Fast Contact Determination in Dynamic Environments

Ming C. Lin* Univ. of Calif., Berkeley
Dinesh Manocha† Univ. of No. Carolina, Chapel Hill
John Canny‡ Univ. of Calif., Berkeley

Abstract

We present an efficient contact determination algorithm for objects undergoing rigid motion. The environment consists of polytopes and models described by algebraic sets. We extend an expected constant time collision detection algorithm between convex polytopes to concave polytopes and curved models. The algorithm makes use of hierarchical representations for concave polytopes and local, global methods for solving polynomial equations to determine possible contact points. We also propose techniques to reduce $O(n^2)$ pairwise intersection tests for a large environment of n objects. These algorithms work well in practice and give real time performance for most environments.

1 Introduction

An essential component of robot motion planning and collision avoidance is a geometric reasoning system which can detect potential contacts and determine the exact collision points between the robot manipulator and the obstacles in the workspace. Although it doesn't provide a complete solution to the path planning and obstacle avoidance problems, it often serves as a good indicator to steer the robot away from its surrounding before an actual collision occurs.

The problems of contact determination or interference detection between two or more objects has been extensively studied in robotics and computational geometry [4, 5, 6, 8, 14, 20, 22, 23]. Besides these areas, it has a received a lot of attention in computer animation, molecular modeling and computer simulated environments as well [1, 2].

In this paper, we present an efficient algorithm for interference detection between polyhedral and curved objects undergoing rigid motion. The algorithm makes use of spatial and temporal coherence between successive instances and no assumption is made on the motion of the object to be expressed as a closed form function of time. The coherence is based on local features of the polytopes and algebraic formulation of contact determination between curved models. For a large environment of n objects, we present techniques to reduce the $O(n^2)$ pairwise detections as well. These algorithms have not been unsuccessfully applied to any test environment and they give a real time performance in most cases.

1.1 Previous Work

Most of the earlier work in robotics and computational geometry has focussed on collision detection between convex polytopes. Algorithms in computational geometry use properties of convex sets and reduce the problem to linear programming. Good theoretical and practical algorithms based on linear complexity of the linear programming problem in fixed dimension are known as well [25, 28]. Algorithms with expected linear time performance are also given in [14] to compute the distance between convex polytopes and keep track of closest features. Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [8] for polyhedron-polyhedron intersection 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 [10, 17, 30]. They make use of subdivision methods or interval arithmetic, bounds on derivatives and constrained minimization. However, they are fairly slow in practice and not applicable for most robotics environments.

In applications involving rigid motion, geometric coherence has been utilized to devise algorithms for convex polyhedra based on local features [1, 22, 21]. Local properties have been used in the earlier motion planning algorithms by [9, 23] 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 $O(nlog^2 n + m)$ have been presented for spheres in [19] and rectangular bounding boxes in [11], where m corresponds to the number of overlaps. Overmars has shown that using a hash table to look up an enetry, we can use a data structure of $O(n)$ storage space to perform the point location queries in constant time [27].

1.2 Organization

The rest of the paper is organized in the following manner. In Section 2 we review literature on object models and solving polynomial equations. Section 3 highlights the constant time algorithm by Lin and Canny (details can be found in [22, 21]) for convex polytopes and its extension to concave objects. In Section 4, we analyze the

*Currently an assist. prof. at Naval Postgraduate School.
†Supported in part by Junior Faculty Award and by DARPA Cont. # DAEA-18-90-C-0044.
‡Supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).

1050-4729/94 $03.00 © 1994 IEEE

problem of contact determination between spline surfaces and algebraic models. In Section 5, we use local methods of solving equations along with polyhedral approximations to convex curved models for collision detection. Concave surfaces are handled using polyhedral bounding boxes, local and global methods for solving polynomial systems. A scheduling scheme and a hybrid algorithm to overcome the $O(n^2)$ bottleneck for large environments is presented in Section 6. Finally, in Section 7 we discuss the performance and implementation of the algorithm to various environments.

