Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces

John M. Snyder, Adam R. Woodbury,
Kurt Fleischer, Bena Currin, Alan H. Barr
California Institute of Technology
Pasadena, CA 91125

Abstract

We present an efficient and robust algorithm for finding points of collision between time-dependent parametric and implicit surfaces. The algorithm detects simultaneous collisions at multiple points of contact. When the regions of contact form curves or surfaces, it returns a finite set of points uniformly distributed over each contact region.

Collisions can be computed for a very general class of surfaces: those for which inclusion functions can be constructed. Included in this set are the familiar kinds of surfaces and time behaviors encountered in computer graphics.

We use a new interval approach for constrained minimization to detect collisions, and a tangency condition to reduce the dimensionality of the search space. These approaches make interval methods practical for multi-point collisions between complex surfaces. An interval Newton method based on the solution of the interval linear equation is used to speed convergence to the collision time and location. This method is more efficient than the Krawczyk–Moore iteration used previously in computer graphics.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; G.4 [Mathematical Software]: Reliability and Robustness

General Terms: collision detection, parametric surface, constrained minimization, interval analysis

Additional Key Words: inclusion function, interval Newton method, interval linear equation

1 Introduction

Detecting geometric collisions between curved, time-dependent (moving and deforming) objects is an important and difficult problem in computer graphics. This paper discusses a practical and robust algorithm for detecting collisions between objects represented as parametric or implicit surfaces. We ignore the problem of computing the physical response to collisions; much of this topic is treated in other work [BARA90,META92]. Instead, we concentrate on the purely geometric problem of computing a solution set of points where a set of time-dependent surfaces first contact (Figure 1).

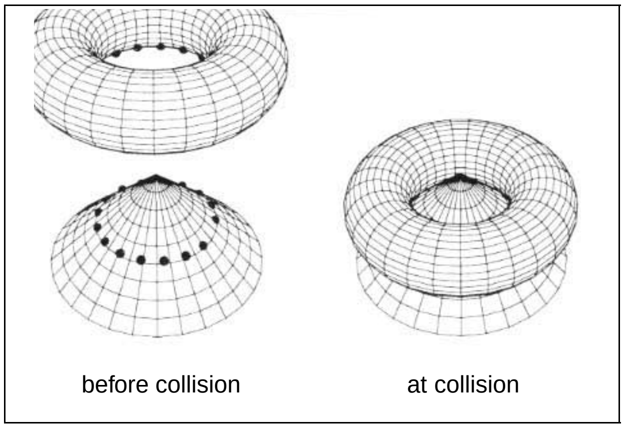

Figure 1: Problem Statement: Given a collection of time-dependent curved surfaces, find a set of collision points representing the contact regions. In this example, the dots show the points detected by the collision algorithm when a torus moves down over a cone, contacting it in a circle.

Previous work on geometric collision detection is fairly extensive, both in computer graphics and in other fields such as CAD/CAM and robotics. Detection of collisions between polyhedral objects was studied in [MOOR88]. Baraff [BARA90] presented a method of computing collisions between parametric or implicit surfaces by computing extremal points using non-linear equation solvers. Sclaroff and Pentland [SCLA91] present a method for detecting collisions between implicit surfaces by "plugging" vertices of a polyhedral approximation of one surface into the inside-outside function of the other. Von Herzen, et. al., [VONH90] presented an algorithm for detecting collisions of parametric surfaces using Lipschitz bounds. Duff [DUFF92] used interval methods to compute collisions between boolean combinations of implicit surfaces.

To make collision detection practical, much of the previous work traded off accuracy and robustness for efficiency, or limited the kinds of shapes that could be handled. Polyhedral methods such as in [MOOR88], although fairly efficient, are not well suited to surfaces that deform in time. Exploiting coherence for rolling or sliding contact of polyhedral objects is difficult, and use of a fixed sampling mesh can cause severe approximation errors. Polyhedral methods also require many numerically difficult special cases which led [MOOR88] and [SCLA91] to neglect cases where "tunneling" may occur either between polygon edges or between small implicit surfaces passing entirely through a large polygon.

Baraff [BARA90] chose to limit objects to the union of con-

```
take one or more steps in the ODE solver
compute collisions in the resulting time interval
if a collision occurs (at time $t^*$) in the interval
    compute a collision response
    reset ODE solver to $t = t^*$
endif
```

Figure 2: Computational Model for Collision Detection and Response

vex polyhedra and strictly convex closed surfaces. This restriction simplified his collision detection algorithm and allowed tracking of single contact points between curved objects. He did not treat non-convex surfaces (such as saddle shapes) and manifolds with boundary (such as half a sphere). We solve the problem for a more general class of surfaces with many points of contact, as shown in Figure 1.

As noted in [VONH90], methods which depend solely on point-wise evaluations, including the above methods, cannot guarantee accurate collision detection. To solve this problem, Von Herzen bounded the output of functions over a region using a Lipschitz bound. Duff [DUFF92] used interval analysis to produce tighter bounds than Von Herzen's Lipschitz bound. Both of these methods used binary subdivision to search for collisions; we speed up the approach significantly by combining binary subdivision with an interval Newton method.

The technique we describe offers several fundamental improvements over previous techniques:

1. The most novel aspect of our technique is the ability to detect simultaneous collisions (multiple contacts at the same time), *even when the collisions occur at a higher dimensional manifold of contact, rather than at a set of isolated points*. In this case, the algorithm samples the region of contact with a finite, uniformly-distributed set of points. The spatial sampling density is a parameter to the algorithm. To our knowledge, no previous algorithm handles this situation.
2. Our technique works for both rigid and deforming objects, and for implicit or parametric objects.
3. Our technique is practical for computer graphics applications, and has been used in animations involving hundreds of objects.
4. Our technique includes a method (tangency constraints) to reduce the dimensionality of the space of possible solution points, as shown in Figure 3, dramatically speeding up the method. The tangency constraints also provide a square system of equations for the interval Newton method, helping us detect isolated point collisions.
5. Our technique uses a test for uniqueness of roots of a system of equations in a region. This test can be verified in many cases, allowing the algorithm to terminate without further subdivision around collision points.
6. Our technique can be used both to compute collisions between formerly disjoint bodies which come into contact, or to compute additional points of contact between bodies as they roll or slide over each other (see Section 1.1).

1.1 Fitting Collision Detection into a Larger System

Figure 2 shows how collision detection fits into a larger program for computing physical simulations of dynamic systems. The system is composed of three parts: the ODE (ordinary differential equation) solver module, the collision detection module, and the collision response computation module. The ODE solver computes the motions of objects over time, using equations governing the dynamic behavior of bodies, and produces a functional representation of the motion.[1] Motion is computed without considering collisions, so that the results are only valid until the next collision occurs. The collision detection module takes the functional representation produced by the ODE solver and computes when and where the first collision occurs in the given time interval. If a collision occurs, a collision response is computed, which may discontinuously change the state of the system of bodies. The ODE solver continues forward in time from this computed collision time, discarding any state after it.

