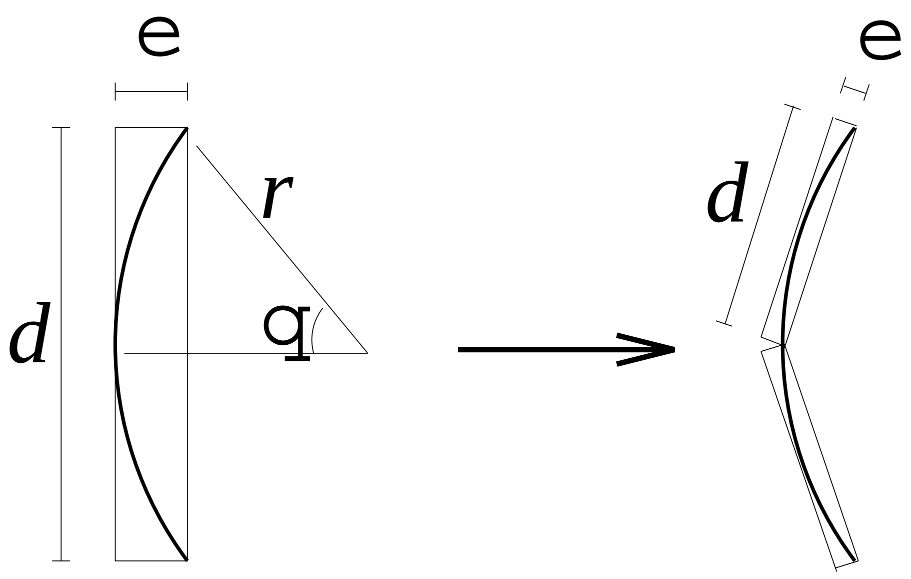

Figure 4: OBBs: Aspect ratio of children are half that of parent when bounding surfaces of low constant curvature when bounding nearly flat geometry.

a large sphere. In Fig. 4 we show a plane curve of fixed radius of curvature r and bounded by an OBB. We have $d = 2r \sin\theta$, and $\tau = r - r\cos\theta$. Using the small angle approximation and eliminating θ, we obtain $\tau = d^2/8r$. So τ has quadratic dependence on d. When d is halved, τ is quartered, and the aspect ratio is halved.

We conclude that when bounding low curvature surfaces, AABBTrees and spheres trees have τ with linear dependence on d, whereas OBBTrees have τ with quadratic dependence on d. We have illustrated this for plane curves in the figures, but the relationships hold for surfaces in three space as well.

Suppose we use N same-sized bounding volumes to cover a surface patch with area A and require each volume to cover $O(A/N)$ surface area (for simplicity we are ignoring packing inefficiencies). Therefore, for these volumes, $d = O(\sqrt{A/N})$. For AABBs and spheres, τ depends linearly on d, so $\tau = O(\sqrt{A/N})$. For OBBs, quadratic dependence on d gives us OBBs, $\tau = O(A/N)$. So, to cover a surface patch with volumes to a given tightness, if OBBs require $O(m)$ bounding volumes, AABBs and spheres would require $O(m^2)$ bounding volumes.

Most contact scenarios do not require traversing both trees to all nodes of a given depth, but this does happen when two surfaces come into *parallel close proximity* to one another, in which every point on each surface is close to some point on the other surface.

This is most common in virtual prototyping and tolerance analysis applications, in which fitted machine parts are tested for mechanical consistency. Also, dynamic simulations often generate paths in which one object comes to rest against another. It should be also be noted that when two smooth, highly tessellated surfaces come into near contact with each other, the region of near contact locally resembles a parallel close proximity scenario in miniature, and, for sufficiently tessellated models, the expense of processing that region can dominate the overall collision query. So, while it may seem like a very special case, parallel close proximity is an abstract situation which deserves consideration when designing collision and evaluating collision detection algorithms.

Experiments: We performed two experiments to support our analysis. For the first, we generated two concentric spheres consisting of 32,000 triangles each. The smaller sphere had radius 1, while the larger had radius $1+\epsilon$. We performed collision queries with both OBBTrees and AABBTrees. The AABBTrees were created using the same process as for OBBTrees, except that instead of using the eigenvectors of the covariance matrix to determine the box orientations, we used the identity matrix.

