To Appear in Proceedings of ACM Siggraph'96

OBB-Tree: 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
{gottscha,lin,manocha}@cs.unc.edu

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 and curved models. It pre-computes a hierarchical representation of models using tight-fitting *oriented bounding box* trees. At runtime, the algorithm traverses the tree and tests for overlaps between oriented bounding boxes based on a new *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 accurately detect all the contacts between large complex geometries composed of hundreds of thousands of polygons at interactive rates, almost *one order* of magnitude faster than earlier methods.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation — Display algorithms; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling — Curve, surface, solid, and object representations.

Additional Key Words and Phrases: hierarchical approximation, model simplification, levels-of-detail generation, shape approximation, geometric modeling, offsets.

*Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048, NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization and 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, computer-graphics, 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:

- **Model Complexity:** The input models are composed of many hundred thousand of polygons.
- **Unstructured Representation:** The input models are represented as collection of polygons with no topology information. Such models are also known as *polygon soups* and their boundaries may have cracks, T-joints, or may correspond to open sets or non-manifold geometries. No robust techniques are known for cleaning such models.
- **Close Proximity:** In the actual applications, the models come in close proximity of each other and can have multiple contacts.
- **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, 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 representations or the motion. The algorithms compute a hierarchical representation using *oriented bounding boxes ($\mathcal{OBB}$'s)*. An $\mathcal{OBB}$ is a rectangular bounding box at an arbitrary orientation in 3-space. The resulting hierarchical structure is referred to as $\mathcal{OBB}$-$\mathcal{T}$ree. The idea of using $\mathcal{OBB}$'s 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 $\mathcal{OBB}$'s.
2. A new *separating axis* theorem used for checking two $\mathcal{OBB}$'s in space (with arbitrary orientation) for overlap. Based on this theorem, we can test two $\mathcal{OBB}$'s 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.

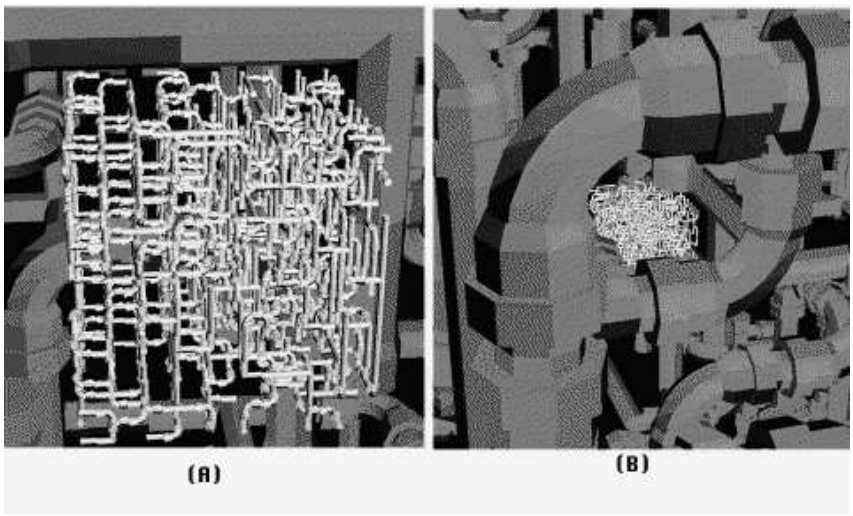

Figure 1: Interactive Interference Detection on Complex Interweaving Pipeline:140,000 polygons each (Two Different Views from the Simulation)

3. Comparison with other hierarchical representations based on sphere trees and *axis-aligned bounding boxes* ($\mathcal{AABB}$'s). We show that for many close proximity situations, $\mathcal{OBB}$'s 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 (as shown in Figure 1). Its performance is almost *one order* of magnitude faster than earlier algorithms for objects in close proximity.

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 hierarchical $\mathcal{OBB}$'s in Section 4. Section 5 presents the separating-axis theorem and shows how it can be used to compute overlaps between two $\mathcal{OBB}$'s very efficiently. We compare its performance with hierarchical representations composed of sphere trees and axis-aligned boxes in Section 6. Section 7 discusses the implementation and performance of the algorithms on complex models. In Section 8, we show how the algorithms can be specialized to ray-tracing and extended to curved geometries and deformable models.

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, spheres, and octrees, and they are chosen due to the

simplicity of finding collision between two such volumes. These include cone trees, k-d trees and octrees [Sam89], sphere trees [Hub94, Qui94], R-trees and their variants [BKSS90], trees based on S-bounds [Cam91], strip trees, and boxtrees [ZF95]. Other spatial representations are based on BSP's [NAT90] and its extensions to multi-space partitions [WG91], spatial representations based on space-time bounds or four-dimensional testing [AANJ94, Cam90, Can86, Hub94] 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 [Hah88, Hub94].

In computational geometry, many theoretically efficient algorithms have been proposed for polyhedral objects. Most of them are either restricted to static environments or convex objects or when only polyhedral object is undergoing rigid motion [CD87, Sei90]. 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 [Bar90, CLMP95, GJK88, LC91, MW88, Sei90] and based on line-stabbing and convex differences for general polyhedral models [HKM95, PML95]. Algorithms utilizing spatial and temporal coherence have been shown to be effective for large environments represented as union of convex polytopes [CLMP95]. However, these algorithms and systems are restrictive in terms of application to general polygonal models with unstructured representations. Furthermore, it is rather difficult to implement them *robustly* for general models. Algorithms based on interval arithmetic and bounds on functions have been described in [Duf92, HBZ90, ea93]. They are able to find all the contacts accurately. However these algorithms expect the motion to be expressed as a closed-form function of time, which restricts the input domain. Furthermore, their performance is slow for interactive applications.