Two modes of operation are required in collision detection:

1. compute any collisions for bodies that are initially not in contact
2. compute *additional* collisions for bodies that are already in continuous (rolling or sliding) contact

The algorithm described in this paper handles both situations. For greatest efficiency and modularity, we advocate handling coherence in the ODE solver. By coherence, we mean the tracking of contact points between bodies rolling or sliding over each other. In these situations, collision detection is required only to compute new points of contact not already tracked by the ODE solver (mode 2 above). The solver must therefore inform the collision detection module of the motion of the contact points it is tracking, so that these points may be excluded from consideration (see Eq. 7). The collision detection module must also compute the initial points of contact when the simulation is begun or when continuous contact begins between bodies (mode 1 above).

1.2 Overview

The mathematics of the collision detection problem is treated in Section 2. Sections 3, 4, and 5 discuss the constrained minimization algorithm, an interval Newton enhancement, and termination criteria, respectively. Section 6 presents a simple culling test which discards non-colliding surface pairs and tightens a bound on the collision time. The full collision algorithm, combining constrained minimization, the culling test, and other tools from computational geometry, is presented in Section 7. Our technique, like all interval methods, requires inclusion functions, whose construction is summarized in Section 8. Finally, results and conclusions are described in Sections 9 and 10. Appendix A extends our approach to surfaces that are piecewise smooth by adding conditions for face, edge, and vertex interactions (see Figure 11). Appendix B describes the construction of inclusion functions for Chebyshev polynomials.

2 The Collision Problem

The equations that specify that two surfaces collide may be divided into two parts: a *contact* constraint, that specifies that the two surfaces intersect, and a *tangency* constraint, that specifies that the two surfaces are tangent at their point of intersection. The tangency constraint reduces the dimensionality of the space of possible collision points, as shown in Figure 3. It also allows faster convergence (using interval Newton, which we will describe in Section 4)

[1] In our rigid body simulations, the solver produces a time-varying quaternion and translation vector. Each component of the quaternion and vector is represented using univariate Chebyshev polynomials.

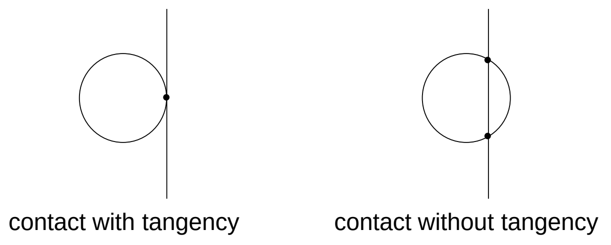

Figure 3: Reducing the Dimensionality of the Space of Collision Points Using the Tangency Condition: The intersection of two bodies (like a sphere moving to the right with a stationary plane) typically forms a whole 2D manifold of contact through time. With the tangency constraint, the solution space is often reduced to one or a few points by eliminating cases like that shown on the right. Reducing the solution space to an isolated space-time point is one of the ideas that makes this method practical.

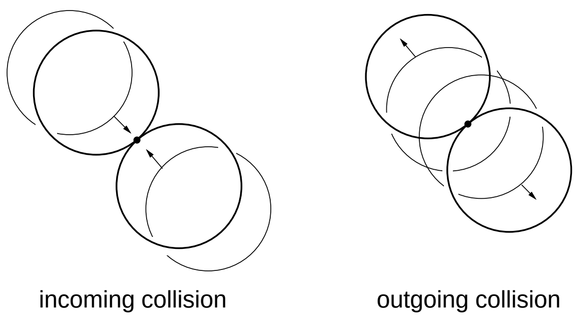

Figure 4: Incoming and Outgoing Collisions: The unbroken circles represent bodies later in time. A dot represents the collision point; the arrows represent the direction of movement.

and robust testing of isolated collisions (using an interval solution uniqueness test described in Section 5).

We also distinguish between *incoming* collisions, in which the surfaces collide by moving closer to each other, and *outgoing* collisions, in which the surfaces are interpenetrating and become tangent as they move apart. These situations are compared in Figure 4. The distinction is necessary in the simulation of dynamic systems where each surface encloses a solid. Eliminating outgoing collisions allows the simulator to ignore collisions which were previously detected; i.e., collisions between surfaces already in contact which are moving away as a *response* to the collision.

2.1 Parametric Surfaces

Let two deforming parametric surfaces be represented by the twice-differentiable mappings $S_1(u_1, v_1, t)$ and $S_2(u_2, v_2, t)$, where $S_i\colon \mathbf{R}^3 \to \mathbf{R}^3$. At a particular instant of time, each of the surfaces is formed by the image of S_i over a rectangle in (u_i, v_i) space.[2] In this section, we consider the case of collisions between solids each bounded by a single, smooth, closed parametric surface. Appendix A generalizes the discussion to parametric surfaces which are only piecewise smooth.

Contact Constraint The contact constraint merely states that the two surfaces intersect (i.e., the vector difference of the two surfaces is the zero vector):

$$S_1(u_1, v_1, t) - S_2(u_2, v_2, t) = 0. \tag{1}$$

Tangency Constraint The tangency constraint implies that the instantaneous normal vectors on the two surfaces at their point of contact are anti-parallel. Stated another way, the (u, v) tangent vectors on one surface must be perpendicular to the instantaneous normal vector on the other surface. We thus have the following system of two equations[3]

$$\begin{pmatrix} \dfrac{\partial S_1}{\partial u_1}(u_1, v_1, t) \cdot N_2(u_2, v_2, t) \\ \dfrac{\partial S_1}{\partial v_1}(u_1, v_1, t) \cdot N_2(u_2, v_2, t) \end{pmatrix} = 0 \tag{2}$$

where N_1 and N_2 are the outward normal vectors to the surfaces S_1 and S_2, respectively, given by

$$N_i(u_i, v_i, t) \equiv \frac{\partial S_i}{\partial u_i}(u_i, v_i, t) \times \frac{\partial S_i}{\partial v_i}(u_i, v_i, t) \quad \text{for } i = 1, 2.$$

The algorithms that follow here assume that N_1 and N_2 are nowhere 0; that is, surfaces have a nonvanishing normal vector everywhere and for all relevant time.[4] The whole collision equality constraint is given by a nonlinear system of 5 equations in 5 variables, three from Eq. 1 and two from Eq. 2.

Incoming Constraint The incoming collision condition states that the relative velocity of the collision point must face the same way as the surface normal (the two vectors must form an acute angle),[5] and the two normals must face in opposite directions (forming an obtuse angle). This condition yields two inequality constraints:

$$\begin{gathered} \left(\frac{\partial S_1}{\partial t}(u_1, v_1, t) - \frac{\partial S_2}{\partial t}(u_2, v_2, t)\right) \cdot N_1(u_1, v_1, t) \geq 0 \\ \text{and} \quad - N_1(u_1, v_1, t) \cdot N_2(u_2, v_2, t) \geq 0. \end{gathered} \tag{3}$$

2.1.1 Example: Rigid Parametric Surfaces

The above constraints may be applied to the special case of rigid parametric surfaces. In this case, we have two time-independent surfaces $s_1(u_1, v_1)$ and $s_2(u_1, v_1)$. The time-varying version of these surfaces is given by

$$S_i(u_i, v_i, t) \equiv R_i(t)\, s_i(u_1, v_1) + T_i(t) \quad \text{for } i = 1, 2$$

where $R_i(t)$ is a time-varying rotation matrix and $T_i(t)$ is a time-varying translation vector, specifying the trajectory of surface i's coordinate origin.

