Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries

Article · April 1998

Source: CiteSeer

CITATIONS
28

READS
312

1 author:

Shankar Krishnan
Google Inc., Mountain View, California
107 PUBLICATIONS **2,672** CITATIONS

Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries *

Shankar Krishnan, *AT&T Research Labs, Florham Park, NJ, USA*
Amol Pattekar, *University of North Carolina, Chapel Hill, NC, USA*
Ming C. Lin, *University of North Carolina, Chapel Hill, NC, USA*
Dinesh Manocha, *University of North Carolina, Chapel Hill, NC, USA*

Hierarchical data structures have been widely used to design efficient algorithms for interference detection for robot motion planning and physically-based modeling applications. Most of the hierarchies involve use of bounding volumes which enclose the underlying geometry. These bounding volumes are used to test for interference or compute distance bounds between the underlying geometry. The efficiency of a hierarchy is directly proportional to the choice of a bounding volume. In this paper, we introduce spherical shells, a higher order bounding volume for fast proximity queries. Each shell corresponds to a portion of the volume between two concentric spheres. We present algorithms to compute tight fitting shells and fast overlap between two shells. Moreover, we show that spherical shells provide local cubic convergence *to the underlying geometry. As a result, in many cases they provide faster algorithms for interference detection and distance computation as compared to earlier methods. We also describe an implementation and compare it with other hierarchies.*

1 Introduction

The problems of interference detection and distance computation between static or dynamic objects are fundamental to robotics and computer simulated environments. Given two or more geometric models, composed of polygonal or curved boundaries, the goal is to check whether they overlap or to compute the minimal distance between them. As an integral part of robot motion planning or collision avoidance, the geometric reasoning system that detects potential contacts and determines the amount of separation between the moving object(s) and the obstacles in the workspace serves as a good indicator for fast proximity queries.

These problems have been well studied in the literature and numerous solutions have been proposed. Especially in the case of convex polytopes or objects that can be represented as union of convex polytopes, efficient algorithms have been presented in [GJK88, LC91, Cam97, BK88]. They work quite well for polygonized manipulators in a simple simulated environment. However, in different applications using motion planning techniques, including virtual prototyping [CL95], haptic rendering [RKK97] and molecular modeling [FKL+97], models composed of tens or hundred of thousands of primitives (e.g. polygons, spline patches) are very common. Furthermore, in many CAD applications, the input models come as unstructured models or polygons soups [LMCG96] and it is very hard to decompose them into convex pieces. As a result, most planning algorithms use hierarchical techniques for proximity queries. Some of the simplest hierarchies are based on the use or variants of spheres or axis-aligned bounding boxes [Qui94, HKM95]. These bounding volumes are used for simple rejection tests and work well when the objects are far away.

However, in several instances such as those encountered in maintainability study systems, objects are in close proximity of each other. As a result, a *very significant* fraction of the motion planning algorithm based on potential field approaches [Lat91, KLMR95] or global methods [CL90], is spent on collision detection or distance computation routines. According to Latombe [Lat91], in many cases, 90% of the overall time of a path planner based on potential field techniques is spent on distance computation. This can cor-

*Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, NSF grant CCR-9625217, ONR Young Investigator Award and Intel.

respond to tens of CPU hours for assembly maintainability study on large CAD models [CL95]. The use of successive spherical approximations [BK88] was used for the rapid detection of collisions in a dynamic environment like a robot being tested for collisions with obstacles along a specified path.

Recently, Gottschalk et al. [GLM96] have proposed the use of oriented bounding boxes (OBB's) for fast interference detection. In terms of shape approximation, OBB's provide local quadratic convergence to the underlying geometry, as compared to linear convergence for hierarchies based on sphere or axis-aligned bounding boxes. Consequently, for many close proximity situations, higher order bounding volumes like OBB's offer better overall performance for proximity queries.

Main Contribution: In this paper, we introduce hierarchies based on *spherical shells* and use them for proximity queries. A spherical shell corresponds to a portion of the volume between two concentric spheres and encloses the underlying geometry. We present efficient algorithms to compute such volumes and fast overlap tests between two shells. The overlap test is only slower by a small factor (2 or 3 times as compared to the overlap test between two OBB's). Their main advantage comes from the fact that they provide local *cubic* convergence to the underlying geometry. By "cubic convergence" we mean that the bounding volume approximates the surface accurately up to the second order (if the surface is expressed as a Taylor's series). Therefore, there are fewer "false positives" in terms of overlap tests between the bounding volumes and the underlying primitives. This results in improved overall performance for proximity queries between two objects in close proximity configurations or between two objects with high-curvature surfaces. However, our results show that the local cubic convergence is restricted to elliptic (unlike hyperbolic) regions of the surface.

Organization: The rest of the paper is organized in the following manner. We survey related work in Section 2 and give an overview of our approach in Section 3. An efficient algorithm to compute the spherical shell is presented in Section 4. We analyze the asymptotic performance of a spherical shell as a bounding volume in Section 5. Section 6 highlights a fast overlap test between two shells. Finally, in Section 7 we describe our implementation and compare its performance with other bounding volumes. We conclude in Section 8.

2 Related Work

The problem of interference detection has been widely studied in computational geometry, robotics and computer simulated environments. Most of the earlier work in robotics and in computational geometry has focussed on collision detection between convex polytopes. Good theoretical and practical approaches based on linear complexity of the linear programming problem are known as well [Meg83, Sei90]. Using the Minkowski difference and convex optimization techniques, algorithms with expected linear time performance are also given in [GJK88] to compute the distance between convex polytopes and keep track of closest features. A representation scheme for rapid collision detection in a dynamic environment using successive spherical approximations (SSA) was discussed in [BK88]. The SSA representation is very similar to the spherical shells presented in this paper. However, there are a number of significant differences in the two approaches. They restrict the geometry to be topologically closed polygonal objects. We can deal with "polygonal soups" without any topology information. Their hierarchy computation is restricted to exactly four levels unlike ours which is dependent on the complexity of the underlying surface. Their collision detection algorithm can become inefficient if the object contains a large number of faces. Their collision detection scheme is very conservative because they handle only the trivial accepts and rejects, and push the complicated cases further down the tree. Further, they do not handle all possible configurations of two objects (like one object completely inside the other) during collision detection. Last, but not the least, the parameters required to represent the bounding volume is essentially different (they used a pyramidal structure, whereas we use a conical representation).

Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [DK90] for polytope-polytope overlap problem, where n is the number of vertices. As

for curved objects, many algorithms are presented for models whose trajectory can be expressed as a closed form function of time [HBZ90]. These methods use either subdivision methods, interval arithmetic or bounds on derivatives and constrained minimization. However, they are fairly slow in practice and not adequate for most of online planning algorithms or interactive dynamic simulation.

In applications involving rigid motion, geometric coherence has been utilized to design algorithms for convex polyhedra based on local features [Bar90, LC91, Lin93]. Local properties have been used in the earlier motion planning algorithms by [Don84, LPW79] when two objects come into contact. These algorithms utilize the spatial and temporal coherence between successive instances and work well in practice.

Most environments consists of multiple objects and performing $O(n^2)$ pairwise interference detection becomes a bottleneck for large n. Algorithms of complexity $O(nlog^2n+m)$ have been presented for spheres in [HSS83] and rectangular bounding boxes in [Ede83], where m corresponds to the number of overlaps. More recently, Cohen et al. have presented algorithms and a system, I-COLLIDE, based on spatial and temporal coherence, for large environments composed of multiple moving objects [CLMP95].

A number of hierarchies have been used for collision detection between general polygonal models. 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 [Sam89], sphere trees [Hub93, Qui94], trees based on S-bounds [Cam91] etc. Other spatial representations are based on BSP's and its extensions to multi-space partitions [WG91], spatial representations based on space-time bounds or four-dimensional testing [AANJ94, Cam90, Can86, Hub93] 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. Many researchers have proposed hierarchical data structures based on other bounding volumes. More recently, Gottschalk et al. [GLM96], have presented a fast algorithm and a system, called RAPID, for interference detection based on oriented bounding boxes. Barequet et al. [BCG+96] have also used oriented bounding boxes for computing hierarchical representations of surfaces in 3D and collision detection. Klosowski et al. [KHM+96] have used bounding volumes corresponding to k-DOP's for fast collision detection.

3 Overview

Any interference detection algorithm based on hierarchical data structures involves computing a hierarchical decomposition of the model and bounding volumes corresponding to each node of the tree. At runtime, the bounding volumes are transformed based on objects' motion and used for performing a quick overlap test. The total cost for interference detection is given by the following *cost* equation:

$$T = N_u \times C_u + N_v \times C_v + N_p \times C_p, \qquad (1)$$

where

T: total cost function for interference detection,
N_u: no. of bounding volumes updated due to object's motion,
C_u: cost of updating a bounding volume,
N_v: no. of bounding volume pair overlap tests
C_v: cost of testing two bounding volumes for overlap,
N_p: number of primitive pairs tested for interference,
C_p: cost of testing two primitives for interference,

This equation differs from the cost function presented in [GLM96]. Here we include a separate term for updating each hierarchy based on objects' motion.

For simpler hierarchies based on spheres or axis-aligned bounding boxes, C_v is a small constant (6 to 10 arithmetic operations). For higher order bounding volumes, C_v can be much higher. For example, for OBB's,

the average value of C_v is about 100 arithmetic operations [GLM96]. Since a higher order bounding volume provides a tighter fit to the enclosed geometry, N_v and N_p are relatively lower for them. The overall performance of different hierarchies varies with the geometric models and their configurations.

Given a large model composed of polygonal primitives, we present algorithms to compute a tight fitting shell around them. Initially, each primitive is decomposed into triangles. Each shell is represented by the center of the spheres, inner and outer radius, an axis vector and the solid angle subtended at the center. We use a top down approach to build the hierarchy and decompose the model by dividing the shell volume into two halves and checking for triangles in each half. Each leaf node corresponds to a triangle. Given two hierarchies composed of spherical shells, the algorithm recursively checks two bounding shells for overlap. If they overlap, the algorithm traverses down the tree and checks the children node. Eventually, it checks the triangles at the leaf node for overlap.

In Section 5, we show that spherical shells provide a local cubic convergence to the enclosed geometry. Most of the earlier work in the literature has focussed on hierarchies providing linear or quadratic convergence only. As a result, for many geometric objects in close proximity, N_v and N_p for hierarchies based on spherical shells are much lower (as compared to other hierarchies) and this results in a faster interference detection algorithm. The same argument can be extended to distance computation algorithms.

4 Building the Shells

In this section, we define a shell and present methods to compute shells for polygonal models. Consider two concentric spheres. Let V_1 be the portion of space outside the smaller sphere but inside the larger sphere. Now, suppose there is a cone whose apex coincides with the common center of the two spheres. Let V_2 be the collection of points in the interior of the cone. Then, the volume in the intersection of volumes V_1 and V_2 is called a *spherical shell*. Without loss of generality, we assume that the common center of the two spheres and the apex of the cone lie at the origin. Then, mathematically we can define a shell as follows: Let $r_{\max}$ and $r_{\min}$ be the radii of the larger and smaller spheres respectively. Let $\mathbf{A} = (\mathbf{A_x}, \mathbf{A_y}, \mathbf{A_z})$ be a unit vector representing the axis of the cone. Let α be the half-angle of the cone. Then, the shell is defined as a set of all points (x, y, z) satisfying:

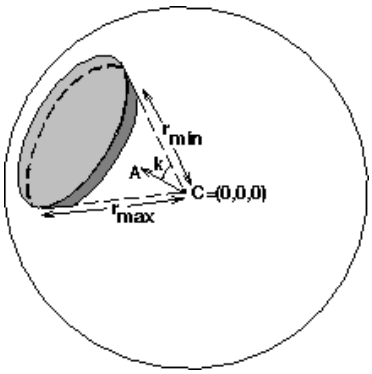

Figure 1: *Spherical shell*

$$r_{\min}^2 \leq x^2 + y^2 + z^2 \leq r_{\max}^2$$
$$\frac{xA_x + yA_y + zA_z}{\sqrt{x^2 + y^2 + z^2}} \geq \cos\alpha \qquad (2)$$

Figure 1 illustrates a spherical shell.