$\mathcal{OBB}$'s have been extensively used to speed up ray-tracing and other interference computations [AK89]. In terms of application to large models, two main issues arise: how can we compute a tight-fitting $\mathcal{OBB}$ enclosing a model and how quickly can we test two such boxes for overlap? For polygonal models, the minimal enclosing bounding box is unique and can be computed in $O(n^3)$ time, where n is the number of vertices [O'R85]. However, it is practical for small models only. Simple incremental algorithms of linear time complexity are known for computing a minimal enclosing ellipsoid for a set of points [Wel91]. The axes of the minimal ellipsoid can be used to compute a tight-fitting $\mathcal{OBB}$. 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 [RW80], modeling hierarchies [WHG84] and incremental techniques which add one primitive at a time [GS87] have been used to

construct hierarchies of $\mathcal{OBB}$'s. However, they cannot be directly applied to compute tight-fitting $\mathcal{OBB}$'s for large unstructured models. A simple algorithm for testing contact between $\mathcal{OBB}$'s is based on testing all edge-face combinations to test possible intersection. Since $\mathcal{OBB}$'s are convex polytopes, algorithms based on linear programming [PS85] and closest features computation [CLMP95, GJK88] can be used as well. Overall, no good and efficient algorithms were known to compute tight-fitting hierarchical representations using $\mathcal{OBB}$'s for large unstructured models and rapidly checking them for overlap.

3 Hierarchical Methods & $\mathcal{OBB}$-Tree's

In this section, we present a framework for evaluating hierarchical data structures for interference detection and give a brief overview of $\mathcal{OBB}$-Tree's. Given two large models and their hierarchical representation, the total cost function for interference detection can be formulated as:

$$T = \mathcal{N} \times \mathcal{B} + \mathcal{P} \times \mathcal{C}, \tag{1}$$

where

T: total cost function for interference detection,
$\mathcal{N}$: number of bounding volume pair overlap tests
$\mathcal{B}$: cost of testing a pair of bounding volumes for overlap,
$\mathcal{P}$ is the number primitive pairs tested for interference,
$\mathcal{C}$: cost of testing a pair of primitives for interference.

A similar cost function was used for analyzing hierarchical methods for ray-tracing in [WHG84]. 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 $\mathcal{N}$ and $\mathcal{P}$).

2. Testing two such volumes for overlap should be as fast as possible (to lower $\mathcal{B}$).

Simple primitives like sphere and $\mathcal{AABB}$'s 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 $\mathcal{OBB}$'s 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 $\mathcal{N}$ and $\mathcal{P}$ 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 the proximity and relative orientation of the models. In particular, when two models are far apart, hierarchical representations based on spheres and $\mathcal{AABB}$'s work well in practice. However, when two models are in *close proximity* with multiple number of closest features, the resulting algorithms perform very large number of pair-wise tests, increasing $\mathcal{P}$ and $\mathcal{N}$ considerably in (1).

For a given model, $\mathcal{P}$ and $\mathcal{N}$ for $\mathcal{OBB}$ trees are much smaller as compared to those of trees using spheres or $\mathcal{AABB}$'s as primitives. At the same time, the best known earlier algorithms for checking two $\mathcal{OBB}$'s for overlaps were almost *two orders* of magnitude slower than checking two spheres or two $\mathcal{AABB}$'s for overlap. We present efficient algorithms for computing tight fitting $\mathcal{OBB}$'s given a set of polygons, for constructing a hierarchy of $\mathcal{OBB}$'s, and for testing two $\mathcal{OBB}$'s for contact. Our algorithms are able to compute tight-fitting hierarchies efficiently and the overlap test between two $\mathcal{OBB}$'s is *one order* of magnitude faster than best known earlier methods. Given sufficiently large models, our interference detection algorithm based on $\mathcal{OBB}$-$\mathcal{T}$ree's is more than one order of magnitude faster as compared to using sphere trees or $\mathcal{AABB}$'s.

4 Building an $\mathcal{OBB}$-$\mathcal{T}$ree

In this section we describe algorithms for building an $\mathcal{OBB}$-$\mathcal{T}$ree. The tree construction has two components: first is the placement of a tight fitting $\mathcal{OBB}$ 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 $\mathcal{OBB}$ of similar dimensions and orientation. We triangulate all polygons composed of more than three edges. The $\mathcal{OBB}$ computation algorithm makes use of first and second order statistics summarizing the vertex coordinates. They are the *mean* and the *covariance matrix*, respectively [DH73]. If the vertices of the i'th triangle are the points p^i, q^i, and r^i, then the mean (μ) and covariance matrix ($\mathbf{C}$) in vector arithmetics can be expressed as:

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

where n is the number of triangles, $p'^i = p^i - \mu, q'^i = q^i - \mu$, and $r'^i = r^i - \mu$, 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.