2 Background

2.1 Object Models

Most of the earlier robotics simulation systems have benn restricted to polyhedral models. However, modeling with surfaces bounded by linear boundaries poses a serious restriction in these systems. Model descriptions of common robots like the PUMA consist of quadric surfaces like cylinders and hemi-spheres. In this paper, robot models and environments are described in terms of *algebraic sets*. Mathematically, algebraic sets correspond to zeros of a system of polynomial equations. Besides polyhedral models, they include quadric surfaces, NURBS representations (Bézier and B-spline patches) used in geometric and solid modeling [18] and combinations of these models using Constructive Solid Geometry (CSG) operations. In particular, we assume that each object in the environment is represented in terms of polynomial equations.

The surfaces can be described using parametric or implicit representations. In particular, a parametric representation of a surface in homogeneous coordinates is :

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

Algebraic surfaces are represented as $f(x,y,z) = 0$, where $f(x,y,z)$ is a polynomial function. Typically, models consist of piecewise parametric or piecewise algebraic descriptions, where we associate a suitable domain with representation.

2.2 Motion Description

All objects are defined with respect to a global coordinate system, the world frame. The initial configuration is specified in terms of the world frame system. As the objects undergo rigid motion, we update their positions using a 4×4 homogeneous transformation matrix, M, used to represent the rotation as well as translational components of the motion (with respect to the origin). The algorithm is based only on local features of the polyhedra (or bounding polytope representations of curved models) and does not require the position of every feature to be updated for the purpose of contact determination at every time instant.

3 Collision Detection for Polyhedra

In this section, we briefly review the simple and efficient collision detection algorithm for convex polyhedra. More details are given in [22, 21]. The algorithms proceeds by tracking the pair of closest features (vertex, edge, or face) between the two convex polyhedra. Based on the closest feature pair it is able to calculate the Euclidean distance between them to detect possible collision. The method is applicable in static environments, 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 infrequently as the objects move along finely discretized paths. By preprocessing the polyhedra, the algorithm runs in roughly *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 (run time is proportional to the number of feature pairs traversed).

3.1 Algorithm Overview

The algorithm is straightforward in its conception. We start with a candidate pair of features, one from each polyhedra, and check whether the closest points lie on these features. Since the objects are convex and regular, this is a local test involving only the boundary and coboundary of the candidate features. If the features fail 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 boundary and coboundary of constant size. As a result, we are able to verify 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. This algorithm outputs the pair of closest features (and closest points), and also the distance between two objects. If the distance is less than or equal to zero (plus δ - a small safety margin defined by user or determined by a given environment), then the objects interpenetrate or just touch.

3.2 Hierarchical Representation for Concave Objects

We extend the almost constant time algorithm for convex polyhedra to concave objects using hierarchical representation. We assume that each nonconvex object is represented as a union of convex pieces *or* composed of several nonconvex subparts, each of these can be further represented as a union of convex subparts or a union of concave pieces. We use a sub-part hierarchy tree to represent each nonconvex object (including curved objects which will be discussed later). At each node of the tree, we store either a convex sub-part or the union of several convex subparts. The algorithm computes the convex hull for each leaf and

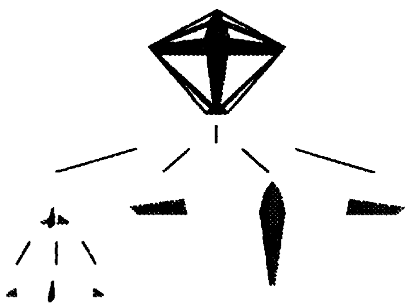

Figure 1: A hierarchical tree representation of an aircraft

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 the the convex hulls of its children (as in Figure 3.2). 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 using a recursive technique. The algorithm first checks for collision between two parent convex hulls. In case 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 an overlap between the convex hulls, the algorithm traverses to their children. This process is repeated as long as the convex hulls at the intermediate nodes are overlapping. Final a collision among the leaf nodes implies a collision between the original models.