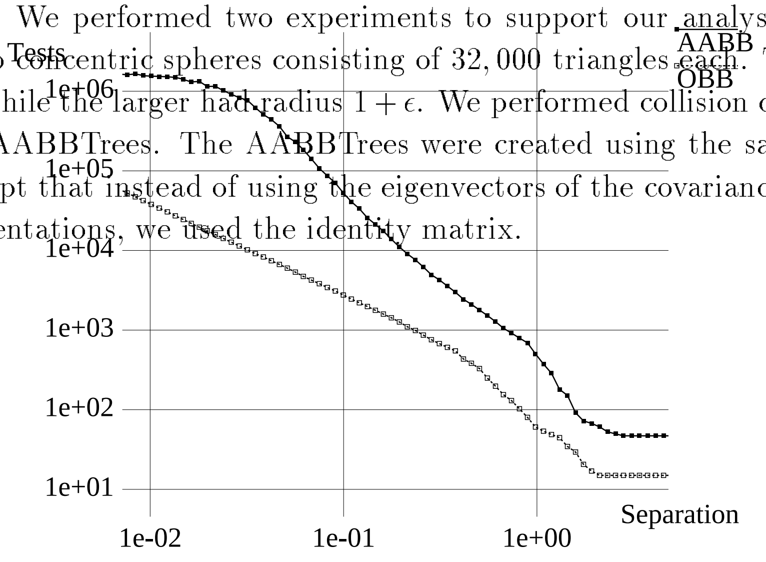

Figure 5: AABBs (upper curve) and OBBs (lower curve) for parallel close proximity (log-log plot)

The number of bounding box overlap tests required to process the collision query are shown in Fig. 5 for both tree types, and for a range of ϵ values. The graph is a log-log plot. The upper curve is for AABBTrees, and the lower, OBBTrees. The slopes of the the linear portions the upper curve and lower curves are approximately -2 and -1, as expected from the analysis. The differing slopes of these curves imply that OBBTrees require asymptotically fewer box tests as a function of ϵ than AABBTrees in our experiment.

Notice that the curve for AABBTrees levels off for the lowest values of ϵ. For sufficiently small values of ϵ, even the lowest levels of the AABBTree hierarchies are inadequate for separating the two surfaces – all nodes of both are visited, and the collision query must resort to testing the triangles. Decreasing ϵ even further cannot result in more work, because the tree does not extend further than the depth previously reached. The curve for the OBBTrees will also level off for some sufficiently small value of ϵ, which is not shown in the graph. Furthermore, since both trees are binary and therefore have the same number of nodes, the OBBTree curve will level off at the same height in the graph as the AABBTree curve.

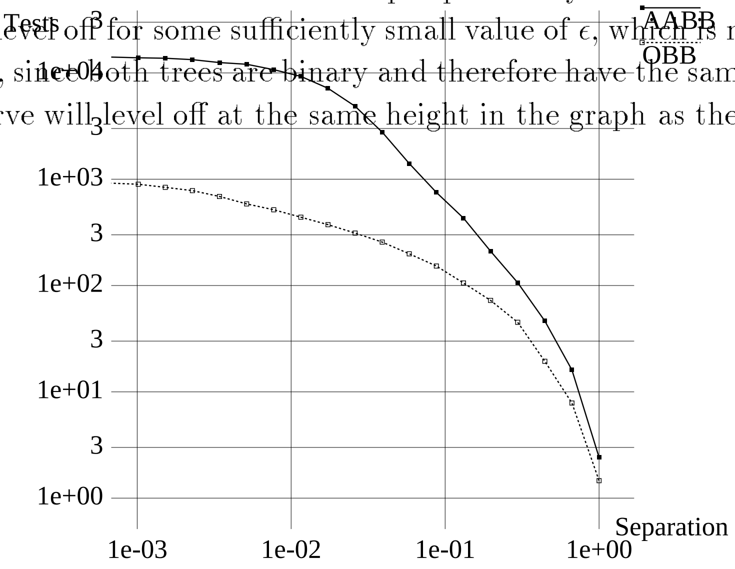

Figure 6: AABBs (upper curve) and OBBs (lower curve) for point close proximity. (log-log plot)