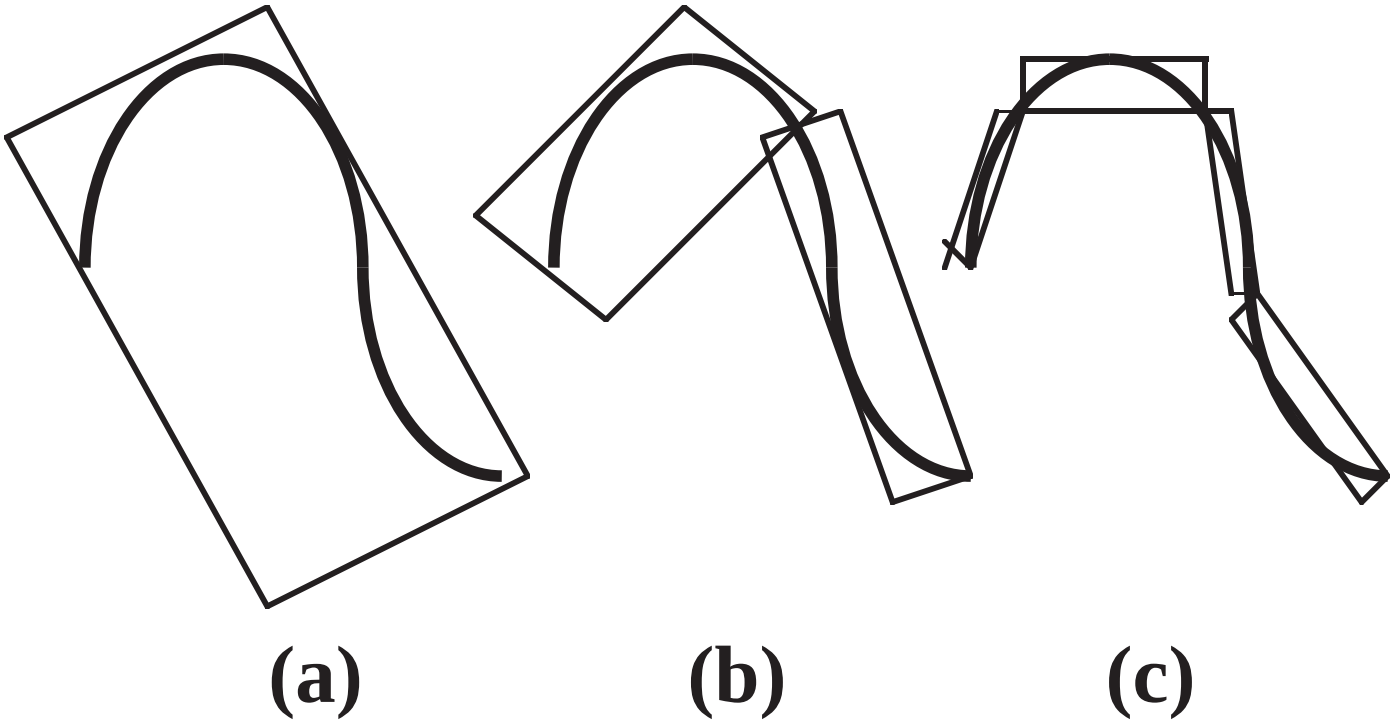

Figure 2: Three levels of an $\mathcal{OBB}$-$\mathcal{T}$ree

Using the covariance matrix of the triangle vertices is a simple way to compute a tight fitting $\mathcal{OBB}$. In many models, a number of vertices of the triangle lie in the interior. 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 [BDH93]. Given the convex hull, the algorithm *samples 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. The resulting integral has a closed form solution. We let a point x^i in the i'th triangle be parameterized by s and t as in:

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

The mean point of the convex hull is then

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

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

$$\mathbf{C}_{jk} = \frac{1}{n}\sum_{i=1}^{n} m^i[(p'^j + q'^j + r'^j)(p'^k + q'^k + r'^k) + p'^j p'^k + q'^j q'^k + r'^j r'^k],$$

where $p'^i = p^i - \mu, q'^i = q^i - \mu$, and $r'^i = r^i - \mu$.

Given an algorithm to compute tight-fitting $\mathcal{OBB}$'s 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 the top-down approach.

Our subdivision rule is to split the longest axis of a box with a plane orthogonal to it, partitioning the polygons according to which side of the plane their center point lay (as shown in Figure (2). 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 Separating Axis Theorem & Overlaps

Given $\mathcal{OBB}$-Tree's of two objects, the interference algorithm typically spends $85 - 95\%$ of the total time in testing two $\mathcal{OBB}$'s for overlap. The simplest algorithm is based on performing 144 edge-face tests. In practice, it is an expensive test. Other algorithms are

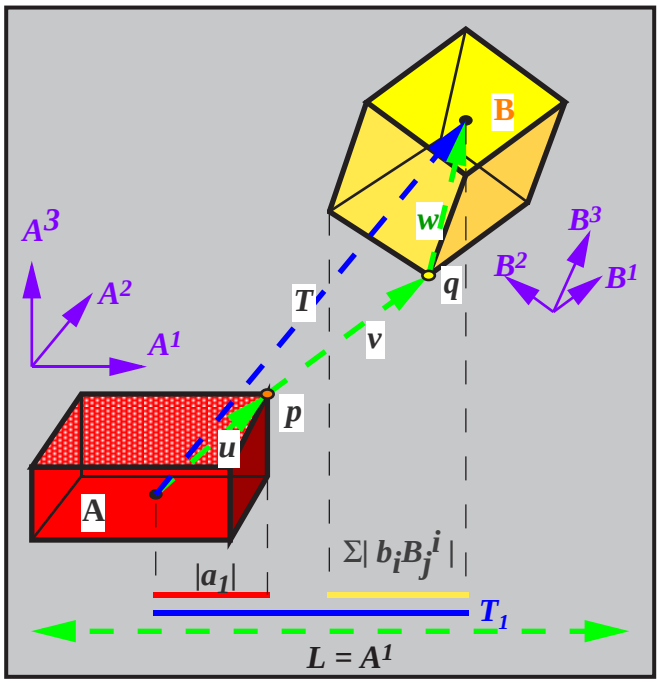

Figure 3: Separating Axis for Vertex-Vertex Case

based on linear programming and closest features computation. In this section, we present a new theorem and algorithm to test such boxes for overlap.