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 concave objects.

4 Contact Determination between Curved Surfaces

In this section, we analyze the problem of contact determination between curved objects represented as piecewise parametric surfaces or piecewise algebraic models.

In general, given two rational parametric or algebraic surfaces, there is no good and quick solution to find out whether there is an actual contact between the two surfaces. The simplest solution is based on the use of bounding polytopes for each model. The bounding polytopes may correspond to rectangular bounding boxes or control polytopes associated with the NURBS surfaces [18]. However, bounding polytopes are useful for performing a rejection test only. In case the two bounding polytopes overlap, they can be further subdivided using a octree representations or subdivision algorithms for NURBS surfaces. In case, there is an actual contact between the two curved models, this approach based on subdivision converges linearly to the solution and can be slow in practice. We reduce these problems to finding solutions of a system of algebraic equations. In particular, we present algebraic formulations for the computation of closest *features* between curved models.

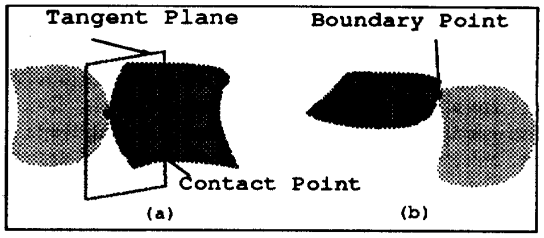

Figure 2: Tangential intersection and boundary intersection between two parametric surfaces

4.1 Closest Features

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.

$$\begin{aligned} \mathbf{H}_1(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{G}_u(u,v) = 0 \\ \mathbf{H}_2(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{G}_v(u,v) = 0 \\ \mathbf{H}_3(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{F}_s(s,t) = 0 \\ \mathbf{H}_4(s,t,u,v) &= (\mathbf{F}(s,t) - \mathbf{G}(u,v)) \bullet \mathbf{F}_t(s,t) = 0, \end{aligned} \tag{2}$$

where $\mathbf{F}_s, \mathbf{F}_t, \mathbf{G}_u, \mathbf{G}_v$ correspond to the partial derivatives and $\bullet$ corresponds to the dot product. This results in 4 equations in 4 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).

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

$$\begin{aligned} f(x_1,y_1,z_1) &= 0 \\ g(x_2,y_2,z_2) &= 0 \\ \begin{pmatrix} f_x(x_1,y_1,z_1) \\ f_y(x_1,y_1,z_1) \\ f_z(x_1,y_1,z_1) \end{pmatrix} &= \alpha_1 \begin{pmatrix} g_x(x_2,y_2,z_2) \\ g_y(x_2,y_2,z_2) \\ g_z(x_2,y_2,z_2) \end{pmatrix} \\ \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} &= \alpha_2 \begin{pmatrix} g_x(x_2,y_2,z_2) \\ g_y(x_2,y_2,z_2) \\ g_z(x_2,y_2,z_2) \end{pmatrix} \end{aligned} \tag{3}$$

These sets of equations can have high algebraic complexity. Even for surfaces of degree as low as three, finding all the solutions to these equations in the suitable domain can take order of a few minutes. Local optimization need good initial guess to all the possible closest pair features. For concave curved models there are typically more than one closest features pair and finding all of them using local optimizations method is fairly non-trivial.

4.2 Contact Formulation

We now formulate the algebraic constraints for a point contact between two curved models. Given two objects moving towards each other, they collide whenever these equations are simultaneously satisfied. The contact can be characterized as either a tangential intersection or a boundary intersection.