Contact Constraint The contact constraint may be expressed as

$$R_1(t)\, s_1(u_1, v_1) + T_1(t) - R_2(t)\, s_2(u_2, v_2) - T_2(t) = 0.$$

[2]Using a rectangular domain for parametric surfaces does not limit the kinds of surfaces that can be collided. Parametric surfaces defined on non-rectangular domains can be handled by mapping a rectangle into the required non-rectangular domain before mapping onto the surface [SNYD92b].

[3]A similar, though functionally dependent, constraint may be derived by switching S_1 and S_2.

[4]If the calculated normal vector becomes zero, such as at the poles of a parametric sphere, the tangency constraint becomes trivially true. The algorithm will therefore rely on the contact constraint to detect a collision in this case.

[5]We assume here that the surfaces are parameterized so that the normals N_1 and N_2 face outward.

Tangency Constraint Let $n_1(u_1, v_1)$ and $n_2(u_2, v_2)$ be the time-independent normals of the surfaces s_1 and s_2, given by

$$n_i(u_i, v_i) \equiv \frac{\partial s_i}{\partial u_i}(u_i, v_i) \times \frac{\partial s_i}{\partial v_i}(u_i, v_i).$$

The time-varying surface normals can therefore be expressed as

$$N_i(u_i, v_i, t) \equiv R_i(t)\, n_i(u_i, v_i)$$

since $R_i(t)$ is a rotation matrix. The tangency constraint is then given by

$$\begin{pmatrix} (R_1(t)\, \frac{\partial s_1}{\partial u_1}(u_1, v_1)) \cdot N_2(u_2, v_2, t) \\ (R_1(t)\, \frac{\partial s_1}{\partial u_2}(u_1, v_1)) \cdot N_2(u_2, v_2, t) \end{pmatrix} = 0.$$

Incoming Constraint The incoming constraint is given by

$$\begin{gathered} \big[\dot{R}_1(t)\, s_1(u_1, v_1) + \dot{T}_1(t) - \dot{R}_2(t)\, s_2(u_2, v_2) - \\ \dot{T}_2(t)\big] \cdot N_1(u_1, v_1, t) \geq 0 \\ \text{and} \quad -N_1(u_1, v_1, t) \cdot N_2(u_2, v_2, t) \geq 0 \end{gathered}$$

where $\dot{R}_i$ and $\dot{T}_i$ are the time derivatives of the rotation matrix and translation vector of the two surfaces.

2.2 Implicit Surfaces

Let two time-varying implicit surfaces be represented using the scalar functions $F_1(x, y, z, t)$ and $F_2(x, y, z, t)$. Points on each surface are defined as the zero-sets of these functions.

Contact Constraint The contact constraint is the system of two equations

$$\begin{pmatrix} F_1(x, y, z, t) \\ F_2(x, y, z, t) \end{pmatrix} = 0. \tag{4}$$

Tangency Constraint Let the function $\nabla F_i(x, y, z, t)$ be the spatial gradient of the implicit functions (i.e., with respect to x, y, and z). The tangency constraint is then given by

$$\nabla F_1(x, y, z, t) \times \nabla F_2(x, y, z, t) = 0. \tag{5}$$

This constraint, although a system of three equations, contains only two functionally dependent equations. The entire collision equality constraint for implicit surfaces is thus given by a system of five (four functionally independent) equations in the four variables x, y, z, and t (Eqs. 4 and 5).[6]

Incoming Constraint The incoming constraint is given by

$$\begin{gathered} -\frac{\partial F_1}{\partial t}(x, y, z, t)\, \|\nabla F_2(x, y, z, t)\| - \\ \frac{\partial F_2}{\partial t}(x, y, z, t)\, \|\nabla F_1(x, y, z, t)\| \geq 0 \qquad (6) \\ \text{and} \quad -\nabla F_1(x, y, z, t) \cdot \nabla F_2(x, y, z, t) \geq 0. \end{gathered}$$

[6]We note that detecting collisions between implicit and parametric surfaces is a simpler problem than colliding pairs of parametric or implicit surfaces. By substituting the output of the parametric surface as the input (x, y, z) of the implicit surface, a system in 3 variables, (u, v, t), results, where u and v are the parametric surface coordinates.

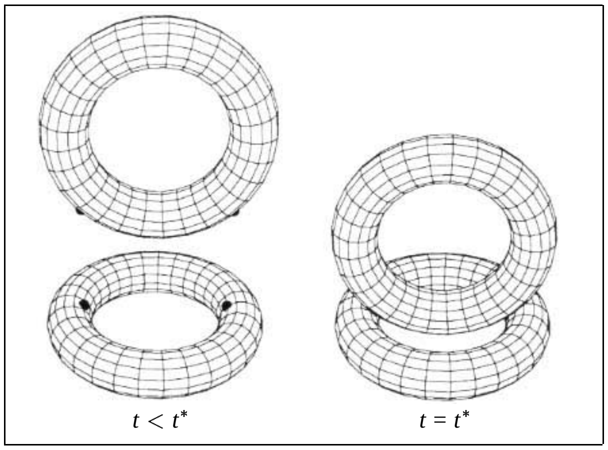

Figure 5: Simultaneous Collisions — The tori collide at two isolated points.

We also note that CSG operations on implicit surfaces, as in [DUFF92], can be handled very efficiently using our techniques. Assume the surface F_1 is represented as the boolean subtraction of two simple implicit surfaces $F_a(x, y, z, t)$ and $F_b(x, y, z, t)$. The first equation in the contact constraint (Eq. 4) then becomes

$$(F_a = 0 \text{ and } F_b \geq 0) \text{ or } (F_b = 0 \text{ and } F_a \leq 0)$$

assuming implicit surface functions are positive outside the surface they represent. Similar restricted equality constraints can be derived for the tangency constraint.

Constraints for rigid motion of implicit surfaces are easily derived by applying the above general equations to the rigidly moving implicit surface

$$F(x, y, z, t) \equiv f(w) \text{ where } w \equiv R^T(t)((x, y, z)^T - T(t))$$

where $f \colon \mathbf{R}^3 \to \mathbf{R}$ is the implicit equation of the time-independent surface, $R^T(t)$ is the transpose of the time-varying rotation matrix, and $T(t)$ is the time-varying translation vector.

2.3 Collision As a Constrained Minimization Problem

The final collision constraint may be described as an equality constraint involving a function C (for the contact and tangency constraints) and a logical composition of inequality constraints involving a function D (for the incoming constraint). For collisions between parametric surfaces, C and D are vector functions of (u_1, v_1, u_2, v_2, t) (equations 1–3); for implicit surfaces they are functions of (x, y, z, t) (equations 4–6). We are interested only in the minimum t collision, since our representation for the time behavior of the surfaces may be invalid after this time.