Thus, a shell is completely determined by the following parameters:

- $C(x, y, z)$, the center,
- $r_{\min}$ and $r_{\max}$, the radii,
- $\mathbf{A}$, a unit vector representing the axis,
- k, the cosine of the half-angle.

In the following section, we describe an algorithm to compute a shell for a given geometry. We assume that the geometry is represented in the form of a set of triangles. No adjacency information is required. As a result, the algorithm can handle all polygonal models. We refer to such models as *triangle soups*. We shall now describe the process of building a shell around a triangle soup.

4.1 Determining the center

The center of the shell is chosen as the center of a good spherical approximation to the given polygonal model. In our case, we choose this approximation to be the least squares fit sphere for all the vertices of the input model. We formulate a set of equations for this least squared problem and obtain a solution using Singular Value Decomposition (SVD) [Law74]. We now present the SVD formulation.

The general equation of a sphere with center (c_x, c_y, c_z) and radius r is $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - r^2 = 0$. Substituting $\frac{r^2 - c_x^2 - c_y^2 - c_z^2}{2}$ by d and rearranging the above equation, we get:

$$c_x x + c_y y + c_z z + d = \frac{x^2 + y^2 + z^2}{2}$$

In our case, c_x, c_y, c_z and d are unknowns. Each of m points (x_i, y_i, z_i) $(i = 1, 2, \ldots, m)$ forms a row of a *linear system*. This results in a $m \times 4$ matrix which is an over-determined system. The result obtained by SVD gives us the vector (c_x, c_y, c_z, d). It is trivial to determine the center and radius of the sphere from this vector.

A concern here is the size of the matrix on which we perform the SVD. Say it is an $m \times n$ matrix. For our problem, n is equal to 4, but m equals three times the number of triangles in the input model. Since the cost of performing an SVD is $O(m^2 n)$, for soups containing a large number of triangles, the computational cost incurred would be prohibitive. In order that the computation of shells be fast, we need to ensure that the size of this matrix remains fairly small even if the number of triangles in the soup increases. For this, we perform a spatial subdivision of the vertices in the triangle soup into an octree hierarchy. Subdivision is performed until the octree has approximately N leaves, each with roughly equal number of triangles, where N is a predetermined constant. We compute the mean of vertices belonging to each leaf. This leaves us with a set of approximately N points. We use these to compute the best fit sphere. In our implementation, we choose $N \approx 100$. Since N is approximately a constant, the cost incurred in the SVD computations is a constant. As a result, the overall cost of computing the center is linear in terms of number of vertices. We choose the center of this best fit sphere to be the center of the shell. It is difficult to justify the use of this approach to determine the center and (later) the radii of the shells. In our experience, this approach has resulted in fairly satisfactory shell fits.

4.2 Computing $r_{\max}$

Now that we have the center, we want to compute the radius of the outer sphere. By definition of the shell, the entire geometry should lie inside the outer sphere. The entire geometry will lie inside this sphere if all vertices of the geometry lie inside the sphere. Therefore, we compute the distance of each vertex from the center of the shell and choose the maximum of these distances to be the $r_{\max}$.

4.3 Computing $r_{\min}$

From the definition of a shell, the entire geometry should lie outside the inner sphere. However, the exterior of a sphere is not a convex volume. As a result, looking only at the vertices of the geometry to determine the radius of the inner sphere will not guarantee that the entire geometry lies outside this sphere. Therefore, we calculate the minimum distance of each triangle in the model from the center of the shell and choose the minimum of these distances to be the $r_{\min}$.

4.4 Computing the axis

Consider the set of unit vectors from the center of the shell pointing towards the centroids of the triangles in the geometry. We compute a weighted mean of these unit vectors, where the weights are the areas of respective triangles. The vector so obtained is normalized and is treated as the axis of the shell. The intuition behind using area weighted mean is as follows: suppose some section of the geometry has a much finer tessellation than the rest. If we were to use an unweighted mean, then the axis would tend to be oriented towards the direction of the finer tessellation. Using an area weighted mean prevents this bias.

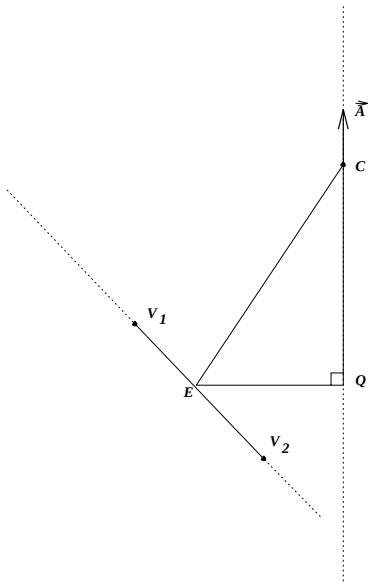

Figure 2: *Computing correct shell angle*

4.5 Computing the angle

For each vertex, we compute the cosine of the angle between the axis and the vector joining the center of the shell to the vertex. Since we require the entire geometry to lie inside the cone, we choose the minimum of all these cosines (i.e. the maximum of all these half-angles). If the half-angle is less than 90°, then the inside of the cone is a convex volume and thus, looking at only the vertices is sufficient to guarantee that the whole geometry lies in the interior of the cone. However, if the half-angle is greater than 90°, then the inside of the cone becomes a concave volume. Now, looking at only the vertices is no longer sufficient since portions of the geometry can lie outside the cone even if all its vertices lie inside the cone. In order to take care of this problem, we use the following algorithm:

Consider Figure 2. Here, **C** is the center of the shell, **A** is the axis and $\mathbf{V_1V_2}$ is an edge of some triangle in the given geometry. Consider a point **E** on the edge $\mathbf{V_1V_2}$. **Q** is the point of intersection of a line passing through **C** and parallel to **A** with a line passing through **E** and perpendicular to **A**. We want **E** to be such that the angle that line **CE** forms with the axis **A** is maximized. In other words, we want to minimize angle **ECQ**, subject to the constraint that the point **E** lies on edge $\mathbf{V_1V_2}$.
We represent **E** parametrically as:

$$\mathbf{e = V_1 + t(V_2 - V_1)} \quad \cdots \quad \mathbf{0 \le t \le 1} \tag{3}$$

In order to minimize angle **ECQ** (this angle is less than 90°), we should to minimize $\frac{|\mathbf{EQ}|}{|\mathbf{CQ}|}$. Instead, lets minimize $\frac{|\mathbf{EQ}|^2}{|\mathbf{CQ}|^2}$. We can write the position vector of point **Q** as:

$$\mathbf{q = c + ka}$$

where **c** is the position vector of the center and **a** is the axis vector. Since $\mathbf{EQ} \perp \mathbf{CQ}$, we have

$$\begin{aligned} &(\mathbf{e - q}) \cdot (\mathbf{c - q}) = \mathbf{0} \\ \therefore \quad &(\mathbf{e - c - ka}) \cdot (\mathbf{-ka}) = \mathbf{0} \end{aligned} \tag{4}$$

This leaves us with 3 cases:

Case 1: $k = 0$ All such points **E** for which this holds have already been taken care of in our original algorithm. So, we neglect this case.

Case 2: $\mathbf{e - c = ka}$ This vector equation forms a system of three scalar equations in two unknowns, (t, k). We solve this system if it is feasible. If the solution (t, k) lies between $([0, 1], (-\infty, 0]))$, then we set the half-angle to 180°. Otherwise, we neglect this case.

Case 3: $(\mathbf{e \cdot a}) - (\mathbf{c \cdot a}) - \mathbf{k}(\mathbf{a \cdot a}) = \mathbf{0}$ Here $\cdot$ denotes dot product. We use this equation to substitute for k in:

$$f(t) = \frac{|\mathbf{EQ}|^2}{|\mathbf{CQ}|^2}$$

This gives us an equation in one unknown t, which can be written in the form:

$$f(t) = \frac{k_1 t^2 + k_2 t + k_3}{k_4 t^2 + k_5 t + k_6}$$

where $k_1, \ldots, k_6$ are known constants. Since we want to minimize $f(t)$, we solve for $f'(t) = 0$. This gives us a quadratic in t and in general, it will have two roots. If none of the roots lies in the interval

$[0,1]$, then we neglect this case. If one or both roots lie in $[0,1]$, then we compute the half-angle subtended by these points and update our half-angle if required (this takes care of turning points that maximize our objective function).

This procedure is repeated for each edge in the geometry. Thus, we have looked at all cases where the point limiting the half-angle of the cone lies on the edges of the geometry. There is a possibility that a point in the interior can limit the cone angle. It can be observed that such a case will arise only if a ray originating through the center of the shell in the negative direction of the axis intersects one of the triangles in the geometry. This case can be checked by ray shooting and if it occurs, we set the half-angle to 180°.

The complexity of fitting a shell to a soup of triangles (without any topological information) as described here is determined by the complexity of each step - determining center, minimum radius, maximum radius, shell axis and shell angle. Determining the center involves an octree decomposition followed by SVD. In our case, SVD takes constant time since the matrix size is fixed. The octree decomposition is performed by first placing a tight axis aligned bounding box around the entire point set belonging to an octree node and then partitioning the point set based on the position of points inside this bounding box. As a result, even in the worst case, the time complexity of the octree decomposition would be $O(n^2)$. In practice, however, it would be extremely rare to find a point set corresponding to a surface mesh that would exhibit this worst case complexity. In most cases, we observe the average case complexity of $O(n \log n)$. All the other steps in the shell building algorithm take linear time in the number of triangles.

5 Convergence Analysis

In this section, we show that the bounding volume of spherical shells approximates a smooth surface up to the second order term. In other words, the errors in the approximations decrease at a cubic rate.

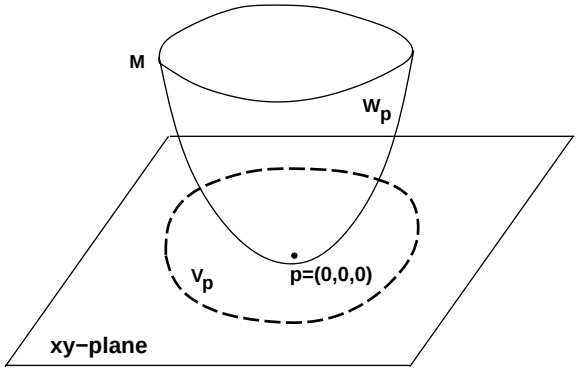

Figure 3: Quadratic approximation of a surface

5.1 Surface Approximations using Power Series

Consider a 2-manifold $\mathbf{M} \subset \mathcal{R}^3$ and a point $p \in \mathbf{M}$ as shown in Figure 3. Without loss of generality, we make the following assumptions about p and $\mathbf{M}$.