For the second experiment, two same-size spheres were placed next to each other, separated by a distance of ϵ. We call this scenario *point close proximity*, where two nonparallel surfaces patches come close to touching at a point. We can think of the surfaces in the neighborhood of the closest points as being in parallel close proximity – but this approximation applies only locally. We have not been able to analytically characterize the performance, so we rely instead on empirical evidence to claim that for this scenario OBBTrees require asymptotically fewer bounding box overlap tests as a function of ϵ than AABBTrees. The results are shown in Fig. 6. This is also a log-log plot, and the increasing gap between the upper and lower curves show the asymptotic difference in the number of tests as ϵ decreases. Again, we see the leveling off for small values of ϵ.

Analysis: A general analysis of the performance of collision detection algorithms which use bounding volume hierarchies is extremely difficult because performance is so situation specific. We assumed that the geometry being bounded had locally low curvature and was finely tessellated. This enabled the formulation of simple relationships between τ and d. We also assumed that the packing efficiency of bounding volumes was perfect so as to formulate the relationships between d and the area of the surface covered. We believe that the inaccuracies of these assumptions account for the deviations from theory exhibited in the graph of Fig. 5.

For surface patches with high curvature everywhere, such as a 3D fractal, we may not expect to see asymptotic performance advantages for OBBs. Similarly, a coarse tessellation of a surface will place a natural limit on the number, N, the number of volumes used to approximate the surface. For a coarse tessellation, OBB-, sphere-, and AABB-Trees may have to traverse their entire hierarchies for sufficiently close proximity scenarios, thus requiring approximately the same number of bounding volume overlap tests. Furthermore, for scenarios in which parallel close proximity does not occur, we don't expect the quadratic convergence property of OBBs to be of use, and again don't expect to see superior asymptotic performance.

7 Implementation and Performance

The software for the collision detection library was written in C++. The primary data structure for an OBB is a "box" class whose members contain a rotation matrix and translation vector, defining its placement relative to its parent, pointers to its parent and two children, the three box dimensions, and an object which holds a list of the triangles the box contains. The overall data structure for the box occupies 168 bytes. The tree formed from boxes as nodes, and the triangle list class, are the only compound data structures used.

An OBBTree of n triangles contains n leaf boxes and $n-1$ internal node boxes. In terms of memory requirements, there are approximately two boxes per triangle. The triangle itself requires 9 double precision numbers plus an integer for identification, totaling 76 bytes (based on 64-bit IEEE arithmetic). The memory requirement therefore totals 412 bytes per triangle in the model. This estimate does not include whatever overhead may exist in the dynamic memory allocation mechanism of the runtime environment. Using quaternions instead of rotation matrices (to represent box orientations), results in substantial space savings, but need 13 more operations per OBB overlap test. Single precision arithmetic can also be used to save memory.

7.1 Robustness and Accuracy

The algorithm and the implementations are applicable to all unstructured polygonal models. The polygons are permitted to be degenerate, with two or even one unique vertex, have no connectivity restrictions. The algorithm requires no adjacency information. This degree of robustness gives the system wider applicability. For example, space curves can be approximated by degenerate triangles as line segments – the system will correctly find intersections of those curves with other curves or surfaces.

The OBB overlap test is very robust as compared to other OBB overlap algorithms. It does not need to check for non-generic conditions such as parallel faces or edges; these are not special cases for the test and do not need to be handled separately. As a series of comparisons between linear combinations, the test is numerically stable: there are no divisions, square roots, or other functions to threaten domain errors or create conditioning problems. The use of an error margin, ϵ, guards against missing intersections due to arithmetic error. Its value can be set by the user.

Since the flow of control for the overlap test is simple and the number of operations required is small, the overlap test is a good candidate for microcoding or implemented in assembly. The test could also be easily implemented in hardware. Since most of the collision query time is spent in the overlap tests, any such optimization will significantly improve overall running time.

The Qhull package [4] is optionally used for computing the OBB orientation. It has been found to be quite robust. If we do use Qhull, we have to ensure that the input to Qhull spans 3 dimensions. If the input is rank deficient, our current implementation skips the use of Qhull, and uses all the triangles in the group. A more complete solution would be to project the input onto a lower dimensional space, and compute the convex hull of the projection (Qhull works on input of arbitrary specified dimension, but the input must be full rank).