The desired collision time for parametric surfaces, t^*, can therefore be expressed using the constrained minimization problem

$$\underset{(u_1, v_1, u_2, v_2, t) \in X_0}{\text{minimum}} \left\{ t \;\middle|\; \begin{array}{ll} C(u_1, v_1, u_2, v_2, t) = 0 & \text{and} \\ D(u_1, v_1, u_2, v_2, t) \geq 0 & \end{array} \right\}.$$

A similar statement results for detection of collisions between implicit surfaces. We would like to compute t^* or detect that the constraint is satisfied nowhere in the parameter space X_0.

We also need the location of the collision and the surface normal vectors there. There may be multiple points of contact at the time of collision, which we call *simultaneous* collisions, as shown in Figure 5. The points of contact may be a finite number of *isolated points* as in Figure 5, or they may form a curve or surface, called the *contact manifold*. For example, if the falling torus from Figure 5 were in the same orientation as the stationary one at the bottom, the collision points would form a circle.

To detect simultaneous collisions, we need to detect minimum t solutions which are simultaneous or simultaneous within some tolerance ϵ. Mathematically, we require the set of *collision points* $(u_1^*, v_1^*, u_2^*, v_2^*)$ such that

$$C(u_1^*, v_1^*, u_2^*, v_2^*, t^*) = 0 \text{ and } D(u_1^*, v_1^*, u_2^*, v_2^*, t^*) \geq 0.$$

For a given collision point, the location of the collision, p^*, is

$$p^* \equiv S_1(u_1^*, v_1^*, t^*) \equiv S_2(u_2^*, v_2^*, t^*).$$

The normal vectors at the collision may be defined similarly by evaluating N_1 and N_2 at the collision point.

To compute a collision among a set of N time-varying parametric surfaces $S_i(u_i, v_i, t)$, we first use simple culling procedures to exclude pairs of surfaces which can't collide. Section 3.2 will discuss a method of solving these sets of constrained minimization problems which can compute simultaneous collisions and which does not spend undue computation on collisions which occur after t^*. It returns the collision points when these occur at a finite set of isolated points, or a finite subset of the collision points uniformly distributed over the contact manifold.

3 Interval Tools for Computing Collisions

We now turn to a discussion of the interval tools necessary to solve the sets of constrained minimization problems that arise in collisions.

3.1 Review of Interval Analysis

An *interval*, $A = [a, b]$, is a closed subset of $\mathbf{R}$ defined as

$$[a, b] \equiv \{x \mid a \leq x \leq b,\ x, a, b \in \mathbf{R}\}.$$

The lower and upper bounds of an interval are written as

$$\begin{aligned} \mathsf{lb}[a, b] &\equiv a \\ \mathsf{ub}[a, b] &\equiv b. \end{aligned}$$

A *vector-valued interval of dimension n*, $A = (A_1, A_2, \ldots, A_n)$, is a subset of $\mathbf{R}^n$ defined as

$$A \equiv \{x \mid x_i \in A_i, i = 1, 2, \ldots, n\}$$

where each A_i is an interval. An interval A_i that is a component of a vector-valued interval is called a *coordinate interval of A*.

The *width* of an interval, written $\mathsf{w}([a, b])$, is defined by

$$\mathsf{w}([a, b]) \equiv b - a.$$

The *midpoint* of an interval, written $\mathsf{mid}([a, b])$, is defined by

$$\mathsf{mid}([a, b]) \equiv \frac{a + b}{2}.$$

Similarly, the width and midpoint of a vector-valued interval of dimension n, A, are defined as

$$\begin{aligned} \mathsf{w}(A) &= \max_{i=1}^{n} \mathsf{w}(A_i) \\ \mathsf{mid}(A) &= (\mathsf{mid}(A_1), \mathsf{mid}(A_2), \ldots, \mathsf{mid}(A_n)). \end{aligned}$$

Hereafter, we will use the term interval to refer to both intervals and vector-valued intervals; the distinction will be clear from the context.

An *inclusion function* for a function f, written $\Box f$, produces an interval bound on the output of f over an interval representing its input domain. Mathematically, for all intervals X in the domain of f, if a point x is in the input interval X then $f(x)$ is contained in the output interval $\Box f(X)$; i.e.,

$$x \in X \Rightarrow f(x) \in \Box f(X) \quad \text{for all } x \in X.$$

Much more information about inclusion functions and their properties can be found in the literature (see, for example, [MOOR79, ALEF83,RATS88]). Section 8 and the Appendices discuss ways to create inclusion functions given the functions they are to bound.

3.2 Constrained Minimization Algorithm

The constrained minimization problem involves finding the global minimizers[7] of an objective function $f\colon \mathbf{R}^n \to \mathbf{R}$ for all points that satisfy a constraint function $F\colon \mathbf{R}^n \to \{0, 1\}$.

For the case of computing collisions between parametric surfaces, we have the following variables, objective function, and constraint function:

$$\begin{aligned} x &\equiv (u_1, v_1, u_2, v_2, t) \\ f(x) &\equiv t \\ F(x) &\equiv (C(u_1, v_1, u_2, v_2, t) = 0) \text{ and} \\ & (D(u_1, v_1, u_2, v_2, t) \geq 0) \end{aligned}$$

A region in the minimization algorithm is a 5D interval vector of the form

$$\begin{aligned} X &\equiv (U_1, V_1, U_2, V_2, T) \\ &\equiv \left(\left[u_1^l, u_1^u\right], \left[v_1^l, v_1^u\right], \left[u_2^l, u_2^u\right], \left[v_2^l, v_2^u\right], \left[t^l, t^u\right]\right) \end{aligned}$$

where the superscripts l and u denote lower and upper bounds. The relevant inclusion functions are[8]

$$\begin{aligned} \Box f(X) &\equiv \left[t^l, t^u\right] \\ \Box F(X) &\equiv \begin{cases} [0, 1], & \text{if } \mathsf{lb}\,\Box C(X) \leq 0,\ \mathsf{ub}\,\Box C(X) \geq 0, \\ & \quad \text{and } \mathsf{ub}\,\Box D(X) \geq 0 \\ [0, 0], & \text{otherwise} \end{cases} \end{aligned}$$

where $\Box C$ is an inclusion function for the collision equality constraint C, and $\Box D$ is an inclusion function for the incoming inequality constraint D.

The algorithm in Figure 6 finds solutions to the constrained minimization problem in a specified region X_0. The algorithm uses a priority queue to order regions based on the upper bound of the objective function. Regions bounding the set of global minimizers

[7]The global minimizers are the domain points at which the global minimum of the objective function is achieved, subject to the constraints.