We are given two $\mathcal{OBB}$'s, A and B, with B placed relative to A by rotation R and translation $\vec{T}$. The dimensions of A and B are represented as 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 *box axes*. Note that if we use the box axes of A as a basis, then the columns of R are the same as the $\vec{B}^i$. Our algorithm makes use of *separating axes*. A line $\vec{L}$ is a separating axis if and only if the *axial projections* of A and B onto $\vec{L}$ are disjoint (as shown in Fig. 3). If $\vec{L}$ is a separating axis, then there exists a separating plane orthogonal to $\vec{L}$, implying that the boxes do not overlap. If no separating axis exists, then no separating plane exists, which implies for convex polytopes, such as the rectangular boxes, that they are touching.

In order for $\vec{L}$ to be a separating axis, the following condition must be met:

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

The boxes are being projected onto the axis $\vec{L}$ to form intervals. The term $|\vec{T} \cdot \vec{L}|$ is the *projected* distance separating the centers of those intervals. The first summation is the radius of the interval formed by the image of box A. Similarly for the second summation for box B. The comparison is merely between the separation distance of the centers of the intervals and the sum of their radii – a kind of 1 dimensional spherical intersection

test. If the expression is satisfied, then $\vec{L}$ is a separating axis. The interference algorithm based on separating axes chooses some lines in space and checks whether any one of them is a separating axis. In terms of application, two main questions arise: How many lines to choose and what are those lines?

The separating axis theorem asserts that 15 axial projections are sufficient to determine the contact status of two arbitrarily positioned and oriented rectangular boxes in 3-space. This formulation of the $\mathcal{OBB}$ overlap test leads to an efficient implementation requiring at most 200 arithmetic operations.

Separating Axis Theorem: If two $\mathcal{OBB}$'s are not in contact, then there exists a separating axis $\vec{L} = \vec{V} \times \vec{W}$, where $\vec{V}$ and $\vec{W}$ are distinct vectors taken from the six box axes.

Proof: Given two non-overlapping boxes, on each there is a point which is closest to the other. These "closest points" can each lie on either a vertex, edge, or face, resulting in six possible *closest feature* combinations: face-face, face-edge, face-vertex, edge-edge, edge-vertex, and vertex-vertex. We consider each of these six cases.

• **Face-face:** This is a nongeneric configuration in which the closest points lie on parallel faces. The separating axis $\vec{L}$ is orthogonal to the parallel faces, and can be formed from the cross product of any two box axes parallel to the faces.

• **Face-edge:** In this nongeneric configuration, the closest points lie on an edge and a face which are parallel. The separating axis $\vec{L}$ is orthogonal to the face, and can be formed from the cross product of the two box axes parallel to the face as well.

• **Face-vertex:** The closest points lie on a vertex and a face. Again, the separating axis $\vec{L}$ is orthogonal to the face, and can be formed from the cross product of any two box axes parallel to the face.

• **Edge-edge:** The closest points lie on edges. The separating axis $\vec{L}$ is orthogonal to both edges, and can be formed from the cross product of the box axes parallel to the edges.

• **Vertex-vertex:** The proof for this case is more complicated than the others. We will sketch the outline of the argument here. For more details, refer to [Got96]. Suppose point $\mathbf{p}$ is at a vertex on A and point $\mathbf{q}$ is at a vertex on B, and these points are the closest points between the boxes, as shown in Figure 3. Let the vector $\vec{u}$ join the center of A with $\mathbf{p}$, vector $\vec{v}$ join $\mathbf{p}$ to $\mathbf{q}$, and vector $\vec{w}$ join $\mathbf{q}$ to the center of B. Recall that $\vec{T}$ joins the center of A to the center of B. In vector notation, the diagram expresses,

$$\vec{T} = \vec{u} + \vec{v} + \vec{w}$$

which implies

$$|\vec{T} \cdot \vec{L}| = |\vec{u} \cdot \vec{L} + \vec{v} \cdot \vec{L} + \vec{w} \cdot \vec{L}| \qquad \text{for any } \vec{L}.$$

We can show that for one of the separating axis, say, $\vec{L} = \vec{A}^j$, that $\vec{u} \cdot \vec{A}^j, \vec{v} \cdot \vec{A}^j$, and $\vec{w} \cdot \vec{A}^j$

have the same sign, so that

$$|\vec{T} \cdot \vec{A}^j| = |\vec{u} \cdot \vec{A}^j| + |\vec{v} \cdot \vec{A}^j| + |\vec{w} \cdot \vec{A}^j|$$

which implies

$$|\vec{T} \cdot \vec{A}^j| > |\vec{u} \cdot \vec{A}^j| + |\vec{w} \cdot \vec{A}^j| \qquad \text{for some } \vec{A}^j$$

• **Edge-vertex:** This case is very similar to the vertex-vertex case, except that $\vec{v}$ will be orthogonal to the edge feature.

We have shown that if two boxes are disjoint, then for each of the six closest feature pair combinations, one of the 15 different axes derived from the cross products of the box axes will be a separating axis. The contra-positive statement says that if none of the 15 axes is a separating axis, then the boxes overlap. So, testing the 15 axes is sufficient to determine the contact status of the boxes.

The test of an axis is very simple. The expression originally given was

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

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

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

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

$$|T_3 R_{22} - T_2 R_{32}| > a_2 |R_{32}| + a_3 |R_{22}| + b_1 |R_{13}| + b_3 |R_{11}|$$

All 15 axis tests simplify in similar fashion. The absolute value of the elements of R are repeated, so those expressions can be computed once before beginning the axis tests. The total number of operation count for all 15 axis tests is less than 200, counting absolute value and comparisons. 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 worst case bound, but often much fewer are needed.

