Efficient Contact Determination Between Geometric Models

Ming C. Lin
Dept. of Electr. Engr.
and Computer Science
University of California
Berkeley, CA 94720

Dinesh Manocha *
Computer Science Department
Sitterson Hall, CB #3175
University of North Carolina
Chapel Hill, NC 27599-3175

Abstract:

The problem of interference detection or contact determination between two or more objects in dynamic environments is fundamental in computer graphics, robotics and computer simulated environments. Most of the earlier work is restricted to either polyhedral models or static environments. In this paper, we present efficient algorithms for contact determination and interference detection between geometric models undergoing rigid motion. The set of models include polyhedra and surfaces described by algebraic sets or piecewise algebraic functions. The algorithms make use of temporal and spatial *coherence* between successive instances and their running time is a function of the motion between successive instances. The main characteristics of these algorithms are their simplicity and efficiency. They have been implemented; their performance on many applications indicates their potential for real-time simulations.

Additional Keywords and Phrases: contact determination, geometric coherence, spline models, simulations, animation

*supported in part by Junior Faculty Award, University Research Award and ARPA Contract DABT63-93-C-0048.

1 Introduction

Intersection problems are among the fundamental topics in computational geometry and geometric modeling. They are also considered important in computer animation, physical based modeling, robotics, and computer simulated or virtual environments. In most of these applications, the objects are undergoing motion governed by dynamic constraints or external forces. It is important to determine contact or interference between any pair of objects in a dynamic environment and generate a collision response to it. This problem has received a lot of attention over the last few years in all these areas.

The problems of contact determination and interference detection have been extensively studied in different areas. The literature in computational geometry consists of a number of theoretically efficient algorithms for polyhedral objects in *static* environments [1, 2, 9, 10, 11, 12, 13, 17, 36]. There are a number of algorithms with good asymptotic bounds, however their practical utility is not clear since many of them are not implemented in a realistic environment.

Many algorithms are known for intersection between curved objects represented as algebraic surfaces or piecewise spline models in geometric modeling [14, 25, 33, 24, 37]. However, they are targeted towards robust computations of the intersection curve between such models for CSG operations and boundary computations. They are usually specialized to static environments and are relatively slow for the kind of performance desired in computer simulated environments

Similarly, the problem of collision detection has been addressed in robotics literature as well. However, the main purpose of applications has been on planning a collision-free trajectory between obstacles [8, 15, 27, 31, 43]. This is different from applications in physical based modeling and virtual environments, where the motion is subject to dynamic constraints or external forces and cannot typically be expressed as a closed form function of time. .

The simplest algorithms for interference detection in dynamic environments are based upon using bounding volumes and spatial decomposition techniques. Typical examples of bounding volumes include cuboids, spheres, octrees etc. and they are chosen due to the simplicity of finding interference between two such volumes. These bounding volumes work very well in performing "rejection tests", when two objects are far apart. However, once the two objects are in the vicinity of each other, spatial decomposition techniques based on subdivision are used to solve the interference problem. This can become rather slow in practice as highlighted by Hahn in [22].

In computer animation, algorithms for polyhedral objects of complexity $O(n^2m^2)$, where m is the number of polyhedra with n vertices per polyhedron, are described by

Moore and Wilhelms in [35]. For convex polytopes, linear time algorithms based on linear programming by Megiddo and Seidel [34, 42] and tracking closest points between incremental motion by Gilbert and etc. [21] have been proposed. However, they do not make use of coherence in their methods between successive instances. More recently, geometric coherence has been utilized to devise algorithms based on local features for convex polytopes by Baraff and Lin [3, 28]. This has significantly improved the performance of interference detection algorithms in dynamic environments. As far as curved models are concerned, algorithms based on interval arithmetic have been proposed by Duff, Herzen, Snyder, etc. [16, 23, 44]. They are relatively slow in practice and expect that the motion can be expressed as a closed form function of time. However, no *practical* and efficient interference detection algorithms are known for *general* geometric models in *dynamic* environments.

In this paper, we present efficient algorithms for contact determination and interference detection between two or more objects undergoing rigid motion. The models include polyhedral models and curved objects whose boundary can be represented in terms of algebraic sets. This includes algebraic surfaces and NURBS models frequently used in computer graphics and geometric modeling. No assumption is made on the motion of the objects. At each instance, their position is described using a transformation matrix with respect to the origin. For convex polytopes, the algorithm keeps track of closest features based on convexity of polyhedra as described in [28] and [29]. Concave polytopes are decomposed into convex polytopes and represented by sub-part hierarchical tree.

For curved models we use a combination of hierarchical representations, algorithms for polytopes, and global and local methods for solving systems of polynomial equations. In particular, we use the control polytopes of the objects and their sub-patches along with sub-part hierarchical tree representation to describe them for top-level collision detection. Once the control polytopes come into contact with each other, we algebraically formulate the problem of contact determination and interference detection. The problem is reduced to solving a system of overconstrained algebraic equations. We initially use global algebraic methods based on resultants and matrix computations to compute the solutions in the corresponding domain of interest and update these solutions using local numerical methods as the objects undergo motion.

One of the main contribution of this paper is to present a methodology which captures the temporal and spatial coherence between successive checks to achieve real-time collision detection for polyhedral objects and to determine the contact information between general *curved* models efficiently using the algebraic analog of geometric coherence. When the objects are not in contact with each other, the resulting algorithm for a pairwise test essentially runs in $\theta(1 + f(M))$, where $f(M)$ is a function of relative motion M (relative homogeneous transformation matrix to describe the relative motion) which the objects undergo.

We also analyze the complexity of our approaches to curved models. The complexity of global methods is $O(M^3)$, where M corresponds to the algebraic complexity of the resulting system. It is a function of the degrees of the polynomials used to represent curved models. At each iteration the complexity of the local methods is a function of the algebraic degrees of the boundary surfaces and the motion undergone by the object pairs.

The rest of the paper is organized as follows. We analyze the problem of interference detection, review literature on object models and solutions of algebraic systems in Section 2. In Section 3, we review the expected constant time algorithm for tracking closest features between convex polytopes and show how it can be extended to interference detection between general polyhedral models. The problem of closest features and interference between curved models are analyzed in Section 4. In Section 5, we present algorithms between curved models using equation solving approaches. Finally, we discuss our implementation and performance in Section 6.

2 Background

2.1 Coherence for Detection Problem

The simplest algorithms for interference detection are based on bounding volumes. Typically they correspond to bounding boxes, spheres or octrees etc. In case, two objects are far apart, the bounding volumes approach can be used to determine that there is no interference. These bounding boxes overlap only if the given pair of objects collide or are close to each other. As a result, any good algorithm for interference detection needs to compute the spatial relationship between a pair of objects, when they are close to each other and perhaps moving towards each other. Since no assumptions are made on the motion of the objects, it is not possible to exactly compute the time at which the given pair of objects would collide. Therefore, in most applications the collision detection routines are being called frequently. Due to the nature of application, there exists spatial and temporal coherence between successive instances. Different algorithms in the literature have utilized the coherence. Baraff uses cached edges and faces to find a separating plane between two convex polytopes [3]. However, this may not work when the closest features between the objects change or when there is a large abrupt motion. In [28], an algorithm for keeping track of closest features between convex polytopes using Voronoi regions of the features and local geometry information is used to determine the minimum separation between them, thus to detect possible collision. This works very well for convex polytopes. However, it is difficult to apply this approach directly to non-convex polygonized *curved* models as there may be more than pair of closest features and the closest pair features may correspond to curves on the surface. In Section 5, we make use of the coherence along with local numerical methods and global techniques for solving algebraic systems to check whether there is a contact or interference between such curved models.

2.2 Object Models

Polyhedral models are represented using boundary representation [24]. A polyhedron consists of vertices, edges and faces represented in terms of parameters needed to describe them. In addition, we also have the topological information such as adjacencies and incidences of all features (vertices, edges, faces). Concave polytopes are decomposed into convex polytopes and represented by hierarchical sub-part trees. Curved models consist of NURBS surfaces and piecewise algebraic surfaces. NURBS models are represented in terms of control polytopes, knot vectors and order continuity [19]. The convex hull of the control polytopes serves as a bounding volume approximation for these surfaces. They are further decomposed into a series of Bézier surfaces using knot insertion algorithms. Each Bézier surface corresponds to a rational parametric surface, which can be represented in homogeneous coordinates as:

$$\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)). \tag{1}$$

The Bézier surface is defined over the domain $(s,t) \in [0,1] \times [0,1]$. Algebraic surfaces are represented implicitly as zero sets of the form:

$$f(x,y,z) = 0.$$

Many commonly used models like the quadrics or torus are described using such formulations.

2.3 Solutions of Algebraic Systems

It is shown in Section 5 that the problem of interference detection corresponds to finding common solutions of a system of algebraic equations in a particular domain. The system may be a zero dimensional system with a finite number of solutions or an overconstrained system with no common solutions. There are different well-known methods for solving system of polynomials equations. Most of the algorithms in literature are for a system of n equations in n unknowns represented as:

$$F_1(x_1, x_2, \ldots, x_n) \quad = \quad 0$$

$$\begin{aligned} F_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ F_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \qquad (2)$$

Let their degrees be d_1, d_2, ..., d_n, respectively.