[8]The terminology $\mathsf{lb}\,\Box C(X) \leq 0$ denotes that $\mathsf{lb}\,\Box C_i(x) \leq 0$ for *each* component interval i of $\Box C(X)$ (and similarly for upper bounds).

```
Minimize($\Box f$,$\Box F$,$A$,$X_0$,$\Box d$,$\epsilon$,$\delta$)
place initial region $X_0$ on priority queue $L$
initialize $f$'s upper bound $u \leftarrow +\infty$
initialize solution set $S \leftarrow \emptyset$
initialize singular solution set $\overline{S} \leftarrow \emptyset$
while $L$ is nonempty
    get next region $Y$ from $L$
    if lb $\Box f(Y) > u + \epsilon$ discard $Y$
    else if lb $\Box f(Y) > u - \epsilon$ and
        there exists $S_i \in \overline{S}$ such that $||\Box d(Y) - \Box d(S_i)|| < \delta$
        then discard $Y$
    else if $Y$ satisfies acceptance criteria $A$ then
        add $Y$ to solution list $S$
        if $Y$ doesn't contain a unique feasible point
            add $Y$ to $\overline{S}$
        endif
        $u \leftarrow \min(u, \text{ub } \Box f(Y))$
        delete from $S$ and $\overline{S}$ all $S_i \ni$ lb $\Box f(S_i) > u + \epsilon$
    else
        subdivide $Y$ into regions $Y_1$ and $Y_2$
        for $Y_i \in \{Y_1, Y_2\}$
            evaluate $\Box F$ on $Y_i$
            if $\Box F(Y_i) = [0, 0]$ discard $Y_i$
            evaluate $\Box f$ on $Y_i$
            if lb $\Box f(Y_i) > u + \epsilon$ discard $Y_i$
            insert $Y_i$ into $L$ according to ub $\Box f(Y_i)$
        endfor
    endif
endwhile
```

Figure 6: Global Constrained Minimization Algorithm: This algorithm finds the global minimizers of an objective function f, with constraints F, acceptance criteria A, initial region X_0, solution distance mapping function d, simultaneity threshold ϵ, and solution separation distance δ.

are subdivided until they are rejected or satisfy the acceptance criteria, A, and are accepted as solutions. It halts with an empty list of solutions if there are no solutions to the constraint function in X_0, or a list of regions, S, representing the set of global minimizers of the constrained minimization problem.

The variable u is a progressively refined least upper bound for the global minimum of the objective function. If we were only looking for a single collision point, we could halt the algorithm immediately after finding the first solution. To find collisions at multiple points of contact, the algorithm must be continued until the priority queue is empty. The variable u helps to prune the search after finding the first solutions.

Selecting Finite Sets of Points from Contact Manifolds The parameters ϵ, δ, and $\Box d$ allow the algorithm to select a finite set of regions distributed "uniformly" within the set of global minimizers, when this set is not finite. The parameter ϵ is the *simultaneity threshold*, which specifies how close the value of the objective function must be for two points to be considered global minimizers. For collisions, ϵ specifies how close in time two events must be in order to be considered simultaneous. The parameter δ is the *solution separation distance*, which specifies how far apart two accepted regions must be to be accepted as separate solutions. The parameter $\Box d$ is an inclusion function for the mapping which takes points in parameter space to points in whatever space we desire distances to be compared. We call the function d the *solution distance mapping function*.

As the algorithm progresses, it maintains two solution lists, S and $\overline{S}$. S contains all accepted regions. We call $\overline{S}$ the *singular solution set*. The elements of S not in $\overline{S}$ are regions in which the existence of a unique feasible point has been verified. The statements

```
if lb $\Box f(Y) > u - \epsilon$ and there exists $S_i \in \overline{S}$
    such that $||\Box d(Y) - \Box d(S_i)|| < \delta$
then discard $Y$
```

check that the region Y is not too close to regions already accumulated onto $\overline{S}$. Note that the test lb $\Box f(Y) > u - \epsilon$ is critical to ensure that Y doesn't have an objective function value small enough to invalidate all the currently accepted regions.[9]

We use two lists, S and $\overline{S}$, so that in the case that the global minimizers form a finite set of points, the algorithm can find all such points without discarding some based on distance to those already found. The algorithm is therefore able to resolve multiple isolated collisions that happen in a small area, regardless of the value of δ.[10]

Ordering Based on Upper Bounds Constrained minimization algorithms that have appeared before [RATS88,SNYD92c] order regions based on the *lower bound* of the objective function. We use the upper bound to make tractable computing solutions on a contact manifold.

At any time, the union of all regions on the priority queue forms a bound on the set of global minimizers of the constrained minimization problem. As the algorithm progresses, regions are subdivided or rejected, so that the regions which remain on the priority queue become a tighter bound on this set. Because of the inclusion monotonicity property of inclusion functions,[11] as regions on the queue shrink, the computed lower bound on the objective function tends to increase and the upper bound tends to decrease.

Assume the set of global minimizers forms a continuous manifold rather than a finite collection of isolated points, as shown Figure 1. If the priority queue is ordered using lower bounds, when a given region is subdivided, its children will generally have larger lower bounds for the objective function, and will be placed in the priority queue behind less highly subdivided regions. A breadth first traversal tends to result, with less highly subdivided regions examined first. If we have a whole manifold of global minimizers and stringent acceptance criteria, we will have to compute a huge number of tiny regions bounding the entire solution manifold before even the first region is accepted as a solution.

By ordering based on the upper bound, more highly subdivided regions tend to be examined first because they tend to have smaller upper bounds. We quickly get to a region which is small enough to satisfy the acceptance criteria. This allows our upper bound u to be updated. It also allows regions to be accumulated onto our singular list $\overline{S}$. Regions that are too close to any member of $\overline{S}$ can then be eliminated, making it possible to find a distribution of points on the contact manifold without undue computation.

Acceptance Criteria The constraint inclusion function, $\Box F(X)$, because it contains an equality constraint, returns either $[0, 0]$ (i.e., the constraint is satisfied nowhere in X) or $[0, 1]$ (i.e., the constraint

[9] If a region can possibly have a feasible point with a value of f less than ϵ from the value of f in regions on $\overline{S}$, we should not reject it just because it is close with respect to the function d to these regions. The algorithm might then discard a global minimizer because of its closeness to regions which are possibly far from the global minimizer, in terms of bounds on the objective function.

[10] One problem with this technique is that if the collisions happened at a contact manifold *and* a finite number of additional isolated points, the algorithm may discard some of the isolated points because of the closeness criterion. We consider this problem minor since the set of global minimizers is infinite and the algorithm must chose a subset anyway.

[11] An inclusion function, $\Box f$, is inclusion monotonic if $Y \subset X \Rightarrow \Box f(Y) \subset \Box f(X)$. In practice, the standard ways of constructing inclusion functions generate inclusion monotonic inclusion functions.

may be satisfied in X). We must resort to other means to determine if the constraint is *actually* satisfied. Section 5 discusses conditions which guarantee that a region contains a unique solution to the equality constraints. These conditions can therefore be used as acceptance criteria in the algorithm, which we call the *isolated point acceptance criteria*. They also allow the upper bound u to be updated via

$$u \leftarrow \min(u, \mathsf{ub}\,\Box f(Y))$$

since Y is guaranteed to contain a feasible point.