- *Tangential Intersection :* This corresponds to a tangential intersection between the two surfaces at a geometric contact point, as in Figure 2(a). At the contact point the normal vectors to the two surfaces are collinear. For parametric surfaces, these constraints can be formulated as:

$$\mathbf{F}(s,t) = \mathbf{G}(u,v) \tag{4}$$
$$(\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t)) \bullet \mathbf{G}_u(u,v) = 0$$
$$(\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t)) \bullet \mathbf{G}_v(u,v) = 0$$

  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 vector 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 typically such equations have no common solution. A tangential contact between the original models implies a common solutions to all these equations. To check for a tangential contact, we compute all the solutions to the first four equations using resultants [24] and substitute them into the fifth equation.

  Similarly for two algebraic surfaces, the problem of tangential intersection can be formulated as:

$$f(x,y,z) = 0$$
$$g(x,y,z) = 0 \tag{5}$$
$$\begin{pmatrix} f_x(x,y,z) \\ f_y(x,y,z) \\ f_z(x,y,z) \end{pmatrix} = \alpha \begin{pmatrix} g_x(x,y,z) \\ g_y(x,y,z) \\ g_z(x,y,z) \end{pmatrix}$$

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

- *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:

$$\mathbf{F}(s,1) = \mathbf{G}(u,v) \tag{6}$$

  Other possible boundary intersections can be computed in a similar manner. An example has been shown in Figure 2(b)

Two objects collide if one of these sets of equations, ((4) or (6)) for parametric surfaces and (5) for algebraic surfaces, have a common solution in their domain.

In a few degenerate cases, it is possible that the system of equations (4) or (5) and (6) have an infinite number of solutions. One such example is two cylinders parallel to each other. 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 [24].

5 Coherence for collision detection between curved objects

In most dynamic environments, we call the collision detection routine at fairly small time intervals. In this section, we present algorithms utilizing temporal and spatial coherence between successive instances for collision detection.

5.1 Approximating Curved Objects by Polyhedral Models

We approximate each curved model by a polyhedra (or polygonal mesh). We apply the fast collision detection algorithm for polyhedra to these polyhedral approximations for curved models. Eventually a geometric contact is determined by combining this approach and the equations highlighted in the previous section. We use an *ϵ-polytope* approximation for a curved surface. It is defined as:

Definition: Let S be a surface and P be an ϵ-polytope approximation, if for every point $\mathbf{p}$ on the boundary of polytope P, there is a point $\mathbf{s}$ on S such that $\| \mathbf{s} - \mathbf{p} \| \leq \epsilon$. Similarly for each point $\mathbf{s}$ on S, there is a point $\mathbf{p}$ on the boundary of P such that $\| \mathbf{p} - \mathbf{s} \| \leq \epsilon$.

An ϵ-polytope approximation is obtained by using either a simple mesh generation algorithm or an adaptive subdivision of the surfaces. Given a user defined ϵ, algorithms for generating such meshes are highlighted for parametric surfaces in [13] and for algebraic surfaces in [15]. In our implementation we used an inscribed polygonal approximation to the surface boundary. The ϵ-polytope approximations are for convex models only. The resulting polytope is convex as well.

5.2 Convex Curved Surfaces

Given two convex surfaces (say S_A and S_B), we compute an ϵ-polytope approximation for each surface. Let P_A and P_B be ϵ_A-polytope and ϵ_B-polytope approximations of S_A and S_B, respectively. At any instance, let d_p be the minimum distance between P_A and P_B (computed using the polyhedral collision detection algorithm [22]). Let d_s be the minimum distance between the two surfaces. It follows from the (inscribed) ϵ-approximation:

$$d_p - \epsilon_A - \epsilon_B \leq d_s \leq d_p. \tag{7}$$

The algorithm proceeds by keeping track of the closest features between P_A and P_B and updating the bounds on d_s based on d_p. Whenever $d_p \leq \epsilon_A + \epsilon_B$, we use local optimization routines to find the closest features on the surfaces S_A and S_B. In particular, we formulate the problem: For optimization routines, we want to minimize the function

$$\mathbf{H}(s,t,u,v) = \sum_{i=1}^{4}(H_i(s,t,u,v))^2,$$

where $\mathbf{F}_1$ abd $\mathbf{F}_2$ are defined in (2). We use *Gauss-Newton* algorithm to minimize $\mathbf{F}$. The initial guess to the variables is computed in the following manner.