Purely symbolic approaches like Grőbner bases and resultants eliminate variables and reduce the problem to finding roots of a univariate polynomial. These can be slow even for low degree systems and need exact arithmetic. On the other hand iterative methods like the Gauss-Newton's method are good for local analysis and need good initial guesses to all the solutions. In the context of floating point arithmetic, the two main approaches are those of homotopy methods [20] and matrix computations [32]. The latter algorithms make use of matrix formulation of resultants of polynomial equations and reduce the problem to computing eigenvalues and eigenvectors of generalized companion matrices. They have been applied to a number of geometric applications and perform very well in practice [32]. The complexity of the algorithm is $O(M^3)$, where M corresponds to the algebraic complexity of the system. For dense polynomial systems, it is the Bezout bound corresponding to the product of the degrees of the equation; and for sparse polynomial systems, it is the BKK bound corresponding to the mixed volume of the Newton polytope corresponding to each equations [7, 45]. Homotopy methods use the solutions of a known system along with tracing algorithms to find the solutions of the given system. The tracing steps corresponds to Newton's iteration.

3 Collision Detection for Polyhedra

In this section, we summarize a simple and efficient collision detection algorithm for convex polyhedra by tracking the closest points between them [28]. We also extend to handle penetration between objects. The method works by finding and maintaining a pair of closest features (vertex, edge, or face) on the two convex polyhedra, in order to calculate the Euclidean distance between them to detect possible collision. The method is applicable in static environment, but is especially well suited to dynamic domains as the objects move in a sequence of small, discrete steps. We take advantage of the empirical fact that the closest features change only infrequently as the objects move along finely discretized paths. By preprocessing the polyhedra, the algorithm runs in *expected constant time* if the objects are not moving very swiftly. Even when the closest feature pair is changing rapidly, the algorithm takes only slightly longer (runtime is proportional to the number of feature pairs traversed which is a fuction of the relative motion objects undergo).

3.1 Preliminaries

We represent each convex polyhedron using modified boundary representation. Each polyhedron data structure has fields for its features (faces, edges, and vertices) and *Voronoi regions* (defined below).

Each feature (a vertex, an edge, or a face) is described by its geometric parameters and its neighboring features, i.e. the topological information of incidences and adjacencies. In addition, we also precompute the Voronoi region (definied below) for each feature as well.

Definition: A *Voronoi region* associated with each feature is a set of points closer to that feature than to any others. [41]

The Voronoi regions form a partition of space outside the polyhedron according to the closest feature. The collection of Voronoi regions of each polyhedron is the generalized Voronoi diagram of the polyhedron. Note that the generalized Voronoi diagram of a convex polyhedron has linear size and consists of polyhedral regions. A *cell* is the data structure for a Voronoi region. It has a set of constraint planes which bound the Voronoi region with pointers to the neighboring cells (which share a constraint plane with it) in its data structure. If a point lies on a constraint plane, then it is equi-distant from the two features which share this constraint plane in their Voronoi regions. Using the geometric properties of convex sets, applicability criteria (explained in Sec.3.3) are established based upon the Voronoi regions. If a point P on object A lies inside the Voronoi region of the feature f_B on object B, then f_B is a closest feature to the point P.

3.2 Overview

The distance computation algorithm which tracks the closest feature pair is more fully described in our earlier work [28] or [29]. Here we only give a general overview of the algorithm.

Our method is straightforward in its conception. We start with a candidate pair of features, one from each polyhedron, and check whether the closest points lie on these features. Since the objects (thereby the faces as well) are convex, this is a local test involving only the neighboring features of the current candidate features. If either feature fails the test, we step to a neighboring feature of one or both candidates, and try again. With some simple preprocessing, we can guarantee that every feature has a constant number of neighboring features. This is how we can verify or update the closest feature pair in constant time.

When a pair of features fails the test, the new pair we choose is guaranteed to be closer than the old one. So when the objects move and one of the closest features changes, we usually find it after one or two iterations. Even if the closest features are changing rapidly, say once per step along the path, our algorithm takes only slightly longer. In this case, the running time is proportional to the number of feature pairs traversed in this process. It is *not* more than the product of the numbers of features of the two polyhedra, because the Euclidean distance between feature pairs must always decrease when a switch is made [28], which makes cycling impossible.

3.3 Applicability Test

Given the overview of the algorithm, we now present a concise description of the main component of the algorithm — the applicability test — which is established based upon Voronoi region.

A Voronoi region, associated with each feature, is bounded by the constraint planes derived from the feature's neighbors. The applicability test is a simple checking process that verifies whether a point lies inside the Voronoi region of a given feature. With the preprocessing procedures to guarantee that every feature has a constant size of neighboring features, each applicability test is only a local test which runs in $O(1)$ time. When a point fails the applicability test of a given feature, the pointer associated with each constraint plane provides a new, closer feature which shares the same constraint plane with the previous cell.

For a given pair of features, $feat_A$ and $feat_B$, on objects A and B, we first find a pair of nearest points, P_A and P_B, between these two features. Then, we need to verify that $feat_B$ is truly the closest feature on B to P_A (i.e. P_A lies inside the Voronoi region of $feat_B$) and $feat_A$ is truly the closest feature on A to P_B (i.e. P_B satisfies the applicability test of $feat_A$). If either check fails, a new and closer feature (which is usually one of neighboring features of the previous features) is substituted, and the new pair of features is checked. Eventually, we must terminate with the closest pair, since the distance between the feature pair is strictly decreasing through each iteration.

This verifying process is demonstrated in Fig.1 where the previous closest feature candidate pair is (F_a, V_b), P_a and V_b are the two nearest points on these two given features. Though P_a satisfies the applicability of V_b (i.e. P_a lies inside of V_b's Voronoi region), V_b fails F_a's applicability test imposed by the constraint plane CP. Therefore, a new candidate pair (E_a, V_b), which is closer in distance, is returned. Then, the algorithm is called again upon (E_a, V_b) to verify whether this is the closest feature-pair iteratively, until the nearest points on two features both satisfy the applicability tests of each other. Then, the algorithm stops and returns the closest feature-pair. Fig.2 shows how this algorithm keeps track of all the closest feature pairs (indicated

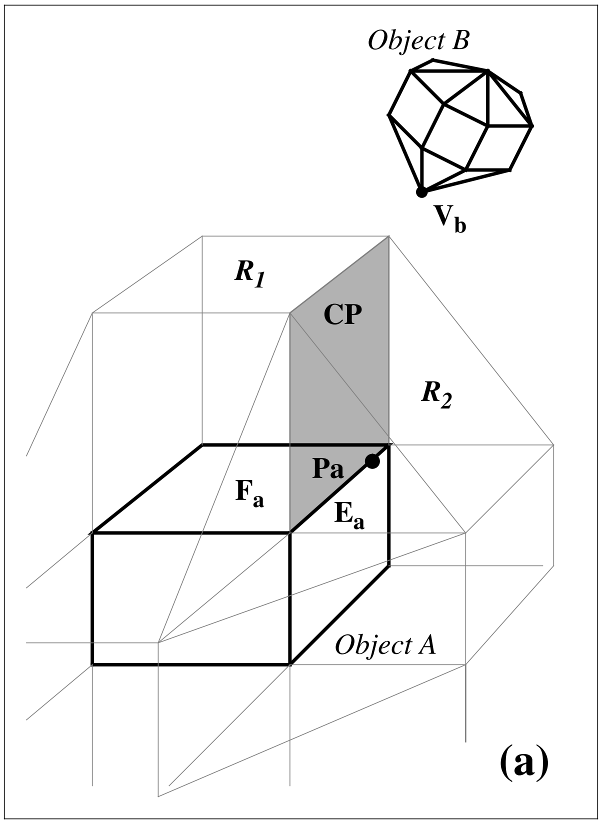

Figure 1: Applicability Test: R_1 and R_2 are the Voronoi regions of F_a and E_a respectively. $(F_a, V_b) \rightarrow (E_a, V_b)$ since V_b fails the applicability test imposed by the constraint plane CP.

Figure 2: Tracking Closest Feature Pairs

by silver white lines) among several moving convex polyhedral objects.

3.4 Subdivision Procedure

In general, each face of a typical convex polyhedron has four or five edges in its boundary and each vertex has three or four incident edges . When a face has more than 5 edges in its boundary or a vertex has more than 5 incident edges, its Voronoi region is preprocessed by subdividing the whole region into smaller cells. In Fig.3, R_a, the Voronoi region of V_a which has 8 incident edges, is subdivided into R_1, R_2, and R_3; R_b, the Voronoi region of F_b which has 8 edges in its boundary, is subdivided into R_4, R_5, and R_6. After subdivision, each of the new cells has only 3 or 4 constraint planes. This subdivision procedure is a simple linear time routine which can be done as a precomputation step. It guarantees that when the algorithm starts, each Voronoi cell has a constant number of constraint planes. Consequently, each applicability test described above runs in *constant time* for updating the closest feature pairs in a dynamic environment.