There is the issue of propagation of errors as we descend the hierarchies, performing overlap tests. When we test two boxes or two triangles, their placement relative to one another is the result of a series of transformations, one for each level of each hierarchy we have traversed. We have not found errors due to the cascading of transformation matrices, but it is a theoretical source of errors we are aware of.

7.2 Performance

Our interference detection algorithm has been applied to two complex synthetic environments to demonstrate its efficiency (as highlighted in Table 3). These figures are for an SGI Reality Engine (90 MHz R8000 CPU, 512 MB).

A simple dynamics engine exercised the collision detection system. At each time step, the contact polygons were found by the collision detection algorithm, an impulse was applied to the object at each contact before advancing the clock.

Scenario	**Pipes**	**Torus**
Environ Size	143690 pgns	98000 pgns
Object Size	143690 pgns	20000 pgns
Num of Steps	4008	1298
Num of Contacts	23905	2266
Num of Box-Box Tests	1704187	1055559
Num of Tri-Tri Tests	71589	7069
Time	16.9 secs	8.9 secs
Ave. Int. Detec. Time	**4.2 msecs**	**6.9 msecs**
Ave. Time per Box Test	7.9 usecs	7.3 usecs
Ave. Contacts per Step	6.0	1.7

Table 3: Timings for simulations

In the first scenario, the pipes model was used as both the environment and the dynamic object, as shown in Fig. 8. Both object and environment contain 140,000 polygons. The object is 15 times smaller in size than the environment. We simulated a gravitational field directed toward the center of the large cube of pipes, and permitted the smaller cube to fall inward, tumbling and bouncing. Its path contained 4008 discrete positions, and required 16.9 seconds to determine all 23905 contacts along the path. This is a challenging scenario because the smaller object is entirely embedded within the larger model. The models contain long thin triangles in the straight segments of the pipes, which cannot be efficiently approximated by sphere trees, octrees, and AABBTrees, in general. It has no obvious groups or clusters, which are typically used by spatial partitioning algorithms like BSP's.

The other scenario has a complex wrinkled torus encircling a stalagmite in a dimpled, toothed landscape. Different steps from this simulation are shown in Fig. 10. The spikes in the landscape prevent large bounding boxes from touching the floor of the landscape, while the dimples provide numerous shallow concavities into which an object can enter. Likewise, the wrinkles and the twisting of the torus makes it impractical to decompose into convex polytopes, and difficult to efficiently apply bounding volumes. The wrinkled torus and the environment are also smooth enough to come into parallel close proximity, increasing the number of bounding volume overlap tests. Notice that the average number of box tests per step for the torus scenario is almost twice that of the pipes, even though the number of contacts is much lower.

We have also applied our algorithm to detect collision between a moving torpedo on a pivot model (as shown in Fig. 7). These are parts of a torpedo storage and handling room of a submarine. The torpedo model is 4780 triangles. The pivot structure has 44921 triangles. There are multiple contacts along the length of the torpedo as it rests among the rollers. A typical collision query time for the scenario shown in Fig. 7 is 100 ms on a 200MHz R4400 CPU, 2GB SGI Reality Engine.

7.3 Comparison with Other Approaches

A number of hierarchical structures are known in the literature for interference detection. Most of them are based on spheres or AABBs. They have been applied to a number of complex environments. However, there are *no standard benchmarks* available to compare different algorithms and implementations. As a result, it is non-trivial to compare two algorithms and their implementations. More recently, [18] have compared different algorithms (based on line-stabbing and AABBs) on models composed of tens of thousands of polygons. On an SGI *Indigo*^{2} Extreme, the algorithms with the best performance are able to compute all the contacts between the models in about $1/7 - 1/5$ of a second. Just based on the model complexity, we are able to handle models composed of hundreds of thousands of polygons (with multiple parallel contacts) in about $1/25 - 1/75$ of a second. We also compared our algorithm with an implementation of sphere tree based on the algorithm presented in [28]. A very preliminary comparison indicates one order of magnitude improvement. More comparisons and experiments are planned in the near future.