- p is the origin,
- the tangent plane of $\mathbf{M}$ at p ($T_p(\mathbf{M})$) is the $z = 0$ plane, and
- the two principal directions of $\mathbf{M}$ at p are the coordinate axes $e_1 = (1,0,0)$ and $e_2 = (0,1,0)$.

It is easily seen that these conditions can be achieved by a simple rigid transformation of $\mathbf{M}$. To proceed further, we make use of a result from classical differential geometry which we state here without proof.

Theorem 1 *[O'N66] There exists a small neighborhood W_p of $p \in \mathbf{M}$ such that the map $\pi : (x,y,z) \Rightarrow (x,y)$ is a one-to-one map with its image being an open set $V_p \subset \mathcal{R}^2$. Moreover the map π is a diffeomorphism.*

The fact that π is a diffeomorphism implies that π^{-1} exists and that π and π^{-1} are smooth mappings. Therefore, we can approximate the surface in the neighborhood of p (W_p) as:

$$W_p = \{(x,y,F(x,y)) : (x,y) \in V_p\}$$

Here, $F(x,y)$ intuitively represents the surface as a height function. Further, the tangent plane of $W_p \subset \mathbf{M}$

is given by the basis vectors $\frac{\partial W_p}{\partial x} = (1, 0, \frac{\partial F}{\partial x}(0,0))$ and $\frac{\partial W_p}{\partial y} = (0, 1, \frac{\partial F}{\partial y}(0,0))$. Since, we assumed that the tangent plane at p is the $z = 0$ plane, $\frac{\partial F}{\partial x}(0,0) = \frac{\partial F}{\partial y}(0,0) = 0$.

Shape operator is a familiar concept in differential geometry. Essentially, the shape operator at a point $p \in \mathbf{M}$ (denoted by $S_p(\mathbf{M})$) is a linear operator that maps an element of $T_p(\mathbf{M})$ to another element in $T_p(\mathbf{M})$. If v_{p1} and v_{p2} are a set of basis vectors for $T_p(\mathbf{M})$, $S_p(av_{p1} + bv_{p2}) = cv_{p1} + dv_{p2}$ (a, b, c and d are real valued scalars). For the special case of a vector v being a principal direction, $S_p v = Kv$, where K is the principal curvature. In our particular case, the tangent plane is spanned by e_1 and e_2. The shape operator applied to the vectors e_1 and e_2 are given by:

$$\begin{aligned} S_p e_1 &= \frac{\partial^2 F}{\partial x^2}(p)e_1 + \frac{\partial^2 F}{\partial x \partial y}(p)e_2 \\ S_p e_2 &= \frac{\partial^2 F}{\partial x \partial y}(p)e_1 + \frac{\partial^2 F}{\partial y^2}(p)e_2 \end{aligned} \tag{5}$$

Since e_1 and e_2 are the principal directions, we can conclude that $\frac{\partial^2 F}{\partial x \partial y}(0,0) = 0$ and that $\frac{\partial^2 F}{\partial x^2}(0,0)$ and $\frac{\partial^2 F}{\partial y^2}(0,0)$ are the principal curvatures (denoted by k_1 and k_2 respectively).

We now use Taylor's formula to expand $F(x, y)$ around the origin $(0, 0)$. Thus:

$$\begin{aligned} F(x,y) &= F(0,0) + x\frac{\partial F}{\partial x}(0,0) + y\frac{\partial F}{\partial y}(0,0) \\ &\quad + \frac{1}{2}(x^2\frac{\partial^2 F}{\partial x^2} + 2xy\frac{\partial^2 F}{\partial x \partial y} + y^2\frac{\partial^2 F}{\partial y^2}) \\ &\quad + \text{higher order terms} \\ &= \frac{1}{2}(k_1 x^2 + k_2 y^2) + \text{higher order terms} \end{aligned}$$

For smooth surfaces with low curvature variations, bounding volumes like shells approximate the surface up to the second order adequately. In fact, $\frac{1}{k_1}$ and $\frac{1}{k_2}$ provide bounds for the minimum and maximum radii of the shells. A caveat is in order here. The approximation of the surface we derived above is approximated well with spherical shells only if the principal curvatures have the same sign (*i.e.,* the surface is elliptic). For hyperbolic surfaces or surfaces with high curvature variations, these bounding volumes may not exhibit cubic convergence.

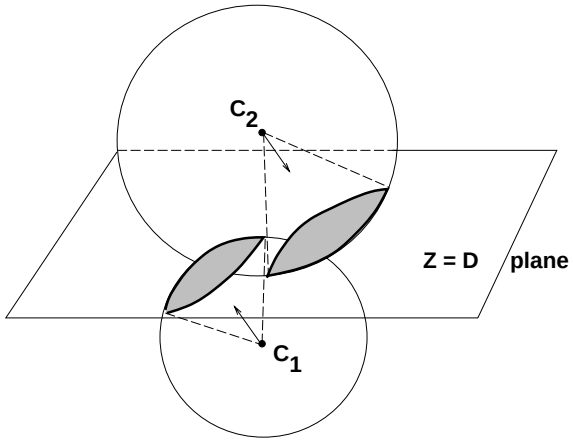

Figure 4: Shells intersecting at specific radii

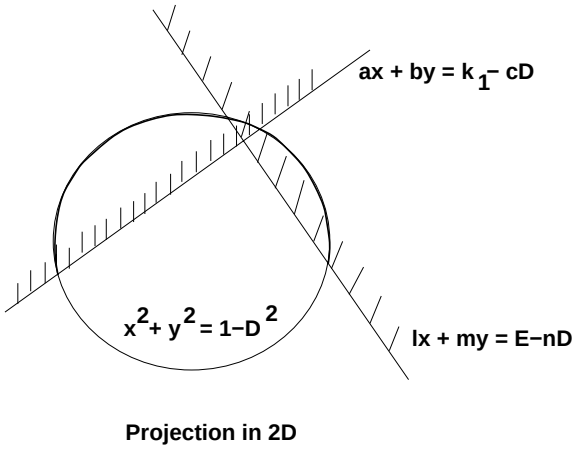

Figure 5: Reduction of overlap test to the plane

6 Overlap Tests between Shells

In this section, we briefly discuss the overlap test between two shells. Before we start, some notation has to be introduced. Without loss of generality, the two shells S_1 and S_2 are determined by the following parameters:

- **shells:** S_1, S_2

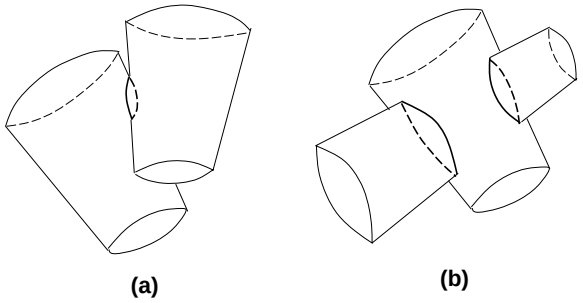

Figure 6: Orientation of shells that do not involve extreme radii

- **centers:** $C_1 = (0,0,0)$, $C_2 = (0,0,d)$
- **axis vectors:** (a,b,c), (l,m,n)
- **cosines of half angle:** k_1, k_2
- **radii:** $[r_{1n}, r_{1x}]$, $[r_{2n}, r_{2x}]$

We start by answering a simpler test. Let us define the set of points in the shell volume that are at a constant distance r from the center as the *shell restricted to* r. A simpler test is to see if the shell S_1 restricted to r_1 intersects shell S_2 restricted to r_2 (see Figure 4). Given the above parameters, the two spheres intersect along the plane $z = D^{'}$, where $D^{'} = \frac{d^2+r_1^2-r_2^2}{2d}$. Any intersection of the two restricted shells can only occur on this plane. Let us now write the algebraic constraints for the two restricted shells. The first of the two equations is the sphere equation and the second is the constraint for it to lie within the cone. This is obtained by taking a simple dot product. For S_1,

$$\begin{aligned} x^2 + y^2 + z^2 &= r_1^2 \\ ax + by + cz &\geq r_1 k_1 \end{aligned} \tag{6}$$

and for S_2,

$$\begin{aligned} x^2 + y^2 + (z-d)^2 &= r_2^2 \\ lx + my + n(z-d) &\geq r_2 k_2 \end{aligned} \tag{7}$$

Subtracting the two sphere equations, we get $z = D^{'}$. The equation of the intersecting circle becomes $x^2 + y^2 = r_1^2 - D^{'2}$. Therefore, the two restricted shells intersect if there is a point $(x, y, D^{'})$ that satisfies

$$\begin{aligned} x^2 + y^2 &= r_1^2 - D^{'2} \\ ax + by &\geq r_1 k_1 - cD^{'} \\ lx + my &\geq r_2 k_2 + nd - nD^{'} \end{aligned} \tag{8}$$

Scaling all the equations in (8) by a factor $\frac{1}{r_1}$ and replacing $D^{'}/r_1$ by D and $(r_2 k_2 + nd)/r_1$ by E, we get

$$\begin{aligned} x^2 + y^2 &= 1 - D^2 \\ ax + by &\geq k_1 - cD \\ lx + my &\geq E - nD \end{aligned} \tag{9}$$

Essentially, the test boils down to checking if there exists some point on the circle which lies in the positive half-space of both lines (see Figure 5). To check for satisfaction of (9), let us parameterize x and y in the circle by

$$x = \sqrt{1-D^2}\frac{1-t^2}{1+t^2}, \qquad y = \sqrt{1-D^2}\frac{2t}{1+t^2}$$

Substituting this in the other two inequalities and simplifying, we get

$$(\frac{k_1 - cD}{\sqrt{1-D^2}} + a)t^2 - 2bt + (\frac{k_1 - cD}{\sqrt{1-D^2}} - a) \leq 0 \tag{10}$$

and

$$(\frac{E - nD}{\sqrt{1-D^2}} + l)t^2 - 2mt + (\frac{E - nD}{\sqrt{1-D^2}} - l) \leq 0 \tag{11}$$

This test can be performed easily by computing the roots of each of these equations (if they exist in the real domain) and checking if the appropriate intervals where the signs of the equations are non-positive have an intersection. However, solving the quadratic equations involves a couple of square roots. Square root is a relatively expensive operation as compared to other arithmetic operations like the additions and multiplications. If we can avoid square roots in our computation, it would really speed up our overlap test. It is fairly simple to still carry out the test without computing their actual roots.

The discussion so far has concentrated on the case when both shells are restricted (r_1 and r_2 are fixed).

However, in our case, r_1 and r_2 range from $[r_{1n}, r_{1x}]$ and $[r_{2n}, r_{2x}]$ respectively. This introduces two variables to the system of constraints and makes the test a lot more difficult. We can reduce the number of variables in our system to one if we observe that when two shells overlap, at least one of the extreme radii (r_{1n}, r_{1x}, r_{2n} or r_{2x}) of the shells is also involved except for the cases shown in Figure 6. So our test for overlap between two shells reduces to four overlap tests between a shell restricted to one of its extreme radii and another shell. The tests described previously carries through to this case as well. However, the signed expression we are evaluating changes to a (quartic) polynomial in a given range. We use Sturm sequences [SB93] to evaluate the signs of these polynomials without actually performing root computation. We omit many of the details here. But a detailed version of the test can be found in [KPLM97].