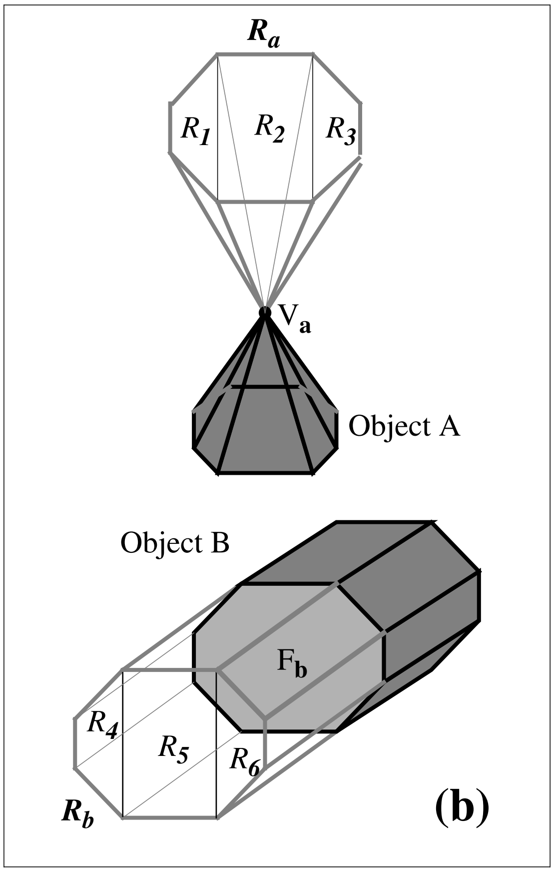

Figure 3: Preprocessing of a vertex's conical applicability region and a face's cylindrical applicability region

3.5 Implementation Issues

In order to minimize online computation time, we do all the subdivision procedures and build all the Voronoi regions first as one-time computational cost. We do *not* need to reconstruct all the Voronoi regions for each polyhedron as the objects move. We only need to transform the *candidate features* and recompute the nearest points, since *local* applicability constraints are all we need for tracking the closest pair of features. That is, we transform the point coordinates using *relative homogeneous transformation matrices* (available from either dynamic calculations or motion transformations as described in Sec 2.2) and leave constraint plane equations fixed. For example, given two objects moving in space, their motions with respect to the origin of the world frame can be characterized by the transformation matrices T_A and T_B respectively. Then, their relative motion can be represented by the homogeneous relative transformation $T_{AB} = T_B^{-1} T_A$.

This algorithm outputs the pair of closest features and also the distance between two objects. If the distance is less than or equal to δ (a small safety margin defined by user or determined by a given environment), then the objects collide.

3.6 Hierarchical Representation for Non-Convex Objects

We extend the collision detection algorithm for convex polyhedra to non-convex objects using hierarchical representation. We assume that each nonconvex object is given as a union of convex pieces *or* is composed of several nonconvex subparts, each of these can be further represented as a union of convex subparts or a union of non-convex pieces. We use a sub-part hierarchy tree to represent each nonconvex object (including curved objects which will be discussed later Fig.7). At each node of the tree, we store either a convex sub-part or the union of several convex subparts. First, we construct the convex hull for each node and work up the tree as part of preprocessing computation. We also include the convex hull of the union of sub-parts in the data structure. The convex hull of each node is the convex hull of the union of its children. For instance, The root of this sub-part hierarchy tree is the nonconvex object with its convex hull in its data structure.

At each time step, we examine the possible interference by a recursive algorithm. The algorithm first checks for collision between two parent convex hulls. Of course, if there is no interference between two parents, there is no collision and the algorithm stops and returns the closest feature pair between two convex hulls of the objects. If there is a collision, then it expands their children. If there is also a collision among the leaves, then the algorithm recursively proceeds down the tree to determine if a collision actually occurs. In this recursive manner, the algorithm only signals a collision if there is actually an impact among sub-parts of two objects; otherwise, there is no collision between the two objects.

For complex objects, using a deep hierarchy tree with lower branching factor will keep down the number of nodes which need to be expanded. This approach guarantees that we find the earliest collision between two non-convex objects.

3.7 Penetration Detection for Convex Polyhedra

The core of the collision detection algorithm is built upon the concepts of Voronoi regions for convex polyhedra. As mentioned earlier in Sec 3.2, the Voronoi regions form a partition of space outside the polyhedron. Therefore, the algorithm can possibly run into a cyclic loop when interpenetration occurs, if no special care is taken to prohibit such events. Hence, if the polyhedra can overlap, it is important that we add a module which detects interpenetration when it occurs as well.

- **Pseudo Voronoi Regions:**

This module can be constructed based upon similar ideas of space partitioning to the *interior* space of the convex polyhedra. The partitioning does not have to form the exact Voronoi regions since we are not interested in knowing the closest features between two interpenetrating polyhedra but only to detect such a case. A close approximation with simple calculation can provide the necessary tools for detecting overlapping.

This is done by partitioning the interior of a polyhedron. We first calculate the centroid of each convex polyhedron, which is the weighted average of all the vertices, and then construct the constraint planes of each face to the centroid of the polyhedron. These interior constraint planes of a face F are the hyperplanes passing through the centroid and each edge in F's boundary and the hyperplane containing the face F. If all the faces are equi-distant from the centroid, these hyperplanes form the exact Voronoi diagrams for the interior of the polyhedron. Otherwise, they will provide a reasonable space partition for the purpose of *detecting interpenetration*.

The data structure of these interior *pseudo* Voronoi regions is very much like the exterior Voronoi regions described in Sec 3.1. Each region associated with each face has $e+1$ hyperplanes defining it, where e is the number of edges in the face's boundary. Each hyperplane has a pointer directing to a neighboring region where the algorithm will step to next, if the constraint imposed by this hyperplane is violated. In addition, a *type* field is added in the data structure of a Voronoi cell to indicate whether it is an interior (pseudo) or exterior Voronoi region.

The exact Voronoi regions for the interior space of the polyhedron can also be constructed by computing all the equi-distant bisectors of all *facets*. But, such an elaborate precomputation is not necessary unless we are also interested in computing the degree of interpenetration for constructing some type of penalty functions in collision response or motion planning.

As two convex polyhedral objects move and pass through each other, the change of feature pairs also indicate such a motion since the pointers associated with each constraint plane keep track of the *pseudo* closest features between two objects during penetration phase as well. However, to ensure convergence, we will need to construct the exact Voronoi regions of the interior space or use special case analysis to guarantee that each switch of feature necessarily decreases the distance between candidate features.

4 Interference Detection for Curved Surfaces

In this section, we analyze the problem of interference detection between curved objects represented as spline models or piecewise algebraic surfaces. We show that these problems reduce to finding solutions of a system of algebraic equations. In particular, we present algebraic formulations corresponding to closest points determination and geometric contacts.

4.1 Closest Features Formulation

Given the homogeneous representation of two parametric surfaces, $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and $\mathbf{G}(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))$, the closest features correspond to points or curves on the surface. The closest features are characterized by the property that the corresponding surface normals are collinear. This can be expressed in terms of the following variables. Let

$$\begin{aligned}
\mathbf{F_{11}}(s,t,u,v,\alpha_1) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \\
\mathbf{F_{12}}(s,t,u,v,\alpha_1) &= \alpha_1(\mathbf{G}_u(u,v) \times \mathbf{G}_v(u,v)) \\
\mathbf{F_{21}}(s,t,u,v,\alpha_2) &= (\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t)) \\
\mathbf{F_{22}}(s,t,u,v,\alpha_2) &= \alpha_2(\mathbf{G}_u(u,v) \times \mathbf{G}_v(u,v)),
\end{aligned}$$

where $\mathbf{F}_s, \mathbf{F}_t, \mathbf{G}_u, \mathbf{G}_v$ correspond to the partial derivatives. The closest features between the two surfaces satisfy the following equations:

$$\mathbf{F_1}(s,t,u,v,\alpha_1,\alpha_2) = \mathbf{F_{11}}(s,t,u,v,\alpha_1) - \mathbf{F_{12}}(s,t,u,v,\alpha_1) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \qquad (3)$$

$$\mathbf{F_2}(s,t,u,v,\alpha_1,\alpha_2) = \mathbf{F_{21}}(s,t,u,v,\alpha_2) - \mathbf{F_{22}}(s,t,u,v,\alpha_2) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

This results in 6 equations in 6 unknowns. However, we are only interested in the solutions in the domain of interest (since each surface is defined on a subset of the real plane). These constraints between the closest features can also be expressed as:

$$\begin{aligned} \mathbf{H}_1(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{G}_u(u,v) = & 0 \\ \mathbf{H}_2(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{G}_v(u,v) = & 0 \\ \mathbf{H}_3(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{F}_s(s,t) = & 0 \\ \mathbf{H}_4(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{F}_t(s,t) = & 0 \end{aligned} \tag{4}$$

where $\bullet$ corresponds to the dot (or inner) product. This results in four equations in four unknowns.