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 [GJK88] and a fast implementation of linear programming based on Seidel's algorithm [Sei90]. Note that the last two implementations have

been optimized for general convex polytopes. All these algorithms are much better than performing 144 edge-face intersections. We report the average time for checking overlap between two $\mathcal{OBB}$'s in Table 1. 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 1: Performance of Box Overlap Algorithms

6 $\mathcal{OBB}$'s vs. other Volumes

In this section, we analyze the performance of $\mathcal{OBB}$-*T*ree's with other hierarchical structures based on sphere trees and $\mathcal{AABB}$'s. We make use of *aspect ratios* of these bounding volumes. In particular, hierarchies based on spheres (shown in Figure 4(c)) or $\mathcal{AABB}$'s(shown in Figure 4(d)) form *fixed aspect-ratio* hierarchies.

We define the *thickness*, ϵ, of a bounding volume as the maximum distance of the bounded surface from any point in the volume, and the *diameter*, d, of a bounding volume as the maximum distance between two points on the surface within the volume. We define the *aspect-ratio* as ϵ/d. The aspect-ratio of a sphere is the same as that of it's children. Likewise for $\mathcal{AABB}$.

Since the $\mathcal{OBB}$ aligns itself with the surface geometry (as shown in Figure 4(b)), it depends more on the curvature of the surface, and not on the orientation. To simplify the analysis, we initially assume the curvature of the surface patch is constant, like that of a sphere. In Figure 4(a), we have $d = 2r\sin\theta$, and $\epsilon = r - r\cos\theta$. Using small angle assumption and eliminating θ, we obtain $\epsilon = d^2/8r$. So ϵ has *quadratic dependence* on d. When d is halved, ϵ is quartered, and the aspect-ratio is *halved*.

If we use N same-sized bounding volumes to cover a patch of area A and require each volume to cover $O(A/N)$ surface area. Therefore, $d = O(\sqrt{A/N})$. However, for $\mathcal{AABB}$'s and spheres, ϵ depends *linearly* on the d's, so $\epsilon = O(\sqrt{A/N})$. For $\mathcal{OBB}$'s, $\epsilon = O(A/N)$. In each case, ϵ is a measure of how *tightly* the tiling of bounding volumes covers the patch.

Such a tiling of two parallel surface patches offset by a gap of 2ϵ can be used to determine their contact status. This scenario is *parallel close proximity*, very common in virtual prototyping, dynamic simulation and tolerance analysis applications, where the two objects can have multiple close contacts (as shown for two interweaving pipelines in Figure 1). If this gap between the surface patches is halved, then the number, N, of spheres or $\mathcal{AABB}$'s

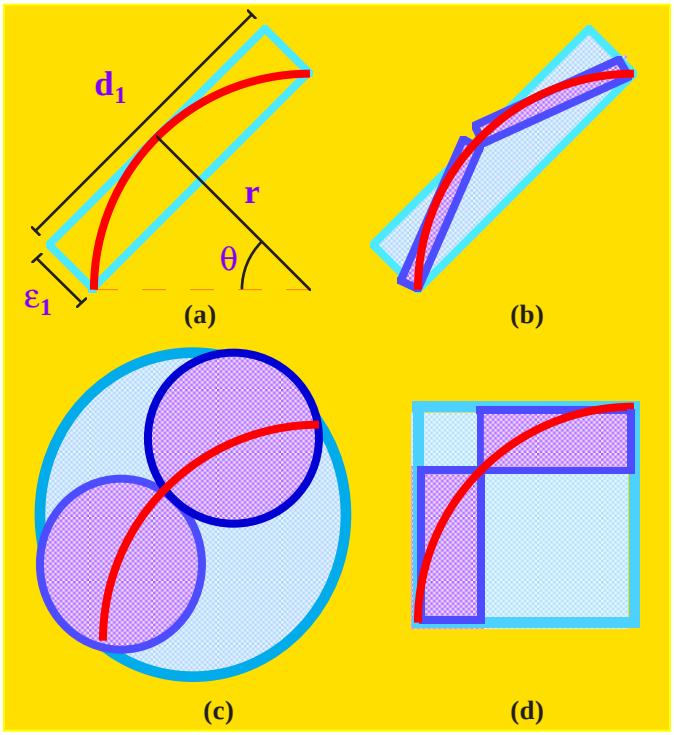

Figure 4: Convergence of Various Hierarchies

required to confirm that they are disjoint increases by a factor of 4. However, for $\mathcal{OBB}$'s, N doubles. Thus, we claim that $\mathcal{OBB}$'s require $O(\sqrt{N})$ work (in terms of number of bounding volume overlaps), whereas spheres and $\mathcal{AABB}$'s require $O(N)$.

We have verified this analysis by performing some simulations. One simulation consists of two concentric spheres, whose radii differ by some ϵ. We compared the performance of $\mathcal{AABB}$-Trees against $\mathcal{OBB}$-Tree's. We looked at how many bounding volume overlap tests each hierarchy required as ϵ varied. The results are shown as a graph in Figure 5. It is a *log-log* plot, showing that $\mathcal{OBB}$-Tree'srequire asymptotically fewer tests than $\mathcal{AABB}$-Trees as ϵ decreases.

A more common scenario is *point close proximity*, where two nonparallel surface 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 $\mathcal{OBB}$-Tree's have superior asymptotic performance than $\mathcal{AABB}$-Trees. We placed two same-sized spheres next to one another, separated by some ϵ, and examined how many tests each hierarchy required to determine disjoint contact status as ϵ was varied. The results are shown in Figure 6.