7 Implementation and Performance

The software for the collision detection library was written in C++. The main data structure is a "shell" which holds all the required parameters for a shell. Given a model, we initially build a hierarchy of shells which can be used for performing fast collision detection. We first construct a top-level shell enclosing all the triangles in the given soup. Further levels are generated by subdividing the geometry and fitting individual shells. While building the top-level shell, we keep track of the point p which limits the shell angle. Let v be the vector from p perpendicular to the axis vector. To build the shells at the next level, we consider the plane with normal v and containing both, the axis of the shell and also the center. We use this plane as a separating plane to partition the triangles in the given soup. The partitioning is based on the centroids of the triangles. If a centroid lies on the separating plane, then we arbitrarily assign the triangle to one of the partitions. This leaves us with two triangle soups. We build shells for these two soups and proceed recursively until we have one triangle per soup. Thus, we get a shell tree for each input model. In our shell building code, we use the SVD routines from the public domain FORTRAN libraries LAPACK and EISPACK.

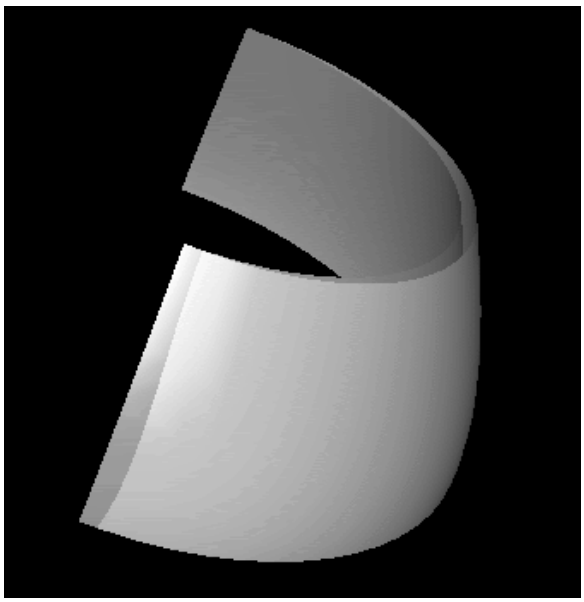

Figure 7: *Two highly tessellated bicubic patches used in the simulation. The patches are placed in parallel close proximity positions and we compare the performance of different bounding volumes on such configurations.*

The method we have described to subdivide the geometry is purely heuristic in nature. However, without any connectivity information in the underlying model, it is very difficult to provide a provably better subdivision strategy.

In the current implementation, tree building is static, in the sense that we compute the entire shell tree for a given object before we start making any collision queries on it. However, in the future, we would like to look at building shell trees dynamically. This would reduce the memory usage significantly.

7.1 Parallel Close Proximity Test

We have tested our system on some models. Of particular interest is the case of parallel close proximity since that is where convergence plays a major role. In one of our simulations, we used two identical models, each of which consisted of two patches from the side of the Utah teapot. This model is shown in Figure 7. The patches have slowly varying curvature and are convex everywhere. This is admittedly a good case for the

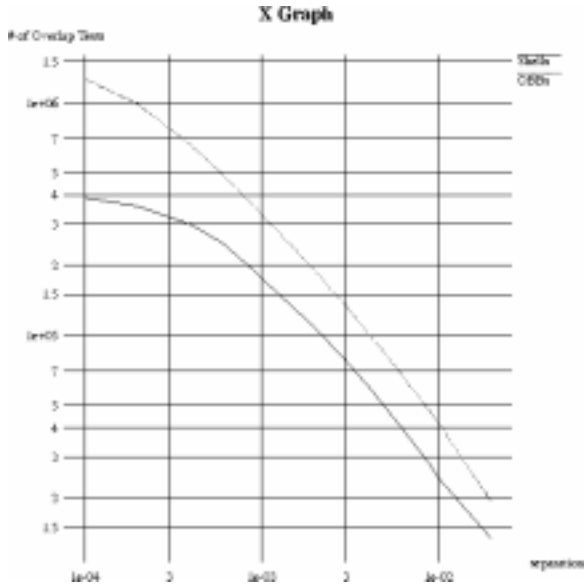

Figure 8: OBBs (upper curve) and Shells (lower curve) for parallel close proximity

shells, but we wanted to study the empirical convergence of such cases. As a result, from the analysis presented in the previous sections, shells should provide cubic convergence for these models. Each of these models consisted of 50,176 triangles. In each frame of the simulation, we displaced one model slightly relative to the other and performed a collision query. The simulation consisted of several such frames, each with a different displacement between the models. We performed the same simulation using both shells and OBBs, and for each collision query, the number of overlap tests required were counted. The results obtained have been summarized in Table 1.

Figure 8 shows a log-log plot of the number of collision queries required for both types of hierarchies as a function of the displacement between models. The upper curve is for OBBs while the lower one is for Shells. The differing slopes of the two curves imply that shells require asymptotically fewer overlap tests than OBBs, as a function of the displacement between the models in our experiment. Therefore, shells have a higher convergence than OBBs. As a result, for large models composed of thousands of polygons in close proximity

S.No.	Displacement	Shells	OBBs
1	0.1	2,277	2,063
2	0.05	4,957	5,571
3	0.02	13,353	19,333
4	0.009	27,795	47,007
5	0.006	41,257	70,091
6	0.003	77,545	133,189
7	0.0009	190,105	360,163
8	0.0006	248,607	490,057
9	0.0004	297,677	651,353
10	0.0002	357,333	982,415
11	0.0001	389,175	1,256,949

Table 1: *Overlap tests required for different displacements*

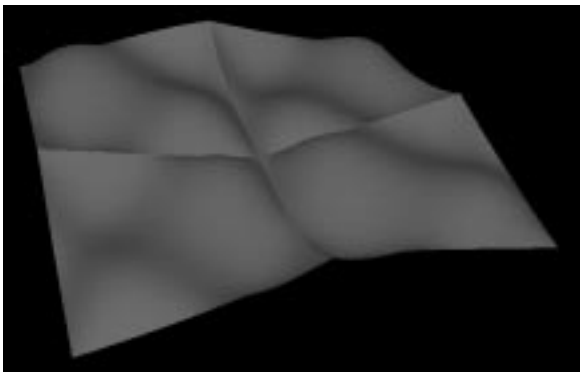

Figure 9: *Model consisting of around 131,000 triangles used in the volume test*

positions, hierarchies based on spherical shells provide faster interference detection algorithms.