We use the line, say $L_{A,B}$, joining the closest features of P_A and P_B as an initial guess to the line joining the closest points of S_A and S_B (in terms of direction). The initial estimate to the variables in the equations in (2) is obtained by finding the intersection of the line $L_{A,B}$ with the surfaces, $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$. This corresponds to a line-surface intersection problem and can be solved using subdivision or algebraic methods [12, 24]. As the surfaces move along, the coefficients of the equations in (2) are updated according to the rigid motion. The closest points between the resulting surfaces are updated using optimization routines. Finally, when these closest points coincide, there is a collision.

In practice, the convergence of the optimization routines to the closest points of S_A and S_B is a function of ϵ_A and ϵ_B. In fact, the choice of ϵ in the ϵ-polytope approximation is important to the overall performance of the algorithm. Ideally, as $\epsilon \to 0$, we get a finer approximation of the curved surface and better the convergence of the optimization routines. However, a smaller ϵ increases the number of features in the resulting polytope. Though polyhedral collision detection is an expected constant time algorithm at each step, the overall performance of algorithm is governed by the total number of feature pairs traversed by the algorithm. The latter is dependent on motion and the resolution of the approximation. Consequently, a very small ϵ can slow down the overall algorithm. In our applications, we have chosen ϵ as a function of the dimension of a simple bounding box used to bound S_A. In particular, let l be dimension of the smallest cube, enclosing S_A. We have chosen $\epsilon = \delta l$, where $.01 \leq \delta \leq .05$. This has worked well in the examples we have tested so far.

The algorithm is similar for surfaces represented algebraically. Objects which can be represented as a union of convex surfaces, we use the hierarchical representation and the algorithm highlighted above on the leaf nodes.

5.3 Concave Curved Objects

In this section we outline the algorithm for concave surfaces, which cannot be represented as a union of convex patches. A common example is a torus. The approach based on ϵ-polytope approximation is not applicable, as the resulting polytope is concave and its convex decomposition would result in lots of convex polytopes (of the order of $O(1/\epsilon)$).

Given two concave surface models corresponding to piecewise parametric or piecewise algebraic, we enclose each with a convex polytope and apply the polyhedral algorithm to the resulting pair. After the resulting polytopes collide, there may or may not be a collision to the resulting pair of surfaces. The problem of collision detection between two parametric or algebraic surfaces is solved by finding all the solutions to the equations (4) or (5) and (6). A real solution in the domain to those equations implies a geometric collision. The algebraic method based on resultants and eigenvalues is used to find all the solutions to the equations (4) and (6) [24]. It is possible that these equations have no common solution. However, we apply the algebraic method to the first four equations in (4) (or three equations in (5)) and compute all the solutions. 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 (4) and (6), whose coefficients are slightly different as compared to the previous set. All the roots of the new set of equations are updated using Newton's method, by using the previous set of roots as initial guesses. The convergence of the Newton's Raphson iteration is a function of motion which the objects undergo between two time instances. In case the closest features change considerably, Newton's method may not converge and we use the global solver to compute all the solutions again. This procedure represents an algebraic analog of the geometric coherence exploited in the earlier section.

As the two objects move closer to each other, the imaginary components of some roots start decreasing. Finally, a real collision occurs when the imaginary component of one of the roots becomes zero. We do not have to track all the paths corresponding to all the solutions. After a few time steps, we only keep track of the solutions whose imaginary components are decreasing.

The complexity of this algorithm is dominated by the time used in global root finding. However, the algorithm is efficient for low degree curved models only. In particular, it is efficient for rational parametric surfaces of degree 2, polynomial parametric surfaces of degree up to 4 and algebraic surfaces of degree up to 5. For degrees beyond these, the global root finder can take order of minutes to compute all the solutions. One possible approach for higher degree surfaces is to use multiple hierarchy of approximations with a number of convex models at each stage.