7.4 RAPID and benchmarks

Our implementation of our algorithms is available as a software package called RAPID (Rapid and Accurate Polygon Interference Detection). It can be obtained from:
http://www.cs.unc.edu/~geom/OBB/OBBT.html.

Most of the models shown in this paper are also available, as well as precomputed motion sequences.

Overall, we find that given two large models in close proximity, with C_v, N_v, and N_p from the cost equation (1):

- C_v for OBBTrees is one-order of magnitude slower than that for sphere trees or AABBs.
- N_v for OBBTrees is asymptotically lower than that for sphere trees or AABBs. Likewise, N_p for OBBTrees is asymptotically lower.

Thus, given sufficiently large models in sufficiently close proximity, using OBBTrees require less work to process a collision query than using AABBTrees or sphere trees.

8 Extensions and Future Work

In the previous sections, we described the algorithm for interference detection between two polygonal models undergoing rigid motion. Some of the future work includes its specialization and extension to other applications. These include ray-tracing, interference detection between curved surfaces, view frustum culling and deformable models. As far as curve and surface intersections are concerned, current approaches are based on algebraic methods, subdivision methods and interval arithmetic [32]. Algebraic methods are restricted to low degree intersections. For high degree curve intersections, algorithms based on interval arithmetic have been found to be the fastest [32]. Such algorithms compute a decomposition of the curve in terms of AABBs. It will be worthwhile to try OBBs. This would involve subdividing the curve, computing tight-fitting OBBs for each segment, and checking them for overlaps.

In terms of view frustum culling, most applications use hierarchies based on AABBs. Rather, we may enclose the object using an OBBTree and test for overlap with the view frustum. The overlap test presented in Section 5 can be easily extended to test for overlap between an OBB and a view frustum.

Libraries and Benchmarks: There is great need to develop a set of libraries and benchmarks to compare different algorithms. This would involve different models as well as scenarios.

9 Conclusion

In this paper, we have presented a hierarchical data structure for rapid and exact interference detection between polygonal models. The algorithm is general-purpose and makes no assumptions about the input model. We have presented new algorithms for efficient construction of tight-fitting OBBTrees and overlap detection between two OBBs based on a new separating axis theorem. We have compared its performance with other hierarchies of spheres and AABBs and find it asymptotically faster for close proximity situations. The algorithm has been implemented and is able to detect all contacts between complex geometries (composed of a few hundred thousand polygons) at interactive rates.

10 Acknowledgements

Thanks to Greg Angelini, Jim Boudreaux, and Ken Fast at Electric Boat for the model of torpedo storage and handling room.

References

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

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

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

[4] B. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, MN, 1993.

[5] 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.

[6] S. Cameron. Collision detection by four-dimensional intersection testing. *Proceedings of International Conference on Robotics and Automation*, pages 291–302, 1990.

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

[8] J. F. Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, 8:200–209, 1986.

[9] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34:1–27, 1987.

[10] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

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

[12] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

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

[14] 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.

[15] S. Gottschalk. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[16] N. Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In *Graphics Gems IV*, pages 74–82. Academic Press, 1994.

[17] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):pp. 299–308, 1988.

[18] 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.

[19] B. V. Herzen, A. H. Barr, and H. R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

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

[21] M.C. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[22] M.C. Lin and Dinesh Manocha. Fast interference detection between geometric models. *The Visual Computer*, 11(10):542–561, 1995.

[23] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

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

[25] J. O'Rourke. Finding minimal enclosing boxes. *Internat. J. Comput. Inform. Sci.*, 14:183–199, 1985.

[26] M. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between general solid models. In *Proc. of ACM/Siggraph Symposium on Solid Modeling*, pages 293–304, 1995.

[27] F.P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

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

[29] A. Rappoport. The extended convex differences tree (ecdt) representation for n-dimensional polyhedra. *International Journal of Computational Geometry and Applications*, 1(3):227–41, 1991.

[30] S. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Proc. of ACM Siggraph*, pages 110–116, 1980.

[31] H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addison Wesley, 1989.

[32] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

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

[34] 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.

[35] H. Weghorst, G. Hooper, and D. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, pages 52–69, 1984.