Let us analyze the algebraic complexity of these two systems of equations corresponding to closest features. Lets consider the first system corresponding to (3). Given 2 rational parametric surfaces $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$, both of their numerators and denominators are polynomials of degree n. The numerator and denominator of $\mathbf{F_{11}}(s,t,u,v,\alpha_1)$ have degree $2n$ and $2n$ due to subtraction of two rational polynomials. As for $\mathbf{F_{12}}(s,t,u,v,\alpha_1)$, the degrees of numerators and denominators of the partials are $2n-1$ and $2n$ respectively in the given variables (due to quotient rule). Taking the cross product doubles the degrees for both the numerator and denominator; therefore, the degrees for the numerator and denominator of $\mathbf{F_{12}}(s,t,u,v,\alpha_1)$ are $4n-2$ and $4n$ respectively. To eliminate α_1 from $\mathbf{F}_1(s,t,u,v,\alpha_1)$, we get $\frac{\mathbf{F}_{11}(X(s,t,u,v,\alpha_1))}{\mathbf{F}_{12}(X(s,t,u,v,\alpha_1))} = \frac{\mathbf{F}_{11}(Y(s,t,u,v,\alpha_1))}{\mathbf{F}_{12}(Y(s,t,u,v,\alpha_1))} = \frac{\mathbf{F}_{11}(Z(s,t,u,v,\alpha_1))}{\mathbf{F}_{12}(Z(s,t,u,v,\alpha_1))}$. After cross multiplication and clearing out the denominators, we get two *polynomials* of degree $12n-2$ each. Once again, by the same reasoning as stated above for subtraction and dot product of rational polynomails, both the numerators and denominators of $\mathbf{F_{21}}(s,t,u,v,\alpha_2)$ and $\mathbf{F_{22}}(s,t,u,v,\alpha_2)$ have degrees of $4n-2$ and $4n$. By similar method mentioned above, we can eliminate α_2 from $\mathbf{F}_2(s,t,u,v,\alpha_2)$. We get two polynomial equations of degree $16n-4$ each after cross multiplication. As a result the system has a Bezout bound of $(12n-2)^2(16n-4)^2$.

Each equation in the second system of equations has degree $4n-1$ (obtained after computing the partials and subtraction of two rational polynomials) and therefore the overall algebraic complexity corresponding to the Bezout bound is $(4n-1)^4$. Since the later system 4 results in a lower degree bound, in the rest of the analysis we will use this system. However, we are only interested in the solutions in the domain of interest (since each surface is defined on a subset of the real plane).

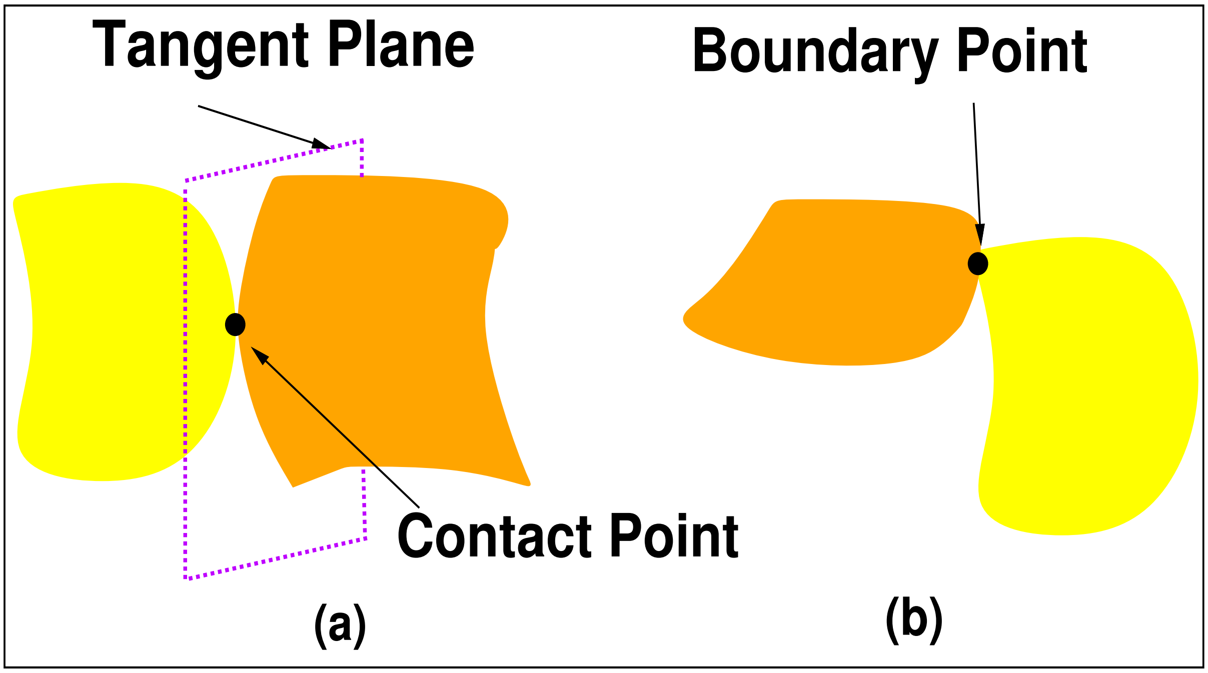

Figure 4: Tangential intersection and boundary intersection between two Bézier surfaces

Baraff has used the formulation in Eqn. 3 to keep track of closest points between closed convex surfaces [3] based on local optimization routines. The main problem is finding a solution to these equations (3) for the initial configuration. In general, these equations can have more than one solution in the associated domain (even though there is only one closest point pair) and the optimization routine may not converge to the right solution. A simple example is the formulation for the problem of interference detection between two spheres. There is only one pair of closest points, however Equations (4) have four pairs of real solutions.

Given two algebraic surfaces, $f(x, y, z) = 0$ and $g(x, y, z) = 0$, the problem of closest features determination can be reduced to finding roots of the following system of 6 algebraic equations:

$$\begin{aligned} f(x_1, y_1, z_1) &= 0 \\ g(x_2, y_2, z_2) &= 0 \\ \begin{pmatrix} f_x(x_1, y_1, z_1) \\ f_y(x_1, y_1, z_1) \\ f_z(x_1, y_1, z_1) \end{pmatrix} &= \alpha_1 \begin{pmatrix} g_x(x_2, y_2, z_2) \\ g_y(x_2, y_2, z_2) \\ g_z(x_2, y_2, z_2) \end{pmatrix} \end{aligned} \tag{5}$$

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \alpha_2 \begin{pmatrix} g_x(x_2, y_2, z_2) \\ g_y(x_2, y_2, z_2) \\ g_z(x_2, y_2, z_2) \end{pmatrix}$$

Given two algebraic surfaces of degree n, we can eliminate α_1 by setting $\frac{f_x(x_1,y_1,z_1)}{g_x(x_2,y_2,z_2)} = \frac{f_y(x_1,y_1,z_1)}{g_y(x_2,y_2,z_2)} = \frac{f_z(x_1,y_1,z_1)}{g_z(x_2,y_2,z_2)}$. After cross multiplication, we have a polynomial equation of $2n-2$, since each partial has degree of $n-1$ and the multiplication results in the degree sum of $2n-2$. Similarly, to eliminate α_2, we set $\frac{x_1-x_2}{g_x(x_2,y_2,z_2)} = \frac{y_1-y_2}{g_y(x_2,y_2,z_2)} = \frac{z_1-z_2}{g_z(x_2,y_2,z_2)}$ and the degree of the resulting polynomial equations is n. We have six quations after eliminating α_1 and α_2: two of degrees $(2n-2)$ and four of degrees n respectively (2 from eliminating α_2 and 2 from $f(x_1, y_1, z_1)$ and $g(x_2, y_2, z_2)$). Therefore, the Bezout bound of the resulting system can be as high as $N = (2n-2)^2n^4$. In general, if the system of equations is sparse, we can get a tight bound with Bernstein bound [5]. The Bernstein bound for Eqn. 5 is $n^2(n^2+3)(n-1)^2$. Canny and Emiris calculate the Bernstein bounds by using sparse mixed resultant formulation [7]. For example, the Bernstein bounds[1] for the case of $n = 2, 3, 4, 5, 6, 7, 8, 9$ are $28, 432, 2736, 11200, 35100, 91728, 210112, 435456$, while the Bezout bounds are $64, 1296, 9216, 40000, 129600, 345744, \cdots$ respectively. Even for small values of n, the bound on the number of solutions can be large, and therefore the algebraic complexity of computing the closest points can be extremely high, even using the exact tight bound. In our applications we are only interested in the real solutions to these equations in the corresponding domain of interest. The actual number of real solutions may change as the two objects undergo motion and some configurations can result in infinite solutions (e.g. when a closest pair corresponds to a curve on each surface, as shown for two cylinders in Fig. 5.). As a result, it is fairly non-trivial to keep track of all the closest features between objects and updating them as the objects undergo motion.