7.2 Volume Test

The experiment conducted here was to measure the order of convergence empirically. Given a model composed of triangles, we constructed a hierarchy of spherical shells and OBBs. We use the same subdivision scheme to partition the model as we go down the hierarchy. Each node in the tree was subdivided into four children. The volume test was performed by accumulating the volume of all the spherical shells (or OBBs) occurring at a particular level of the tree. The ratio of the volumes at successive levels gives us an insight into the empirical convergence.

The model we used to conduct the volume test is

Level No.	Shells		OBBs	
	Volume	Ratio	Volume	Ratio
1	237.37	–	83.432	–
2	35.1976	6.74	28.9779	2.88
3	5.31472	6.62	7.62836	3.80
4	0.969967	5.48	1.87269	4.07
5	0.250267	3.88	0.442474	4.23

Table 2: *Bounding volume ratios at various levels*

Triangles per torus	Collision calls	Time (in micro secs.)		
		Tree const.	Coll. Query	Avg. Coll. Query
450	2553	8.01e+6	232469	34.46
882	5148	2.36e+7	482771	49.06
1152	8116	8.61e+6	167086	51.53
1458	8842	9.00e+6	207463	43.38
1800	11971	1.43e+7	362056	39.54
2178	13049	1.40e+7	369196	38.10

Table 3: *Performance of collision detection algorithm*

shown in Figure 9. It consists of 131,072 triangles. We built two hierarchies: one with spherical shells and another with OBBs. For a shell with inner and outer radii r_1 and r_2, respectively, and cosine of half angle k, its volume is given by the formula $\frac{2\pi(r_2^3-r_1^3)(1-k)}{3}$. The volume computation for an OBB is quite trivial. We build the tree to 5 levels (all leaves had 512 triangles each). The results are summarized in Table 2. We observe that the ratios of volumes at the higher levels is much higher for spherical shells than for OBBs. We expect a ratio of around 8 for ideal cubic convergence. We were unable to find a case for which spherical shells reached this ratio. But we were able to achieve super-quadratic convergence. The ratios fall to about 4 near the lower levels. We believe that the reason for this has to do with the relatively flat nature of the geometry at these levels.

7.3 Timing Measurements

We tested the performance of our system with a set of five finely tessellated *interlaced tori*. Each torus moves in a direction opposite to its adjacent torus. On collision, the direction of motion of both the tori are reversed. We ran this experiment for 200 frames. The simulations were repeated for varying tessellations of the tori. The results are summarized in Table 3. The columns, in order, represent the number of triangles per torus, number of calls made to the collision detection system, total time taken to construct the entire hierarchy, total and average time required to answer the collision query. In this simulation, the average number of collisions reported was 55, the average collision query time was 42 microseconds, and the average number of calls made to the collision detection system was 303,506. All the timings presented here were measured on an SGI-Onyx with an R10000 processor, 195 MHz clock and 128 MB main memory.

8 Conclusion

In this paper, we have introduced a new bounding volume called spherical shells. We present efficient algorithms to build such shells for general polygonal models and for performing fast overlap tests between shells. The shells provide local cubic convergence to the underlying geometry. We compare its performance to other hierarchies and have shown that for many close proximity situations, the hierarchies based on spherical shells provide faster interference detection algorithms.

We are currently working on a public domain interference detection system based on spherical shells. Besides close proximity configuration, it would also be useful for highly tessellated models or curved models. We also plan to use them for fast distance computation between geometric models.

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.

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

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

[BK88] S. Bonner and R. B. Kelley. A representation scheme for rapid 3-d collision detection. *IEEE International Symposium on Intelligent Control*, pages 320–325, 1988.

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

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

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

[CL90] J. F. Canny and M. C. Lin. An opportunistic global path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1554–1559, 1990.

[CL95] H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proceedings of International Conference on Robotics and Automation*, 1995.

[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.

[DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – A unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes Comput. Sci.*, pages 400–413. Springer-Verlag, 1990.

[Don84] B. R. Donald. Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab., 1984. AI-TR-791.

[Ede83] H. Edelsbrunner. A new approach to rectangle intersections, part i. *Int. J. of Comput. Math*, 13:209–219, 1983.

[FKL+97] P. Finn, L. Kavraki, J.-C. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: randomized pharmacophore identification for drug design. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 324–333, 1997.

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

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

[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.

[HSS83] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. Efficient detection of intersections among spheres. *The International Journal of Robotics Research*, 2(4):77–80, 1983.

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

[KHM+96] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. In *Siggraph'96 Visual Proceedings*, 1996.

[KLMR95] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 353–362, 1995.

[KPLM97] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shells: A higher-order bounding volume for fast proximity queries. Technical report, Department of Computer Science, University of North Carolina, 1997.

[Lat91] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[Law74] C. Lawson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

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

[Lin93] 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.

[LMCG96] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In *Algorithms for Robot Motion and Manipulation*, pages 129–142. A K Peters, 1996.

[LPW79] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[Meg83] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Computing*, 12:pp. 759–776, 1983.

[O'N66] B. O'Neill. *Elementary Differential Geometry*. Academic Press, London, UK, 1966.

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

[RKK97] D.C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, pages 345–352, 1997.

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

[SB93] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, 1993.

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

[SWZ89] T.W. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[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.

[ZL91] David Zhu and Jean-Claude Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. Robot. Autom.*, 7:9–20, 1991.