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

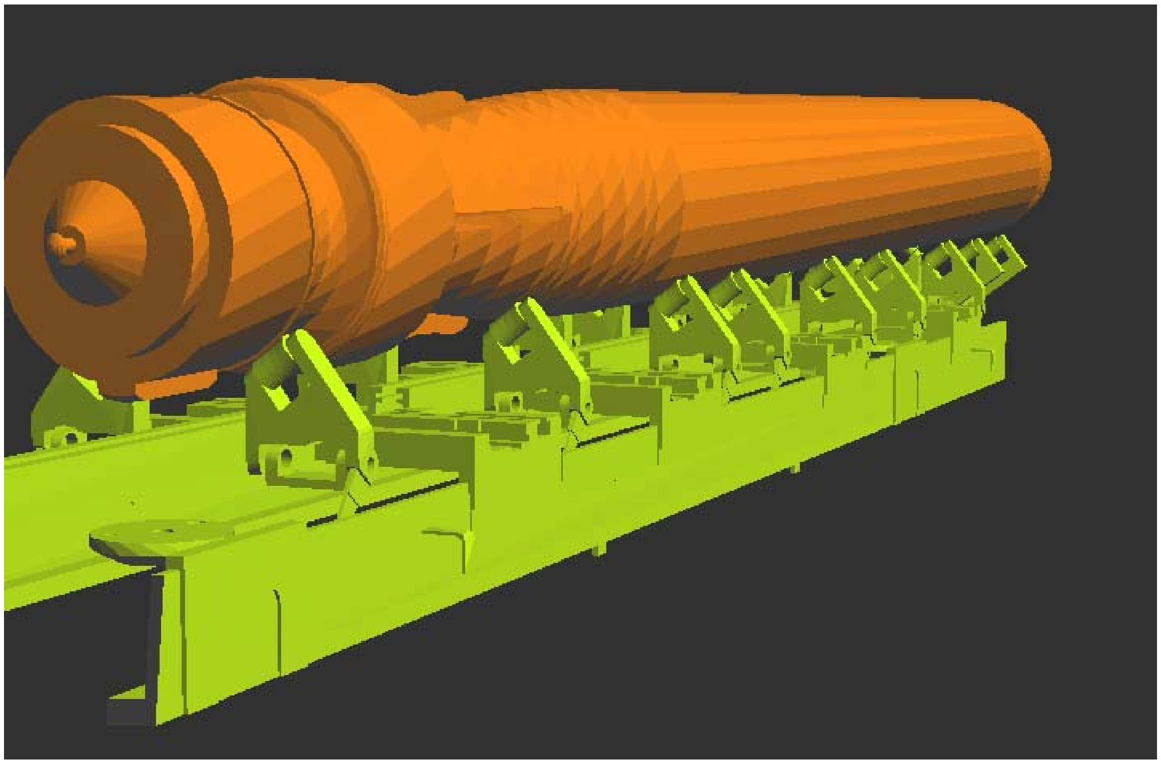

Figure 7: Interactive Interference Detection for a Torpedo (shown on the top) on a Pivot Structure – Torpedo has 4780 triangles; Pivot has 44921 triangles; Average time to perform collision query: 100 msec on SGI Reality Engine with 200MHz R4400 CPU