4.2 Contact Formulation

The problem of interference detection corresponds to determining whether there is any contact between the two objects. In particular, it is assumed that in the begining two objects are not overlapping. As they undergo motion, we are interested in knowing whether there is any precise contact between the objects. There are two types of contacts. They are tangential contact and boundary contact. In this section, we formulate both of these contact determination problems in terms of a system of

[1]These figures are calculated by John Canny and Ioannis Emiris using their code based on the sparse mixed resultant formulation [7].

algebraic equations. In the next section, we describe how the algorithm tests for these conditions as the object undergoes rigid motion.

- *Tangential Intersection :* This corresponds to a tangential intersection between the two surfaces at a geometric contact point, as in Fig.4(a). The contact point lies on the interior of each surface (as opposed to being on the *boundary* curve) and the normal vectors at that point are collinear. These constraints can be formulated as:

$$\begin{aligned} \mathbf{F}(s,t) &= \mathbf{G}(u,v) \\ (\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t)) \bullet \mathbf{G}_u(u,v) &= 0 \\ (\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t)) \bullet \mathbf{G}_v(u,v) &= 0 \end{aligned} \tag{6}$$

  The first vector equation corresponds to a contact between the two surfaces and the last two equations represent the fact that their normals are collinear. They are expressed as scalar triple product of the vectors. The last vector equation represented in terms of cross product corresponds to three scalar equations. We obtain 5 equations in 4 unknowns. This is an overconstrained system and has a solution only when the two surfaces are touching each other tangentially. However, we solve the problem by computing all the solutions to the first four equations using global methods and substitute them into the fifth equation. If the given equations have a common solution, than one of the solution of the first four equation will satisfy the fifth equation as well. For the first three equations, after cross multiplication we get 3 polynomial equations of degree $2n$ each. The dot product results in the addition of degrees of the numerator polynomials (each of these partials has numerator polynomial of degree $(2n-1)$). Therefore, we get a polynomial equation of degree $6n-3$ from the fourth equation. Therefore, the Bezout bound of the system corresponding to the first four equations is bounded by $N = (2n)^3(6n-3)$, where n is the parametric degree of each surface. Similarly for two algebraic surfaces, the problem of tangential intersection can be formulated as:

$$\begin{aligned} f(x,y,z) &= 0 \\ g(x,y,z) &= 0 \\ \begin{pmatrix} f_x(x,y,z) \\ f_y(x,y,z) \\ f_z(x,y,z) \end{pmatrix} &= \alpha \begin{pmatrix} g_x(x,y,z) \\ g_y(x,y,z) \\ g_z(x,y,z) \end{pmatrix} \end{aligned} \tag{7}$$

  In this case, we obtain 4 equations in 3 unknowns (after eliminating α) and these equations correspond to an overconstrained system as well. These over-

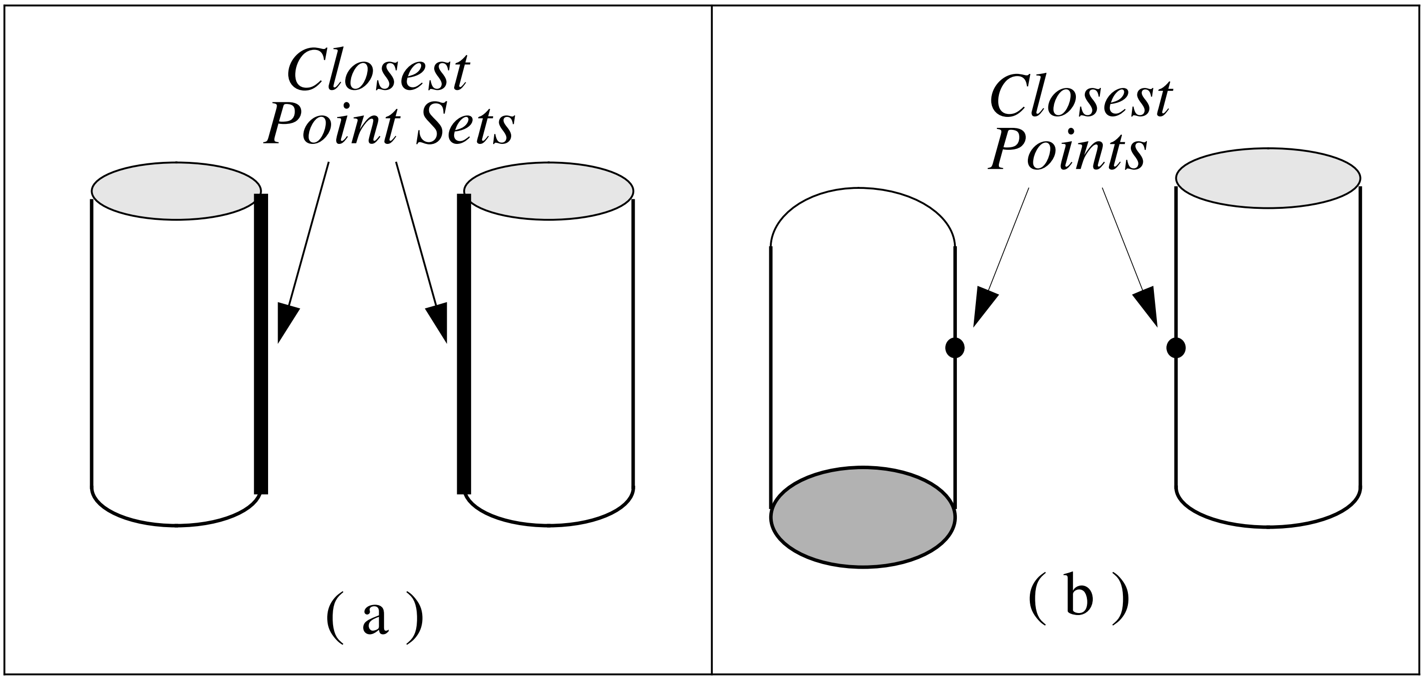

Figure 5: Closest features between two different orientations of a cylinder

constrained system is solved in a manner similar to that of parametric surfaces. The Bezout bound for the first three equations is $M = n^2(2n-2)$.

- *Boundary Intersection :* Such intersections lie on the boundary curve of one of the two surfaces. Say we are given a Bézier surface, defined over the domain, $(s,t) \in [0,1] \times [0,1]$, we obtain the boundary curves by substituting s or t to be 0 or 1. The resulting problem reduces to solving the equations:

$$\mathbf{F}(s,1) = \mathbf{G}(u,v) \tag{8}$$

  Other possible boundary intersections can be computed in a similar manner. The intersection points can be easily computed using global methods. An example has been shown in Figure 4(b)

Two objects have a contact if one of these sets of equations, ((6) or (8)) for parametric surfaces and (7) for algebraic surfaces, have a common solution in their domain.

In a few degenerate cases, it is possible that the system of equations (6) and (8) have an infinite number of solutions. One such example is shown for two cylinders in Fig.5. In this case the geometric contact corresponds to a curve on each surface, as opposed to a point. These cases can be detected using resultant methods as well [32].

4.3 Penetration Analysis

In most examples, it is difficult to express the motion as a closed form function of time. As a result, it is almost impossible for the algorithm to compute the exact time to collision as at instance we only have the information about the object's position and geometry. Based on the algorithm for polyhedral models and curved models (presented in the next section) it is possible for us to know that the two objects are getting closer. As a result, the main routine of the system can check for interference detection at shorter time steps. However, it is possible that at a given time instance the two objects are not colliding and at the next instance they have penetrated into each other. The algebraic constraints for contact analysis, based on tangential contact or boundary contact, are satisfied only if the two objects have a tangential or boundary contact. They may no longer be satisfied, when the two objects have penetrated. In this section, we formulate conditions for penetration between curved models and how they can be used for collision detection.

Two objects are penetrating, if there is a finite area of contact between them (as shown in Fig. 6). In contact analysis, we have assumed that there is a point contact (or a higher order contact along the boundary or on the tangent plane). The penetration therefore results in a one-dimensional intersection curve corresponding to the intersection of the two surface boundaries. In this section, we show that there is still a real solution to the subset of equations corresponding to (6), (8) or (7).

Lets consider boundary intersections as in Fig. 6(b). Lets assume that there was a contact along the boundary curve corresponding to $\mathbf{F}(s,1)$ and at the given instance the two surfaces have penetrated. As a result, if we compute the common solutions of the equations:

$$\mathbf{F}(s,1) = \mathbf{G}(u,v) \tag{9}$$

there is a real solution in the corresponding domain, which may not correspond to the boundary curve of $\mathbf{G}(u,v)$ or there are more than one distinct solutions to these equations. However, any small perturbation along the boundary curve $\mathbf{F}(s,1)$ would result in a solution to these equations in the corresponding domain and based on that we can detect interference.