The algorithm also makes use of *emergency acceptance criteria* which do not guarantee a unique solution but are guaranteed to be satisfied for regions of small enough size.[12] The simplest such criterion is $\mathsf{w}(X) < \epsilon$; a better one is $\mathsf{w}(\Box S(X)) < \epsilon$ where $\Box S$ is the parametric mapping of one of the colliding surfaces. Regions which are accepted via the emergency acceptance criteria are inserted both onto the list of solutions, S, and the singular solutions, $\overline{S}$.

Subdivision The simplest method of subdividing candidate intervals in the minimization algorithm is bisection, in which two intervals are created by subdividing one of the input dimensions at its midpoint. Many methods can be used to select which dimension to subdivide. For example, we can simply pick the dimension of greatest width. A better alternative is to scale the parametric width by some measure of its importance to the problem we are solving. For each variable in the collision problem x_i, and a given candidate region X, we have used a scaling value s_i defined by

$$s_i \equiv \sum_{j=1}^{m} \max(|\,\mathsf{lb}\,\Box\frac{\partial f_j}{\partial x_i}(X)|, |\,\mathsf{ub}\,\Box\frac{\partial f_j}{\partial x_i}(X)||).$$

Here, f refers to the equality constraints (contact and tangency) of the collision problem. We then pick a dimension to subdivide, i, such that the scaled width $s_i\ \mathsf{w}(X_i)$ is largest.

Given a candidate interval, techniques also exist which allow us to compute a smaller interval which can possibly contain feasible points of the constraint. These methods and how they can be added to our simple minimization algorithm are discussed in Section 4. Even more sophisticated subdivision methods exist, such as Hansen's method which involves accumulating gaps inside candidate intervals by using infinite interval division (see [RATS88] for a full description).

Multiple Element Constrained Minimization The algorithm can easily be modified to accept an array of sets of minimization parameters $(\Box f, \Box F, A, X_0)_i$. This allows simultaneous solution of sets of problems from different pairs of surfaces, or different tangency situations for the same pair of piecewise parametric surfaces. As a result, computation is not wasted on collisions which happen after the first collision, $t > t^*$. We call this modified constrained minimization algorithm the *multiple element constrained minimization algorithm*.

Sets of minimization subproblems may be implemented by associating the array index of the appropriate minimization subproblem with each region inserted onto the priority queue, and using the appropriate indexed inclusion functions and acceptance criteria when processing the region.

[12] Although we cannot guarantee a region X contains a solution, we can guarantee that it is arbitrarily close, in the sense that $\mathsf{w}(\Box C(X)) < \epsilon$ where $\Box C$ is an inclusion function for the collision equality constraint function C.

Avoiding Detection of Tracked Points We can add additional inequality constraints to the constraint function F in order to avoid detecting collisions which occur at contact points already being tracked. If p is such a tracked point on a surface $S(u, v, t)$, the ODE solver computes a trajectory for $p = S(\tilde{u}(t), \tilde{v}(t), t)$. We then discard all global minimizers to the constrained minimization problem which satisfy

$$\|S(u, v, t) - S(\tilde{u}(t), \tilde{v}(t), t)\| < \lambda, \qquad (7)$$

where λ is a constant chosen by the user. The functions $\tilde{u}$ and $\tilde{v}$ have known representations, as computed by the solver. A natural interval extension of this constraint involving an inclusion function for S is then included in the constraint inclusion $\Box F$. An additional constraint is added for each tracked point.

4 Interval Newton Methods

In order to more quickly refine our intervals towards the solutions of the collision equality constraint $C = 0$, we make use of an interval Newton method. Interval Newton methods are applicable to the general problem of finding zeroes of a differentiable function $f\colon \mathbf{R}^n \rightarrow \mathbf{R}^m$ in an interval $X \subset \mathbf{R}^n$. They allow us to find an interval bound on the set

$$X^* = \{x \in X \mid f(x) = 0\}$$

Let $Z(X)$ be such a bound (i.e., $X^* \subset Z(X)$). We can reduce the size of our candidate region X by[13]

$$X' = X \bigcap Z(X)$$

In particular, $Z(X) \bigcap X = \emptyset$ implies that X contains no solutions. We call the operator $Z(X)$ the *interval Newton operator*.

Since X can only decrease in size after it is intersected with $Z(X)$, this procedure can be applied iteratively to produce smaller and smaller regions, as in

$$X_{i+1} = (X_i \bigcap Z(X_i))$$

Note however that a smaller region is not *necessarily* produced. Interval Newton methods should therefore be combined with bisection. When interval Newton iteration is effective at reducing the size of X its use is continued. Otherwise, bisection subdivision is performed.

The following sections present three methods for computing $Z(X)$:

- use of the Krawczyk-Moore form (Section 4.1)
- use of the interval inverse (Section 4.2.1)
- use of matrix iteration (Section 4.2.2)

In each case, we modify the constrained minimization algorithm from Figure 6 by replacing the subdivide step with the code shown in Figure 7.

4.1 Fixed Point Methods: the Krawczyk–Moore Form

The familiar (point-wise) Newton's method is used to converge on the solution to a system of equations $f(x) = 0$ where $f\colon \mathbf{R}^n \rightarrow \mathbf{R}^n$.

[13] $A \bigcap B$ denotes the interval formed by the intersection of the intervals A and B.

```
Newton($Y$)
compute interval Newton step on $Y$
if step succeeds then
        $Y' \leftarrow Y \bigcap Z(Y)$
        if $Y' = \emptyset$, discard $Y$
                (proceed with next region)
        else if $Y'$ is sufficiently smaller than $Y$, insert $Y'$ into $L$
                (proceed with next region)
        else subdivide $Y'$
                (continue with $Y$ replaced by $Y'$)
else subdivide $Y$
                (continue with $Y$)
```

Figure 7: Interval Newton Modification to the Constrained Minimization Algorithm: The above algorithm replaces the subdivide step in the algorithm of Figure 6.

The method starts with an initial "guess" at a solution x_0 and iterates via

$$x_{i+1} = p(x_i)$$

where

$$p(x) = x - Yf(x).$$

Y is a nonsingular $n \times n$ matrix which in straightforward Newton's method is the inverse of the Jacobian matrix of f at x, i.e.

$$Y \equiv J^{-1}(x)$$

Under certain conditions, this iterative procedure converges to a fixed point x^*. If convergence is achieved, then the fixed point x^* is a solution since

$$p(x^*) = x^* \iff f(x^*) = 0$$

because Y is nonsingular.

An interval analog of this method may be developed. Let X be an interval in $\mathbf{R}^n$ in which zeroes of f are sought. We require a bound on X^*. But

$$\begin{aligned} X^* &\equiv \{x \in X \mid f(x) = 0\} \\ &= \{x \in X \mid p(x) = x\} \subset (\Box p(X) \bigcap X) \end{aligned}$$

where $\Box p(X)$ is an inclusion function for the Newton operator $p(x)$.