On surface patches with high curvature everywhere, such as a 3D fractal, it is possible that we may lose some or all of the performance advantages. Similarly, a very coarse

polygonalization of a surface will place a natural limit on N, the number of volumes used to approximate the surface, and $\mathcal{OBB}$-, sphere-, and $\mathcal{AABB}$-trees will have to traverse their entire hierarchies, all requiring the same order of bounding volume overlap tests. In Figure 7, we show the different level of hierarchies for $\mathcal{AABB}$-tree's and $\mathcal{OBB}$-Tree's while approximating a torus. The number of bounding volumes in each tree at each level is the same. Even for a symmetric object like a torus, the ϵ for $\mathcal{OBB}$-Tree's is much smaller as compared to ϵ for the $\mathcal{AABB}$-tree's.

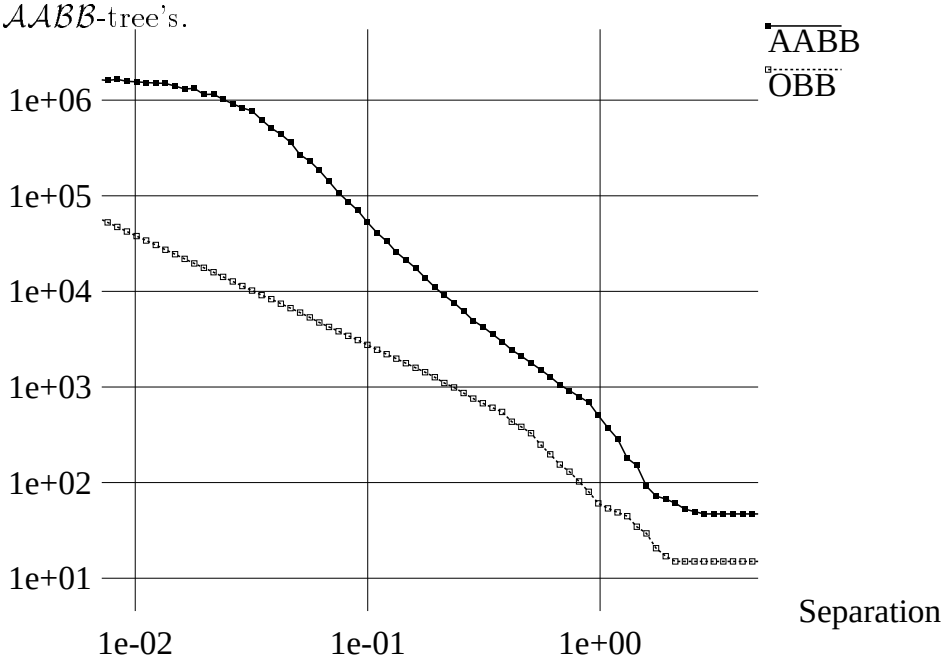

Figure 5: $\mathcal{OBB}$'s vs. $\mathcal{AABB}$'s: Parallel Proximity

7 Implementation and Performance

The software for the collision detection library was written in C++. The *primary data structure* for an $\mathcal{OBB}$ 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

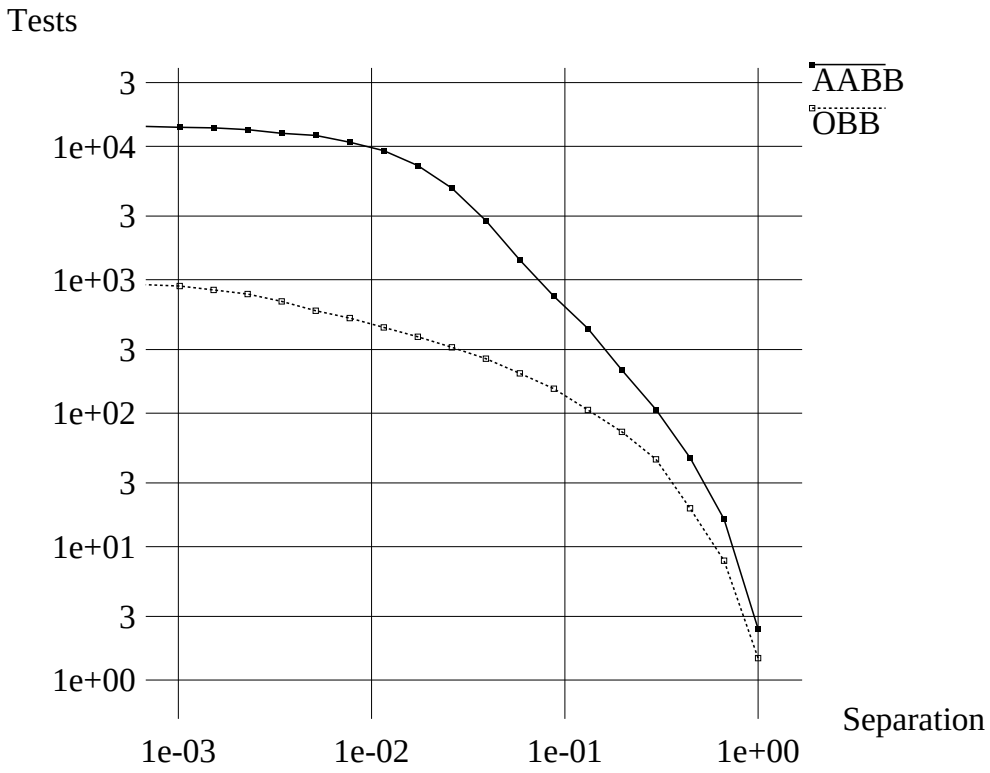

Figure 6: $\mathcal{OBB}$'s vs. $\mathcal{AABB}$'s: Point Proximity

from boxes as nodes, and the triangle list class, are the only compound data structures used.