6 Interference Test for Multiple Moving objects

To check for all possible contacts among n objects at all time can be quite time consuming, especially in a large environment. In order to avoid unnecessary computations, we present 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.

6.1 Scheduling Scheme

The algorithm maintains a priority queue of all object pairs. They are sorted by bound on collision time, with the one most likely to collide appearing at the top of the queue. The approximation is a lower bound on the time to collision, so no collisions are missed. Non-convex objects, which are represented as hierarchy trees, are treated as single objects from the point of view of the queue. That is, only the roots of the hierarchy trees are paired with other objects in the queue.

The algorithm first has to compute the initial separation and bound on collision time among all pairs of objects, assuming that the magnitude of relative initial velocity, relative maximum acceleration and velocity limits among them are given. After initialization, at each step it only computes the closest feature pair and the distance between the pair of objects at the top of priority queue; meanwhile we ignore all other object pairs until the clock reaches the next "wakeup" time. "Wakeup" time W_i for each object pair P_i is defined as

$$W_i = t_w^i + t_0$$

where t_w^i is the lower bound on time to collision for each pair P_i for most situation and t_0 is the current time. t_w^i can be derived as the following:

Let a_{max} be an upper bound on the relative acceleration between any two *points* on any pair of objects. The bound a_{max} can be easily obtained from bounds on the absolute linear and angular acceleration and angular velocities of the bodies and their diameters. Let d be the initial separation for a given pair of objects, and v_i the initial relative velocity of the closest points on these objects. Then we can bound the time t_c^i to collision as

$$t_c^i = \frac{\sqrt{v_i^2 + 2a_{max}d} - v_i}{a_{max}}$$

We redefine t_w^i as $t_w^i = \max(t_c^i, t_{min})$, where t_{min} is a minimum effective time resolution of the calculation.

If no upper bounds on the velocity and acceleration can be assumed, we suggest algorithms which impose a bounding box hierarchy on each object in the environment to reduce the naive bound of $O(n^2)$ pairwise comparisons for a dynamic environment of n objects.

6.2 Sorting and Interval Tree

In the three-dimensional workspace, if two bodies collide then their projections down to the lower-dimension hyperplanes must overlap as well. Therefore, if we can efficiently *update* their overlapping status in each axis or in a 2-dimensional hyperplane, 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.

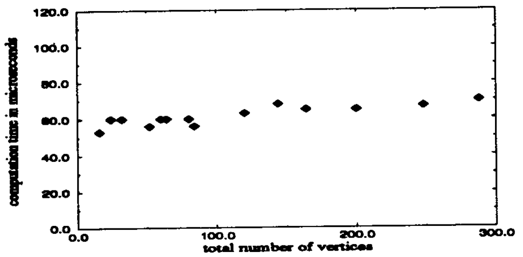

Figure 3: Computation time vs. total no. of vertices

In computational geometry, algorithms to solve the overlapping problem for d-dimensional bounding boxes in $O(nlog^{d-1}n + s)$ time are known, where s is the number of pairwise overlaps [11, 19, 29]. This bound can be improved using coherence [3].

6.2.1 One-Dimensional Sweep and Prune

Let a one-dimensional bounding box be $[b, e]$ where b and e are the real numbers representing the beginning and ending points. To determine all pairs of overlapping intervals given a list of n intervals, we need to verify for all pairs i and j if $b_i \in [b_j, e_j]$ or $b_j \in [b_i, e_i]$, $1 \leq i, j \leq n$. This can be solved by first sorting a list of all b_i and e_i values, from the lowest to the highest. Then, the list is swept to find all the intervals which overlap. The sorting process takes $O(nlogn)$ and $O(n)$ to sweep through a sorted list and $O(s)$ to output each overlap where s is the number of overlap.