The Krawczyk–Moore form, $K(X, c, Y)$, provides the necessary inclusion function for the Newton operator $p(x)$. It is simply a mean value form for p (see [SNYD92c] for a discussion of the mean value form) given by

$$K(X, c, Y) \equiv c - Yf(c) + (I - Y\Box J(X))(X - c)$$

where I is the $n \times n$ identity matrix, $\Box J(X)$ is an inclusion function for the Jacobian of f evaluated on X, and c is any point in X. Note that the vector addition and subtraction and the matrix/vector multiplication operations used in K must be computed using interval arithmetic.

We therefore have

$$X^* \subset (X \bigcap K(X, c, Y))$$

for any $c \in X$ and nonsingular matrix Y. Thus, $K(X, c, Y)$ can be used as an interval Newton operator. Fairly good results can be achieved with $c = \mathrm{mid}(X)$ and $Y = J^{-1}(c)$ [TOTH85,MITC92].

In our research, we have found a different method to be superior, described in the next section.

4.2 Linear Interval Equation Methods

A second method for finding an interval bound on X^* involves solving the interval analog of a linear equation.

Let the coordinates of x be $x_1, x_2, \ldots, x_n$. By the Mean Value Theorem, given a $c \in X$, for each $x \in X$, there exist n points, $\xi_1, \xi_2, \ldots, \xi_n$ such that

$$f(x) = f(c) + J(\xi_1, \ldots, \xi_n)(x - c),$$

where the jacobian matrix J is given by

$$J_{ij}(\xi_1, \xi_2, \ldots, \xi_n) = \frac{\partial f_i}{\partial x_j}(\xi_i)$$

and where each $\xi_i \in X$. Let $\Box J$ be an inclusion function for the Jacobian matrix of f, i.e.,

$$\Box J(X) \equiv \left\{ J \mid J_{ij} \in \Box \frac{\partial f_i}{\partial x_j}(X) \right\}$$

If x is a zero of f, then there exists $J \in \Box J(X)$ such that

$$f(x) = 0 = f(c) + J(x - c).$$

Therefore, if $Q(X)$ is the set of solutions

$$Q(X) \equiv \{x \mid f(c) + J(x - c) = 0 \text{ for some } J \in \Box J(X)\},$$

then $Q(X)$ contains all zeroes of f in X.

To compute an interval bound, Z, on $Q(X)$, let $y = x - c$, and let Z' be an interval bound on the set

$$\{y \mid Jy = -f(c) \text{ for some } J \in \Box J(X)\}.$$

Then the interval Z defined using interval addition as

$$Z \equiv Z' + [c, c],$$

is an interval bound on $Q(X)$. Thus, computing the interval Newton bound Z can be accomplished by solving an interval linear equation of the form

$$Mx = b$$

where $M \equiv \Box J(X)$ is an $n \times n$ interval matrix, and $b \equiv -f(c)$ is an interval vector.[14] Stated another way, we require a bound on the set

$$Q(M, b) \equiv \{x \mid \exists \mathcal{M} \in M, \beta \in b \text{ such that } \mathcal{M}x = \beta\}.$$

The next two sections discuss two methods for solving these interval linear equations.

4.2.1 Solving the Interval Linear Equation with the Interval Inverse

One method to bound the set of solutions to the interval linear equation involves computing the interval inverse. We seek a bound on $Q(M, b)$: the set of solutions, x, for $Mx = b$. If M is an $n \times n$ interval matrix, an interval that bounds $Q(M, b)$ is

$$Z \equiv M^{-1}b$$

[14]In this case, the interval b has a lower bound equal to its upper bounds in each coordinate (called a *point* interval), neglecting inaccuracies in the computation of f.

where M^{-1} is the *interval inverse* of the interval matrix. Assuming M contains no singular matrices, the interval inverse is an interval bound on the set

$$\{m^{-1} \mid m \in M\}$$

A simple way of computing the interval inverse is to use the interval analog of LU decomposition. That is, we take the LU decomposition algorithm [PRES86, pages 31–38] and replace all arithmetic operations with their corresponding interval arithmetic counterparts (see [MOOR79] for a discussion of interval arithmetic). If at any point in the iteration we attempt to divide by an interval which contains zero, then we cannot compute the interval inverse, and the interval Newton step fails (but see the next section for a way to reduce the size of candidate regions without using the interval inverse). After enough iterations of the constrained minimization algorithm, and assuming the conditions discussed in Section 5 hold, candidate regions are usually small enough to make this technique effective.

4.2.2 Solving the Interval Linear Equation with Matrix Iteration

Another method to bound the set of solutions to the interval linear equation involves matrix iteration. The algorithm we present here requires an initial bound on the set of solutions; that is it finds a bound on the set $Q(M, b) \bigcap X$ where X is a given interval. The algorithm is therefore effective at reducing the size of a candidate interval in which solutions to an equality constraint are sought, but cannot be used to verify solution existence using the theorems in Section 5.[15]

Figure 8 contains the code for Linear_Solve, which finds bounds on the solution to the linear interval equation. Linear_Solve is based on the observation that the i-th equation of the linear system $Mx = b$:

$$M_{i1}x_1 + M_{i2}x_2 + \cdots + M_{in}x_n = b_i$$

implies, for each j such that $M_{ij} \neq 0$, that

$$x_j = \frac{b_i - \sum_{k \neq j} M_{ik}x_k}{M_{ij}}.$$

The interval analog of this equation may therefore be used for each interval matrix entry, M_{ij}, which does not contain 0 to find a bound on one of the variables x_j. This bound is intersected with the old bound on x_j yielding an interval which is possibly smaller but no larger than it was. Reducing the size of one interval may then further reduce the sizes of others as the iteration proceeds. Note that the algorithm does not halt when an interval element of M contains 0; it just proceeds to the next element which excludes 0.

An important property of Linear_Solve is that it can be applied to a nonsquare linear equation,[16] and is therefore useful in the "overconstrained" equality constraint for implicit surfaces, and the vertex-to-edge and vertex-to-vertex tangency situations of piecewise parametric surfaces (see Appendix A). Linear_Solve can be applied in many situations where LU decomposition fails because of the singularity of the interval Jacobian matrix. Even when the Jacobian matrix is singular *at the solution point,* Linear_Solve is usually effective at reducing the widths of some of the input variables. These features are critical in making the singular situations described in Section 5 computationally tractable.[17]

[15]This is because, unlike the technique presented in Sections 4.2.1, this technique does not bound $Q(M, b)$ directly, but instead bounds $Q(M, b) \bigcap X$.

[16]That is, the number of equations, m, is unequal to the number of variables, n.

```
Linear_Solve($M$,$b$,$x$)
repeat
    loop through rows of $M$ ($i = 1, 2, \ldots, m$)
        loop through columns of $M$ ($j = 1, 2, \ldots, n$)
            if $0 \notin M_{ij}$ then
                $x'_j \leftarrow (b_i - \sum_{k \neq j} M_{ik}x_k)/M_{ij}$
                $x_j \leftarrow x'_j \bigcap x_j$
                if $x_j = \emptyset$ return no solution
            endif
        endloop
    endloop
while there is sufficient improvement in $x$
```

Figure 8: Interval Linear Equation Solution Algorithm: This algorithm computes the interval Newton step (first statement of the algorithm in Figure 6).