Now, lets consider the tangential contact between two algebraic surfaces as in Fig. 6(a). Similar analysis would apply to the tangential intersection of piecewise parametric surfaces as well. To check for tangential contact, we use the equation solving methods to compute the solutions of the following four equations in four unknowns:

$$\begin{aligned} f(x,y,z) &= 0 \\ g(x,y,z) &= 0 \end{aligned} \tag{10}$$

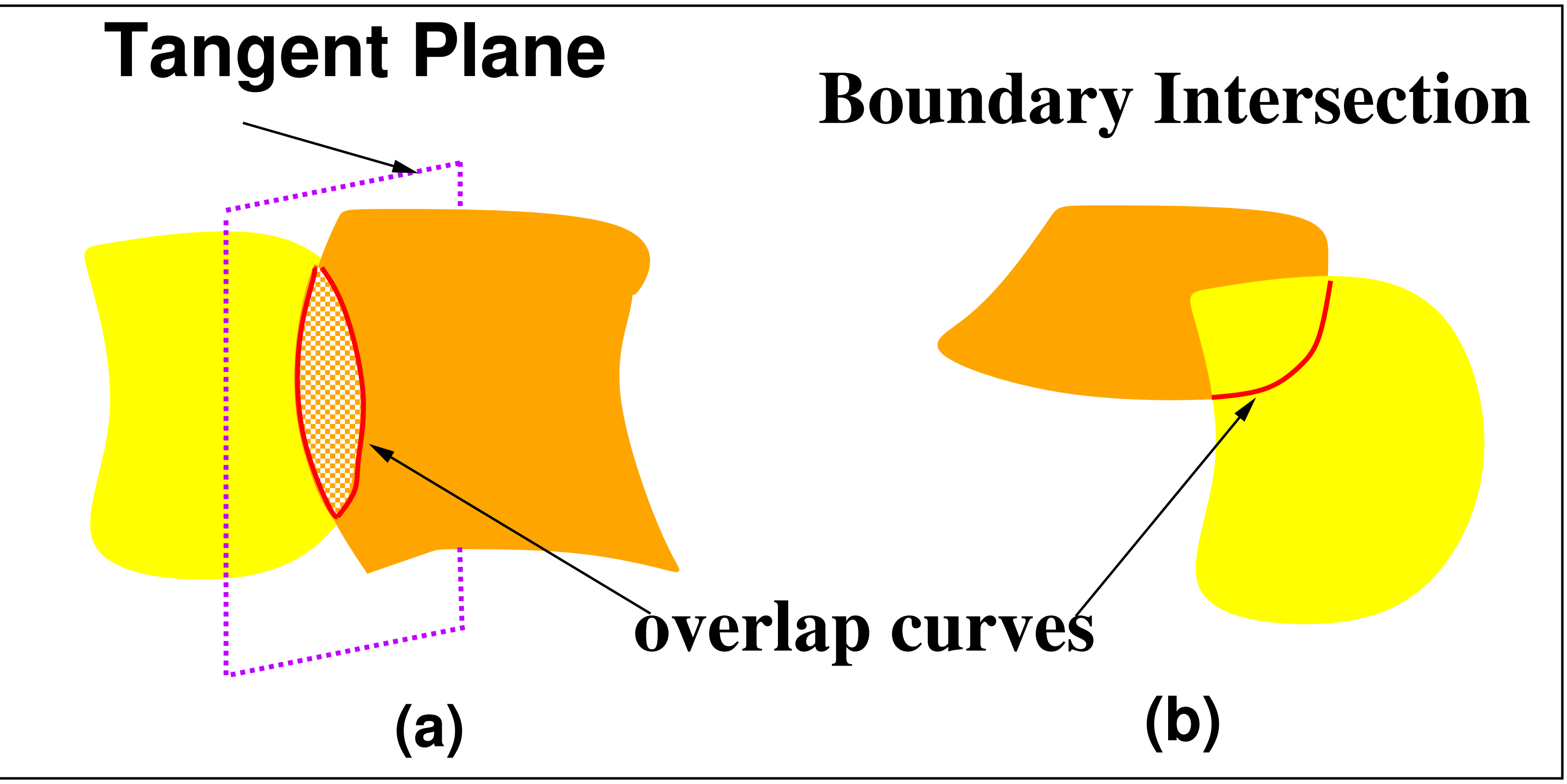

Figure 6: (a) penetration resulted from tangential intersection; (b) overlap resulted from boundary intersection

$$\begin{pmatrix} f_x(x,y,z) \\ f_y(x,y,z) \end{pmatrix} = \alpha \begin{pmatrix} g_x(x,y,z) \\ g_y(x,y,z) \end{pmatrix}.$$

The solutions are substituted into the fifth equation, as shown in (6) to check for contact analysis. The penetration information is obtained based on following theorem. Lets assume that the intersection between the two surfaces is *non-planar*.

Theorem 4.1 *Two algebraic surfaces, $f(x,y,z) = 0$ and $g(x,y,z) = 0$ have a non-planar interpenetration, iff there is at least one real solution of the system of equations (10).*

Proof: Lets take the case when the two objects have no geometric contact between them. As a result, there is no common real solution to the first two equations, $f(x,y,z) = 0$ and $g(x,y,z) = 0$.

Lets consider the case, when the objects penetrate. Moreover, the intersection is non-planar. As a result, the two surfaces intersect along a one dimensional space curve. In a few degenerate cases, the intersection curve is higher dimensional. Lets denote the real curve of intersection as C. C is an algebraic space curve corresponding to all the common solutions of $f(x,y,z) = 0$ and $g(x,y,z) = 0$. Typically, C consists of one closed component in space (as shown in Fig.6). In some cases, corresponding to non-convex models it may consist of multiple components. Eliminating α from the last two equations of (10) results in

$$G(x,y,z) = f_x(x,y,z)g_y(x,y,z) - f_y(x,y,z)g_x(x,y,z) = 0.$$

The normal at a point on the intersection curve is obtained by taking the cross product of the two surfaces at that point. In other points, the normal function is defined as:

$$N(x,y,z) = \begin{pmatrix} f_x(x,y,z) \\ f_y(x,y,z) \\ f_z(x,y,z) \end{pmatrix} \times \begin{pmatrix} g_x(x,y,z) \\ g_y(x,y,z) \\ g_z(x,y,z) \end{pmatrix}.$$

It follows from this definition that $G(x,y,z)$ corresponds to the Z-component of the normal vector of the intersection curve. As a result, the real solutions of the system of equation (10) correspond to all those points on the intersection curve where the Z-component of the normal vector is zero. Since C consists of non-planar closed loops, there are at least two such points on each component of C. As a result, the given system of equations have a real solution. Q.E.D.

The penetration criterion highlighted in Theorem 4.1 assumes that the intersection curve is non-planar. This is typically the case for most models (unless one of them is a plane or if the two models correspond to quadric surface). In fact, the non-planarity condition for penetration can be relaxed and it is required that the curve should have a point where the Z-component of the normal vector is zero. This condition is not satisfied, if the intersection curve lies in a plane parallel to the XY-plane. To circumvent this case, we take three generic linear combinations of the last three equations highlighted in (7) and pick two of the resulting three equations along with $f(x, y, z) = 0$ and $g(x, y, z) = 0$ to formulate a new system. For almost all linear combinations the penetration condition highlighted in Theorem 4.1 is exact. Similar analysis can be applied to the tangential intersection of two parametric surfaces. In case the two models penetrate and there is no solution for the equations corresponding to boundary intersection in the given domain, there is a real solution to the first five equations highlighted in (6) in the given domain. Again this may only fail if the intersection curve is planar and there is no point where the Z-component of the normal vector vanishes. To improve the robustness of the approach we replace the last three equations in (6) by their generic linear combinations.

5 Coherence for interference detection between curved objects

In most dynamic environments, the closest features between two moving objects change infrequently between two time frames. We have used this coherence property, utilizing spatial as well as temporal coherence, in designing the expected constant time algorithm for collision detection among convex polytopes and extended to non-convex polytopes as well. In this section we utilize this coherence property along with the algebraic formulations presented in the previous section for contact determination between curved models.

the total number of faces resulting from the polygonization.

Given two B-spline models, we decompose them into a series of Bézier surfaces using knot insertion algorithm [19]. After decomposition we use a hierarchical representation for the curved models. The height of the resulting tree is two. Each leaf node corresponds to the Bézier surface. The nodes at the first level of the tree correspond to the convex hull of the control polytope of each Bézier surface. The root of the tree represents the union of the convex hull of all these polytopes. The torus is composed of biquadratic Bézier surfaces, shown at the leaf nodes (Fig. 7). The convex polytopes at the first level are the convex hull of control polytopes of each Bézier surface and the root is a convex hull of the union of all control points.

The algorithm proceeds on the hierarchical representation, as explained in Section

Figure 7: Hierarchical representation of a torus composed of Bézier surfaces

3. However, each leaf node is a Bézier surface. The fact that each surface is non-convex implies that the closest features of the polytopes may not be a good approximation to the closest points on the surface. Moreover, two such surfaces can have more than one closest feature at any time. Given two algebraic surfaces, we intially enclose them in a bounding box and test the bounding boxes for collision. At the leaf nodes, we check the two models for precise contact, once their control polytopes collide.