An $\mathcal{OBB}$-$\mathcal{T}$ree 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, totalling 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), we can use 80 bytes fewer per triangle, but need 13 more operations per $\mathcal{OBB}$ overlap test.

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 $\mathcal{OBB}$ overlap test is *very robust* as compared to other $\mathcal{OBB}$ 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.

The Qhull package [BDH93] is optionally used for computing the $\mathcal{OBB}$ 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 2). 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.

In the first scenario, the pipes model was used as both the environment and the dynamic object, as shown in Figure 1. 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

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 2: Timings for simulations

efficiently approximated by sphere-trees, octrees, and $\mathcal{AABB}$-Trees, 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 Figure 8 (and the video). 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.

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 $\mathcal{AABB}$'s. 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, [HKM95] have compared different algorithms (based on line-stabbing and $\mathcal{AABB}$'s) 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 [Qui94]. A very preliminary comparison indicates one order of magnitude improvement.

Overall, we find that given two large models in close proximity:

- $\mathcal{B}$ for $\mathcal{OBB}$-Tree's is one-order of magnitude slower than that for sphere-trees or $\mathcal{AABB}$'s.
- $\mathcal{N}$ for $\mathcal{OBB}$-Tree's is asymptotically lower than that for sphere trees or $\mathcal{AABB}$'s. Likewise, $\mathcal{P}$ for $\mathcal{OBB}$-Tree's is asymptotically lower.

Thus, given *sufficiently* large models, our interference detection algorithm based on $\mathcal{OBB}$-Tree's is more than **one order** of magnitude faster as compared to using sphere trees or $\mathcal{AABB}$'s.

8 Extensions and Future Work

In the previous sections, we described the algorithm for interference detection between two polygonal models undergoing rigid motion. In this section, we discuss its specialization and extension to other applications. These include ray-tracing, interference detection between curved surfaces and deformable models.

Ray-Tracing: An efficient algorithm for computing the intersection between a ray and an $\mathcal{OBB}$ was presented in [RW80]. It performs intersection tests with all the boundary planes. An alternative is to view the ray as an $\mathcal{OBB}$ with infinite length, and zero width and height. In this case, the Separating Axes Theorem (in Section 5) can be specialized to show that it is sufficient to test only three axes.

Curve and Surface Intersections: Computing the intersection between curves and surfaces is a fundamental problem in geometric and solid modeling [SP86]. Current approaches are based on algebraic methods, subdivision methods and interval arithmetic. 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 [SP86]. Such algorithms compute a decomposition of the curve in terms of $\mathcal{AABB}$'s. We plan to apply our algorithm based on $\mathcal{OBB}$'s to such problems. It involves subdividing the curve, computing tight-fitting $\mathcal{OBB}$'sfor each segment, and checking them for overlaps.

Deformable Models: Efficient interference detection between deformable models is a challenging problem. Current efficient algorithms are restricted to particular deformations (like quadratic deformations) or when the motion can be expressed as a closed form function of time. Our interference detection algorithm could be applied to deformable models,

but an $\mathcal{OBB}$-$\mathcal{T}$ree would ordinarily be invalidated as the model deforms. Our current implementation of building the entire $\mathcal{OBB}$-$\mathcal{T}$ree anew is not interactive for large models. One possible approach is lazy, top-down construction of $\mathcal{OBB}$-Tree'sin which only those nodes which get visited are explicitly represented and processed.

Libraries and Benchmarks: We propose to make our models, scenarios and the interference detection library publically available sometime in the future. This would facilitate the comparison of different approaches, algorithms and implementations among researchers.

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 $\mathcal{OBB}$-Tree'sand overlap detection between two $\mathcal{OBB}$'s based on a new separating axis theorem. We have compared its performance with other hierarchical trees based on spheres and $\mathcal{AABB}$'s 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

We are grateful to Ray Brynes and Ford Motor company for partially supporting this research.

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.

[BDH93] B. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, MN, 1993.

[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.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. *Proceedings of International Conference on Robotics and Automation*, pages pp. 291–302, 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.

[Can86] J. F. Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, 8:pp. 200–209, 1986.

[CD87] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34:1–27, 1987.

[CLMP95] 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.

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

[Duf92] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

[ea93] J. Snyder et. al. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proceedings of ACM Siggraph*, pages 321–334, 1993.

[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:pp. 193–203, 1988.

[Got96] S. Gottschalk. Separating axis theorem. manuscript, 1996.

[GS87] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

[Hah88] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):pp. 299–308, 1988.

[HBZ90] B. V. Herzen, A. H. Barr, and H. R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

[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.

[Hub94] P. Hubbard. *Collision Detection for Interactive Graphics Applications.* PhD thesis, Brown University, 1994.

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

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

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

[O'R85] J. O'Rourke. Finding minimal enclosing boxes. *Internat. J. Comput. Inform. Sci.*, 14:183–199, 1985.

[PML95] 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.

[PS85] F.P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

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

[RW80] S. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Proc. of ACM Siggraph*, pages 110–116, 1980.

[Sam89] H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addison Wesley, 1989.

[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.

[SP86] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

[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.

[WHG84] H. Weghorst, G. Hooper, and D. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, pages 52–69, 1984.

[ZF95] G. Zachman and W. Felger. The boxtree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *Proc. of SIVE'95*, 1995.