6.2.2 Interval Tree for 2D Intersection Tests

Another approach is to extend the one-dimensional sweeping and pruning technique to the higher dimensional. We use an "interval tree" for 2-dimensional intersection tests [29] to reduce the number of pairwise checks for overlapping. However, as mentioned earlier, the time bound will be worse than $O(n)$ for two or three-dimensional sweep and prune due to insertion and deletion of an interval in a tree structure. It is more efficient for a dense environment where swaps of intervals along each axis occur frequently.

7 Experimental Results

We have implemented the above algorithms in C and Lisp and tested their performance on various environments. The collision detection routine for convex polytopes is very efficient. Our implementation in C gives us a roughly constant time performance of 50-70 microseconds per object pairs of arbitrary complexity on SGI Indigo (200-300 microseconds per object pairs on a 12.5 Mips 1.4 Mega flops Sun4 Sparc Station). Figure 3 shows the roughly constant time performance for polygonized spherical and cylindrical objects of various resolutions. Each data point is taken as the average value of 1000 trials.

As for curved objects, the algorithm only checks for collision between their polyhedral control polytopes, which takes roughly constant time. For example, once the control polytopes of two tori intersect, the problem is reduced to finding the eigendecomposition of a 96×96 matrix [24] using resultants. The eigendecomposition takes slightly more than a second on the IBM RS/6000.

The pairwise polyhedral collision algorithm and scheduling scheme with bounding boxes have been used in dynamic simulator developed at Berkeley [26] and significantly reduced the overall run time to make real time dynamic simulation possible. The hybrid algorithm for multiple objects reduces overall run time remarkably as compared to keeping track of $O(n^2)$ features for an environment of n bodies. Efficient algorithms for collision detection between large-scaled environments and their application to architecture walkthrough are presented in [7].

We have applied our implementation on a Puma560 moving in an environment with several obstacles. When there is no critical event (one or multiple collisions), the scheduling scheme only checks for possible interferences between a Puma link and one obstacle which are most likely to collide. When there is at least one pairwise contact occurring, usually only a small portion of priority queue gets rearranged.

Comparing these performance figures with more recent results [14, 16, 31], we feel that our approaches compare very favorably as on-line collision detection for obstacle avoidance path planning or other dynamic simulations in robotics as well.

References

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

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

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

[4] J. W. Boyse. Interference detection among solids and surfaces. *Comm. ACM*, 22(1):3–9, 1979.

[5] S. A. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. *Proc. IEEE ICRA*, pages pp. 591–596, 1986.

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

[7] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. Exact collision detection for interactive, large-scaled environments. Tech Report #TR94-005, 1994. University of North Carolina, Chapel Hill.

[8] 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.

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

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

[11] H. Edelsbrunner. A new approach to rectangle intersections, part i&ii. *Intern. J. Computer Math.*, 13:pp. 209–229, 1983.

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

[13] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *GAGD*, 3:295–311, 1986.

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

[15] Mark Hall and Joe Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(6):33–42, November 1990.

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

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

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

[19] 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.

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

[21] Ming. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, 1993.

[22] Ming C. Lin and John F. Canny. A fast algorithms for incremental distance calculation. *Proc. IEEE ICRA 1991*, 2:1008–1014, 1991.

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

[24] 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.

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

[26] B. Mirtich and J. Canny. Impusle-based, real time dynamic simulation. Submitted to ACM SIGGRAPH, 1993. University of California, Berkeley.

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

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

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

[30] J. M. Snyder and etc. Interval methods for multi-point collisions between time-dependent curved surfaces. *ACM SIGGRAPH*, pages 321–334, August 1993.

[31] S. Zeghloul, P. Rambeaud, and J. Lallemand. A fast distance calculation between convex objects by optimization approach. *IEEE Conference on Robotics and Automation*, pages pp. 2520–2525, 1992.