The problem of collision detection between two Bézier surfaces is solved by finding all the solutions to the equations (6) and (8). If the two models are described implicitly, the first collision corresponds to the solution of (7). A real solution in the domain to those equations implies a geometric collision and a precise contact between the models. The algebraic method based on resultants and eigenvalues is used to find all the solutions to the equations (6),(8) or (7) [32]. This global root finder is used when the control polytopes of two Bézier surfaces collide. At that instant the two surfaces may or may not have a geometric contact. It is possible that all the solutions to these equations are complex. The set of equations in (6) represents an overconstraint system and may have no solution in the complex domain as well. However, we apply the algebraic method to the first four equations in (6) and compute all the solutions.

The total number of solutions of a system of equations is bounded by the Bezout bound. The resultant method computes all these solutions. As the objects move, we update the coefficients of these equations based on the rigid motion. We obtain a new set of equations corresponding to (6) and (8), whose coefficients are slightly different as compared to the previous set. All the roots of the new set of equations are updated using Gauss-Newton's method. The previous set of roots are used as initial guesses. Gauss-Newton's method works well, if the relative motion between the two objects is small. This approach is similar to *homotopy methods* for tracing paths to compute solutions of a given system of polynomial equations. In case the Gauss-Newton's method does not converge to a solution, we may again use the problem in terms of an eigenvalue problem and use the previous eigenvalues as a guess and use *inverse power iteration*. This approach has been explained in detail in [32]. The convergence of inverse power iteration is well understood and it converges to the eigenvalue closest to the guess. As a result, we are performing local computations at each step corresponding to a few Gauss-Newton's iterations or inverse power iterations. This procedure represents an algebraic analog of the geometric coherence exploited in the earlier section.

Typically, if the objects have a tangential contact and as the two objects are moving closer to each other, the imaginary components of some of the roots start decreasing. This involves the use of complex arithmetic while updating the solution with Gauss-Newton's method or inverse power iterations. Finally, a real collision occurs, when the imaginary component of one of the roots becomes zero. A real solution of the resulting system implies a collision. In case it satisfies all the equations simultaneously, there is a tangential contact otherwise there is penetration. The objects have a boundary intersection, if the equations corresponding to boundary intersection have a real solution, which is not in the domain. As the objects undergo motion, the solution moves and based on the roots of the given system of equations, we can determine whether there is a boundary contact or penetration.

It is possible for the objects to have *multiple contacts*. In such cases two distinct sets of solution paths converge to real solutions in the corresponding domain. As a result, the algorithm can easily find all the multiple point of contacts.

The performance of the algorithm is dependent on the complexity of the global root finder and the number of paths we need to trace. The total number of paths correspond to the Bezout bound of the given system and can be fairly high for a given system of equations. Tracing each path at each instance can be time consuming. The choice of paths is made based on the solutions of the global equation solver. In most applications, the global root finder is used when the two objects are fairly close to each other. This happens when the bounding boxes corresponding to control polytopes overlap. As a result, we only choose those solutions as the start paths whose imaginary components are relatively small or whose real solutions are close to the corresponding domain. This is a function of the degree of the resulting surfaces. For higher degree models, we pick initial solutions with high imaginary components as well.

Example 5.1 *Lets illustrate the algorithm on two torii undergoing motion and eventually colliding at a point. We assume that each torus is enclosed by a bounding box. Such a bounding box can either be obtained using representation of the torus as a union of rational Bézier surfaces (taking the convex hull of associated control points) or from the geometric definition. In particular, a torus can be described as a union of points equidistant from a circle in space. As a result, a torus can be described in terms of a circle C in 3 space and a radius r. The bounding box is computed by enclosing the circle C with the square of minimum area, say S, and its height is r along each side. A canonical representation of the torus, assuming a circle of radius k centered at the origin is*

$$F(x,y,z) = (\sqrt{x^2+y^2} - k)^2 + z^2 - r = 0.$$

In Figure 8, we have shown two torus moving towards each other. The initial position of the first torus is described using equation

$$F(x,y,z) = 64 - 20x^2 + x^4 - 20y^2 + 2x^2y^2 + y^4 + 16z^2 + 2x^2z^2 + 2y^2z^2 + z^4$$

and the initial configuration of the second torus corresponds to

$$G(x,y,z) = 392.0625 - 291x + 84.5x^2 - 12x^3 + x^4 + 32.5y^2 - 12xy^2 + 2x^2y^2 +$$

$$y^4 - 227.5z + 84xz - 14x^2z - 14y^2z + 81.5z^2 - 12xz^2 + 2x^2z^2 + 2y^2z^2 - 14z^3 + z^4$$

To check for contact we add the following three equations corresponding to the first order differentials of $F(x,y,z)$ and $G(x,y,z)$:

$$\begin{pmatrix} -40x + 4x^3 + 4xy^2 + 4xz^2 \\ -40y + 4x^2y + 4y^3 + 4yz^2 \\ 32z + 4x^2z + 4y^2z + 4z^3 \end{pmatrix} = \alpha \begin{pmatrix} -291 + 169x - 36x^2 + 4x^3 - 12y^2 + 4xy^2 + 84z - 28.xz - 12z^2 + 4xz^2 \\ 65y - 24xy + 4x^2y + 4y^3 - 28yz + 4yz^2 \\ -227.5 + 84x - 14x^2 - 14y^2 + 163z - 24xz + 4x^2z + 4y^2z - 42z^2 + 4z^3 \end{pmatrix},$$

To solve for the equations, we take linear combinations of these three equations, eliminate α and reduce the problem of contact determination to a system of 4 equations in 3 unknowns. As the objects undergo motion, these system of equations is updated in the following manner. Let the rigid motion of the first object be represented as

$$\begin{pmatrix} \overline{x} \\ \overline{y} \\ \overline{z} \end{pmatrix} = R_1 \begin{pmatrix} x \\ y \\ z \end{pmatrix} + T_1,$$

where R_1 is an orthogonal matrix corresponding to the rotation and T_1 corresponds to the translation. This can be expressed as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_1^T (\begin{pmatrix} \overline{x} \\ \overline{y} \\ \overline{z} \end{pmatrix} - T_1)$$

The corresponding position of the new object is obtained by substituting for (x,y,z) in $F(x,y,z)$ and obtaining the equation $\overline{F}(\overline{x},\overline{y},\overline{z}) = 0$ corresponding to the position of the torus after undergoing motion. Similarly, we compute a new representation $\overline{G}(\overline{x},\overline{y},\overline{z}) == 0$ for the second torus. Notice that, it undergoes a motion represented by R_2 and T_2. Given these equations, we formulate the problem of contact determination by taking their differentials and taking the linear combinations of the resulting equations. The solution of the previous set of equations are used as the starting guesses. This process is repeated until there is a contact or penetration between the two objects or their bounding boxes do not overlap anymore.

6 Interference Tests for Multiple Objects

In a typical environment, there are moving objects and stationary obstacles as well. Assume that there are N objects moving around in an environment of M stationary obstacles. Each of N moving objects is also considered as a moving obstacle to the other moving objects. Keeping track of the distance for $\binom{N}{2} + NM$ pairs of objects at all time can be quite time consuming, especially in a large environment. In order to avoid unnecessary computations and to speed up the run time, we use two methods: one assumes the knowledge of maximum acceleration and velocity, the other purely exploits the spatial arrangement without any other information to reduce the number of pairwise interference tests. For the scheduling scheme [28] in which we assume the dynamics (velocities, accelerations, etc.) is known, we rely on the fact that only the object pairs which have a small separation are likely to have an impact within the next few time instances, and those object pairs which are far apart from each other cannot possibly come to interfere with each other until certain time. If the information of velocity and acceleration is not available, then we fall back on some well-known algorithms for rectangular intersections [17, 46]. For more details on scheduling scheme which uses a priority queue based on lower bound on time to collision, please refer to [28]. Here we will only give a brief survey and comments on several intersection algorithms for isothetic bounding boxes and spheres, based on our experiences.

6.1 Using Bounding Volumes

In the three-dimensional workspace, if two bodies collide then their projections down to the lower-dimensional $x-y, y-z, x-z$ hyperplanes must overlap as well. Therefore, if we can efficiently *update* their overlapping status in each axis or in a 2-dimensional plane, we can easily eliminate the object pairs which are definitely not in contact with each other. In order to quickly determine all object pairs overlapping in the lower dimensions, we impose a virtual bounding box hierarchy on each body.

To compute exact collision contacts, using a bounding volume for an interference test is not sufficient, but it is rather efficient for eliminating those object pairs which are of no immediate interests from the point of collision *detection*. The bounding box can either be spherical or rectangular (even elliptical), depending on the application and the environment. We prefer spherical and rectangular (or cubic) volume due to their simplicity and suitability in our application.

Consider an environment where most of objects are elongated and only a few objects (probably just the robot manipulators in most situations) are moving, then rectangular bounding boxes are preferable. In a more dynamic environment like a vibrating parts feeder where all objects are rather "fat" [38] and bouncing around, then spherical or cubic bounding boxes are more desirable. If the objects are non-convex or articulated, then a subpart-hierarchical bounding box representation (similar to subpart-hierarchical tree representation, with each node storing a bounding box) should be employed.