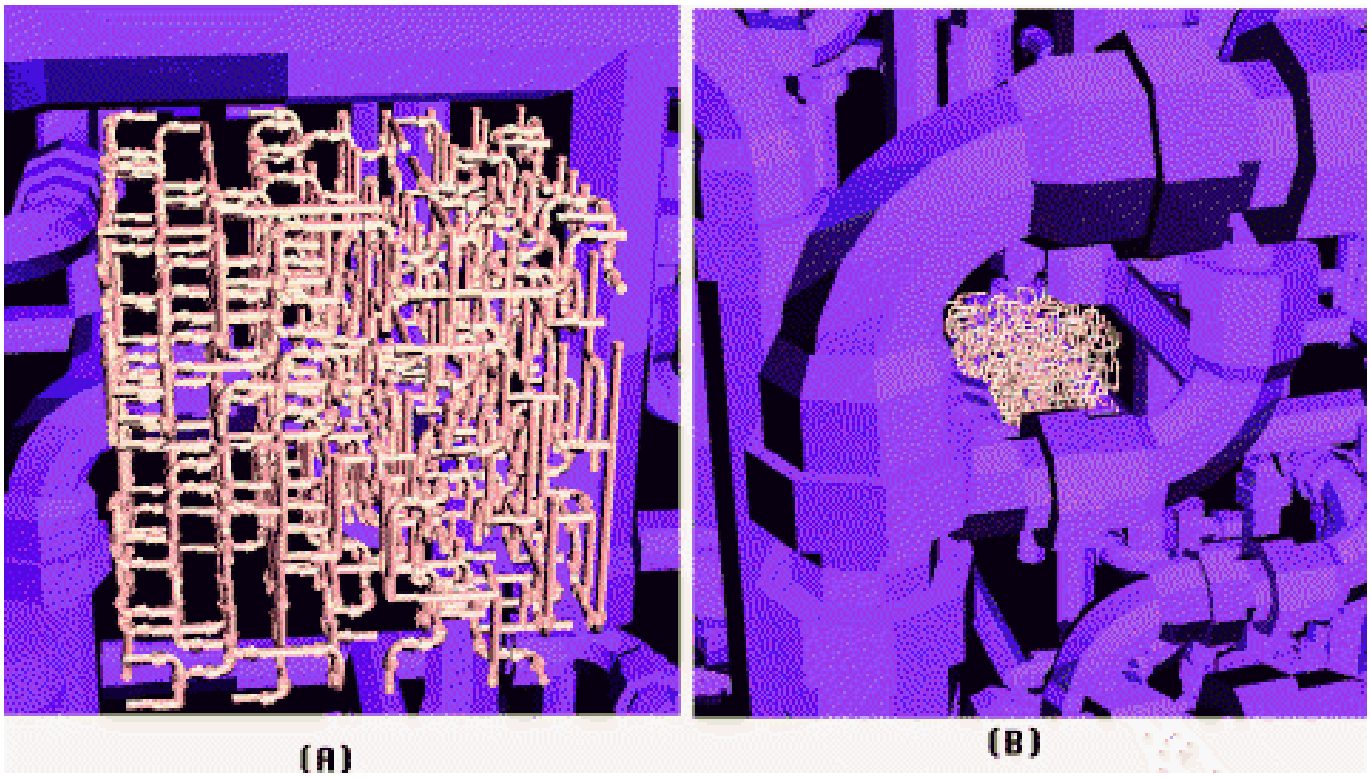

Figure 8: Interactive Interference Detection on Complex Interweaving Pipeline : $140,000$ polygons each; Average time to perform collision query: 4.2 msec on SGI Reality Engine with 90MHz R8000 CPU