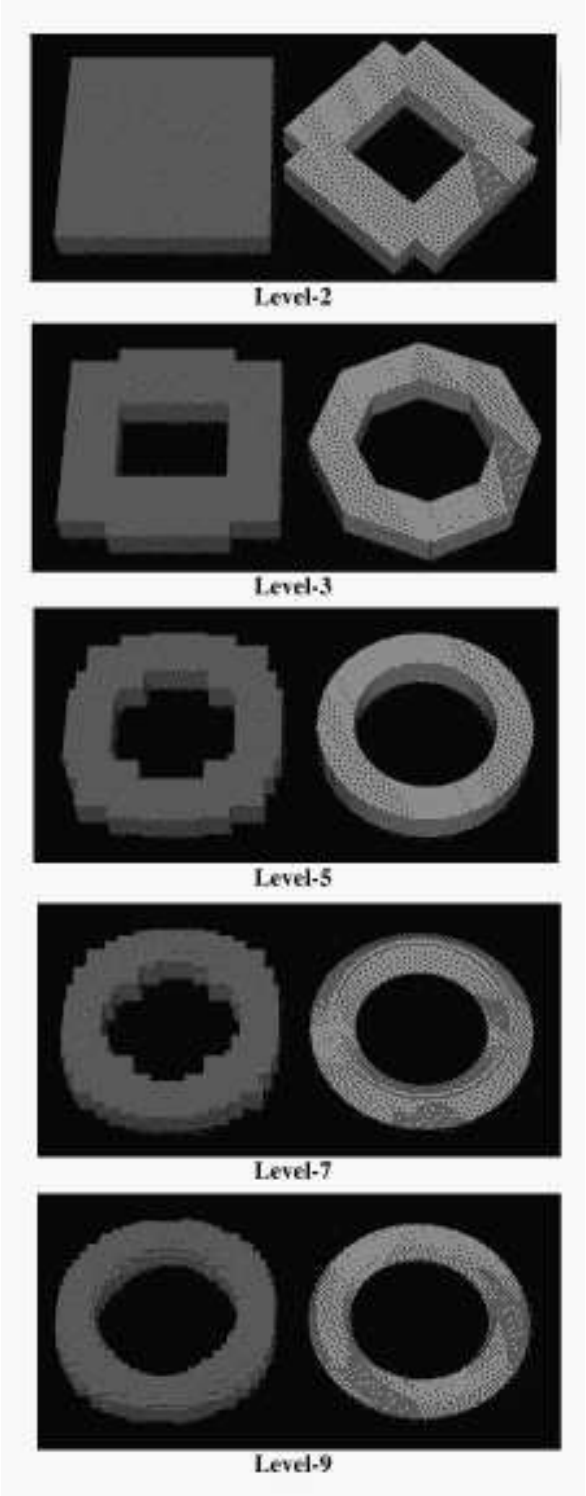

Figure 7: $\mathcal{AABB}$'s vs. $\mathcal{OBB}$'s: Approximation of a Torus – This shows $\mathcal{OBB}$'s converging to the shape of a torus more rapidly than $\mathcal{AABB}$'s.

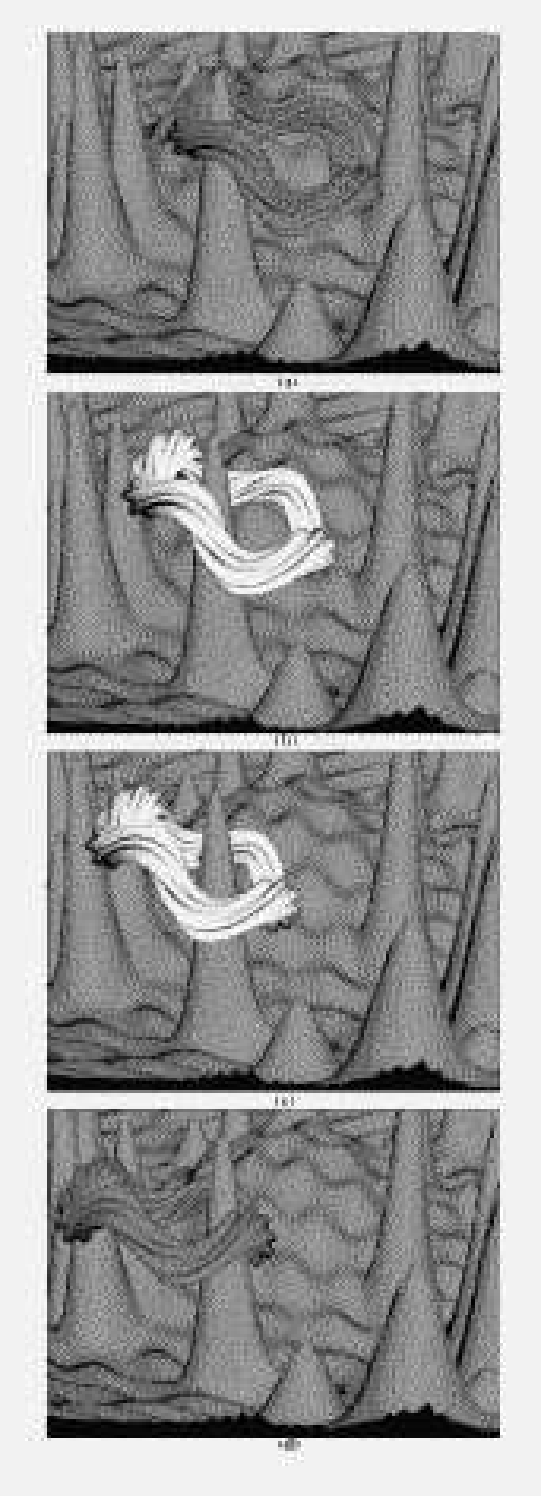

Figure 8: Interactive Interference Detection for a Complex Torus – Torus has 20000 polygons; Environment has 98000 polygons; Average time to perform collision detection: 6.9 msec on SGI Reality Engine