6.2 Intersection Tests for Rectangular and Spherical Bounding Boxes

In earlier literature, there are several algorithms which can solve the overlapping problem for d-dimensional rectangular bounding boxes in $O(nlog^{d-1}n+k)$ time, where n is the number of objects and k is the number of pairwise overlaps by Edelsbrunner, Six and Woods [17, 46]. For spherical bounding boxes, we can use the algorithm by Hopcroft, Schwartz and Sharir in [26], which runs in $O(n^2logn)$ time.

6.3 Other Approaches

Here we will also briefly mention a few different approaches which we have considered.

We can divide the space into unit cells (or volumes) and place each object (or bounding box) in some cell(s) [6, 30]. To check for collisions, we have to examine the cell(s) occupied by each box to verify if the cell(s) is(are) shared by other objects. But, it is difficult to set a near-optimal size for each cell and it requires tremendous amount of allocated memory. If the size of the cell is not properly chosen, the computation can be rather expensive. For an environment where almost all objects are of uniform size, like a vibrating parts feeder bowl or molecular modeling [30, 47], this is a rather ideal algorithm, especially to run on a parallel-computing machine.

Edahiro and etc. [18] proposed a $O(k)$ search time, $O(n)$ preprocessing time approach for the reporting all segment intersection with a given orthogonal query segment, where n is the number of orthogonal segments and k is the number of reported intersecting segments. Although this approach cannot be applied directly to our problem, it indicates the promises for good average running time performance of the modified bucketing algorithm for $3d$ intersection tests.

In fact, Overmars has shown that using a hash table to look up an entry and variable levels (sizes) of cubes or cells, we can use a data structure of $O(n)$ storage space to perform the point location queries in constant time [38]. This approach works well if the objects are not intersecting, since the assumption behind the basic algorithm depends on the fact that each cube or cell can only be occupied by a fixed number of objects. If the objects are allowed to interpenetrate each other, this assumption can no longer hold. Other mechanism must be used to detect interpenetration. One possibility is to "grow" the objects by a small amount for the "first cut" intersection test, then perform exact collision detection.

7 Implementation and Performance

We have implemented the algorithms described in this paper and tested their performance on many cases. The collision detection routine for convex polytopes is very efficient and gives us a constant time performance of 50 to 70 usec (x 10^{-6} sec) per pair of object independent of complexity on SGI Indigo2 XZ. Fig.2 highlights an instance of five polytopes with all the closest features. Each polytope consists of about 25 faces. The hierarchical approach works well for concave objects, given a good convex decomposition of the object. We are able to perform collision detection in *real time* using our polyhedral collision detection algorithm.

Using the spatial and temporal coherence, we can keep track of the closest feature pairs between convex curved objects and the control polytopes of non-convex curved objects efficiently as well. In Fig.8(a) - 8(f) (from left to right, top to bottom) we demonstrate the algorithm on two torus. Each torus is represented hierarchically as shown in Fig.7. In particular, Fig.8(c) and Fig.8(d) represent the same time instant. In Fig.8(c) the convex hulls of two tori collide but in Fig.8(d) the control polytopes of the subparts do not touch each other. Once two control polytopes of the suparts also collide Fig.8(e), we use the global methods to find the initial solutions of (6) and (8) and followed by Gauss-Newton method to update the solutions at each time instance. It involves complex arithmetic. Although the hierarchical representation is in terms of biquadratic Bézier patches, we use the implicit representation of a torus (as that results in a system with lower Bezout bound). Each torus is represented as an algebraic surface of degree 4. Its equation is obtained by squaring the expressions in

$$F(x, y, z) = (\sqrt{x^2 + y^2} - k)^2 + z^2 - r = 0.$$

Since the torus is a closed object, there are no boundary intersections and we keep track of tangential intersections for geometric contacts. The resulting system of equations corresponding to (6) are of degrees 4, 4 and 6. The Bezout bound of this system is 96. Using resultants, the problem is reduced to finding the eigendecomposition of a 96×96 matrix [32]. The eigendecomposition takes slightly more than a second on the IBM RS/6000 and all the solutions at that instant are complex. As the torus move towards each other, we eventually track 8 of these solutions, whose imaginary parts are decreasing. This uses Gauss-Newton's method at each time instance. Finally a real solution is obtained corresponding to the instance shown in Fig.8(f).

8 Conclusion

In this paper we have described efficient algorithms for contact determination in dynamic environments. The algorithms are based on spatial and temporal coherence between successive instances and applied to polyheral models and whose boundaries can be desribed as piecewise algebraic sets. This include NURBS surfaces commonly used in computer graphics and geometric modeling. The resulting algorithm make use of hierarchical representations along with local and global methods for equation solving. These algorithms have been implemented and work well in practice.

Many issues related to contact determination for general dynamic environments still remain open. In an environment with N objects, we want to avoid the $O(N^2)$ pairs testing at each instance. Some techniques to speed them up based on spatial subdivision are proposed in [47] and algorithms of complexity $O(N log^2 N)$ have been highlighted for spheres in [26] and axis-aligned bounding boxes in [17]. However, they are for static environments and donot make use of coherence for dynamic environments. Furthermore, there are no good algorithms for contact determination between deformable models.

Acknowledgement:

We wish to thank Professor John Canny for his comments on the draft of this paper. We like to acknowledge Brian Mirtich for his help in re-implementing parts of the distance computation algorithm for convex polyhedra in ANSI C [2]. We are also grateful to Ransom Murphy and Krish Ponamgi for integrating parts of collision detection code on Pixel Planes. Figure 2 is from their implementations. We are also thankful to Anselmo Lastra for his assistance in generating color pictures.

[2]The original algorithm was first written in Sun Common Lisp by the first author

Figure 8: Collision detection algorithm applied to two tori

References

[1] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM J. Comput.*, 19:297–321, 1990.

[2] P. K. Agarwal, M. van Kreveld, and M. Overmars. Intersection queries for curved objects. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 41–50, 1991.

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

[4] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *ACM Computer Graphics*, 22(4):31–39, 1988.

[5] D. N. Bernstein. The number of roots of a system of equations. *Funktsional'nyi Analiz i Ego Prilozheniya*, 9(3):1–4, Jul-Sep 1975.

[6] J. L. Bentley and J. H. Friedman. Data structures for range searching. *Computing Surveys*, 11(4), December 1979.

[7] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAECC*, 1993.

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

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

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

[11] D. P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *J. Algorithms*, 8:348–361, 1987.

[12] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27:241–253, 1983.

[13] 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 in Computer Science*, pages 400–413. Springer-Verlag, 1990.

[14] D. P. Dobkin and D. L. Souvaine. Detecting the intersection of convex objects in the plane. *Comput. Aided Geom. Design*, 8:181–200, 1991.

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

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

[17] H. Edelsbrunner. A new approach to rectangle intersections, Part I. *Internat. J. Comput. Math.*, 13:209–219, 1983.

[18] M. Edahiro, K. Tanaka, R. Hoshino, and Ta. Asano. A bucketing algorithm for the orthogonal segment intersection search problem and its practical efficiency. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 258–267, 1987.

[19] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1990.

[20] C.B. Garcia and W.I. Zangwill. Finding all solutions to polynomial systems and other systems of equations. *Math. Prog.*, 16:159–176, 1979.

[21] 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.

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

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

[24] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[25] M.E. Hohmeyer. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1(4):473–490, 1991. Special issue on Solid Modeling.

[26] 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.

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

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

[29] M.C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *IEEE Conference on Robotics and Automation*, volume 2, pages 1008–1014, 1991.

[30] C. Leventhal. Molecular model-building by computer. *Scientific American*, 214(6), June 1966.

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

[32] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[33] D. Manocha and J. F. Canny. A new approach for surface intersection. *Internat. J. Comput. Geom. Appl.*, 1(4):491–516, 1991.

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

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

[36] D. M. Mount. Intersection detection and separators for simple polygons. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 303–311, 1992.

[37] B.K. Natarajan. On computing the intersection of b-splines. In *ACM Symposium on Computationl Geometry*, pages 157–167, 1990.

[38] M. H. Overmars. Point location in fat subdivisions. *Inform. Proc. Lett.*, 44:261–265, 1992.

[39] A. Pentland. Computational complexity versus simulated environment. *Computer Graphics*, 22(2):185–192, 1990.

[40] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):185–192, 1990.

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

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

[43] M. Sharir. Efficient algorithms for planning purely translational collision-free motion in two and three dimensions. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1326–1331, 1987.

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

[45] B. Sturmfels. Sparse elimination theory. In D. Eisenbud and L. Robbiano, editors, *Computational Algebraic Geometry and Commutative Algebra*. Cambridge University Press, 1991.

[46] H.W. Six and D. Wood Counting and reporting intersections in d-Ranges. *IEEE Trans. on Computers*, C-31(No. 3), March 1982.

[47] G. Turk. Interactive collision detection for molecular graphics. Master's thesis, Computer Science Department, University of North Carolina at Chapel Hill, 1989.