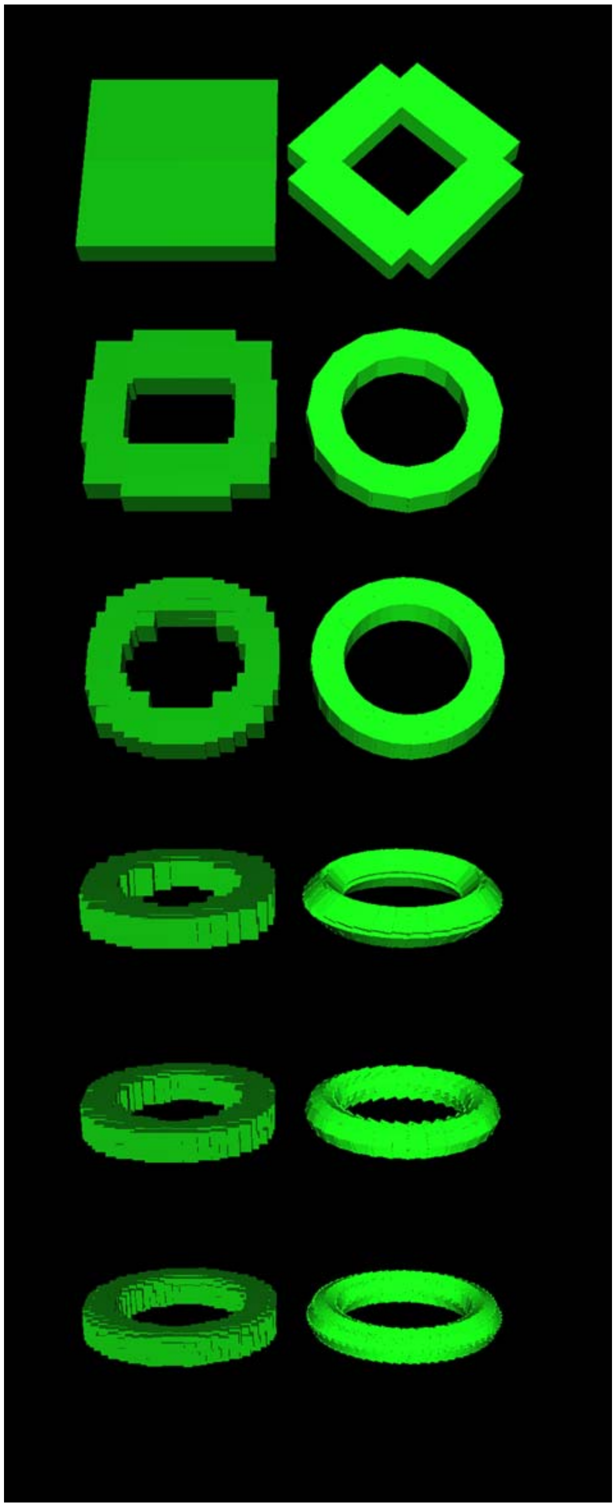

Figure 9: AABBs vs. OBBs: Approximation of a Torus – This shows OBBs converging to the shape of a torus more rapidly than AABBs.

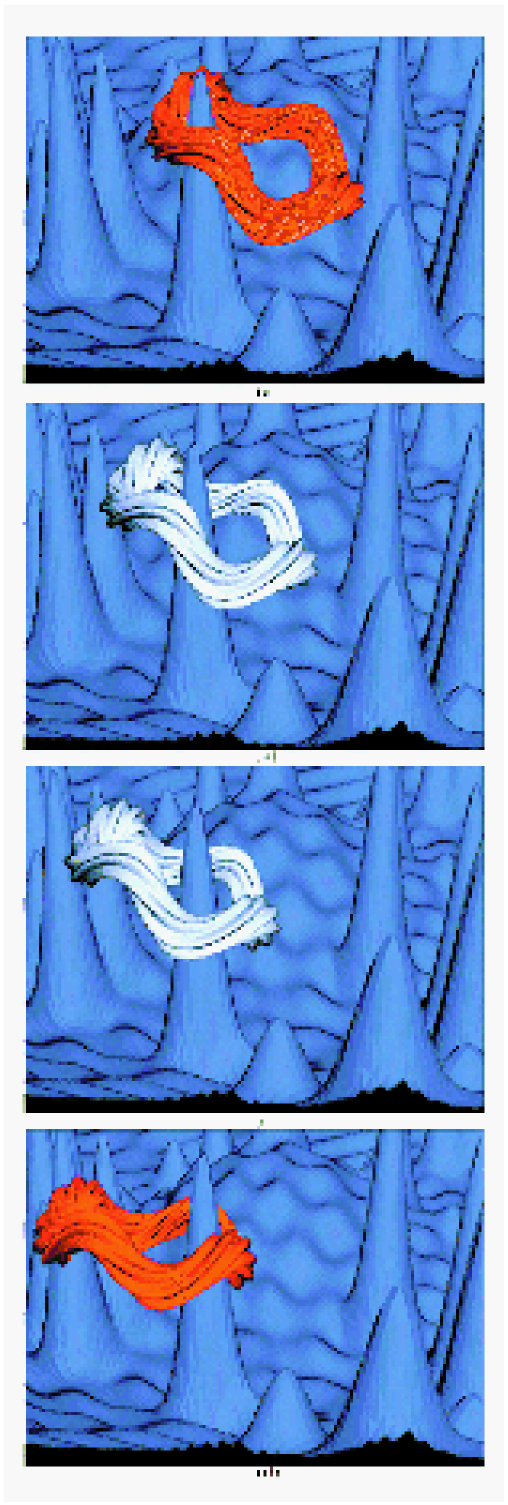

Figure 10: Interactive Interference Detection for a Complex Torus – Torus has 20000 polygons; Environment has 98000 polygons; Average time to perform collision detection: 6.9 msec on SGI Reality Engine