The "sufficient improvement" condition mentioned in the algorithm can be implemented as

$$\mathrm{w}(x^{i+1}) \leq \alpha\, \mathrm{w}(x^i)$$

where a typical value of the improvement factor, α, is 0.9. Here x^i denotes the interval bound computed after i iterations of the repeat loop. Specifying a maximum number of repeat iterations also limits the amount of computation.

5 Termination Criteria

Two theorems in interval analysis specify conditions under which a square system of equations contains a unique solution in a region.[18]

Theorem 1 (Krawczyk–Moore Existence) If $K(X, c, Y) \subset X$, $K(X, c, Y) \neq \emptyset$, and $\|I - Y\Box J(X)\| < 1$, then there is a unique root in X, and pointwise Newton's method will converge to it.

Theorem 2 (Linear Interval Equation Existence) If $Q(X) \subset X$ and $Q(X) \neq \emptyset$, then there is a unique root in X.

The conditions implied by these theorems thus lead to acceptance criteria, A, for the constrained minimization algorithm. Implementation of Theorem 1's conditions is clear from the discussion in Section 4.1. To verify the conditions of Theorem 2, we use the interval inverse method discussed in Section 4.2.1. We have been able to verify solution uniqueness much earlier (i.e., in larger regions) using Theorem 2's test.

We note the conditions for Theorems 1 and 2 can only be verified when the determinant of the Jacobian of the equality constraint function, C, is nonzero in some neighborhood of the solution. For collision detection, the Jacobian determinant is zero at a solution to the contact and tangency constraints in the following situations:

- the contacting surfaces become tangent but never interpenetrate. They can even stay tangent for an interval of time.

[17]We prefer the method of matrix iteration described here to a faster method (the interval analog of Gauss-Seidel iteration) which involves solving only for the diagonal matrix elements after a preconditioning step (see [RATS88]). This method requires a square system of equations, and will fail when the Jacobian matrix is singular at the solution. Interval computations in the preconditioning step also have the effect of increasing the size of the solution set $Q(M, b)$ even before any iteration takes place.

[18]See [TOTH85] for a proof sketch and references for Theorem 1, [SNYD92b] for a proof of Theorem 2.

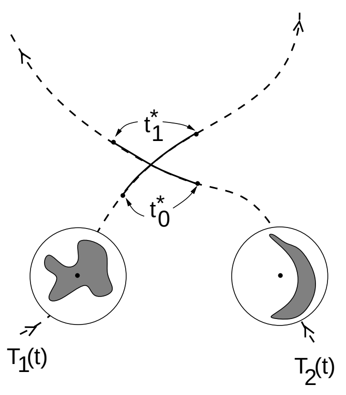

Figure 9: Bounding Sphere Collision Time Bound

- the surfaces contact instantaneously on a curve or higher-dimensional region.
- both — the surfaces contact at an infinite set of points through an interval of time.

In these cases, we can not verify that a unique solution exists and must resort to heuristic criteria (the "emergency" criteria of the constrained minimization algorithm). That is, we consider surfaces to have collided when a bound on C is sufficiently small in a region.[19]

If a solution is on the boundary of a candidate region, we note that the conditions of either theorem will be difficult to verify since the set result (e.g., $Q(X)$) will always slightly spill out of the original region X. To solve this problem, we should slightly increase the region examined for acceptance, X, via

$$X' \equiv c + (X - c)(1 + \gamma)$$

where γ is a small constant (like .1). We then can perform the test on this bigger region X'. Then, even if the solution is on the boundary of X, the theorem conditions will eventually be satisfied if the Jacobian of C is nonsingular in a neighborhood of the solution.

6 A Simple Bound for the Time of Collision

We can save time in the collision algorithm by using a fast algorithm to reduce the time interval over which collisions are searched. This time bound may also tell us, with a minimum of computation, if the two surfaces fail to intersect, obviating the need for further computation. The test presented here uses 1D minimization (only over time) rather than minimization over the 4 or 5 dimensional space required to solve the full problem.

As shown in Figure 9, we compute two bounding spheres around each of our parametric surfaces. In the case of rigid motion, this may be computed beforehand as a preprocessing step. We specify two points for each parametric surface O_1 and O_2 about which to compute a bounding sphere. These points should be chosen to minimize the size of the resulting bound; using the center of mass of the surface is a good choice. The bounding radii are:

$$R_1 \equiv \max_{(u_1,v_1)} \|s_1(u_1, v_1) - O_1\|$$
$$R_2 \equiv \max_{(u_2,v_2)} \|s_2(u_2, v_2) - O_2\|.$$

Note that these bounding radii can be computed using a 2D unconstrained minimization problem, for which the algorithm of Section 3.2 is suitable.

Using the constrained minimization algorithm, this time on a simple 1D problem, we then find a bound on the time of collision via

$$t_0^* \equiv \underset{t\in[t_0,t_1]}{\text{minimum}}\{\ t \mid \|T_1(t) + O_1 - T_2(t) - O_2\| \le R_1 + R_2\}$$
$$t_1^* \equiv -\underset{t\in[t_0,t_1]}{\text{minimum}}\{-t \mid \|T_1(t) + O_1 - T_2(t) - O_2\| \le R_1 + R_2\}.$$

We can then replace the $[t_0, t_1]$ interval in the full collision minimization problem for that pair of surfaces with $[t_0^*, t_1^*]$, or cull the pair of surfaces if no solutions to the 1D problem are found.

7 The Full Collision Algorithm

The complete algorithm for detecting collisions can now be described. The following discussion pertains to a set of parametric surfaces; a similar algorithm can be developed for the case of implicit surfaces or parametric/implicit combinations. We are given a set of solids defined by a parametric boundary representation, and a time interval in which to detect collisions, $[t_0, t_1]$. The following steps summarize the final algorithm:

1. Detect pairs of objects which can possibly collide. For this step, we bound each time-varying surface by evaluating an inclusion function for its time-varying mapping over $[t_0, t_1]$. More precisely, a bounding box through time on the surface $S(u, v, t)$ is given by
$$\Box S([0, 1], [0, 1], [t_0, t_1])$$
assuming S is evaluated on the unit square.[20] We can then test whether any of the resulting bounding boxes intersect using highly efficient algorithms from computational geometry [SIX82]. All pairs of bounding boxes which do intersect must be processed further; the rest are culled.
2. For rigid bodies, additional object pairs can be culled using the bounding sphere test of Section 6. A variant of this test can also be used for deformable surfaces.
3. If any pairs of objects remain to be processed, we must invoke the full constrained minimization algorithm. Here, we distinguish between "free" objects and objects already in continuous contact, whose contact points are being tracked with the ODE solver. For objects already in continuous contact, additional constraints are added (Section 3.2) to prevent re-detection of the tracked points. All such problems are placed on the initial priority queue of the multiple element constrained minimization algorithm.
4. We use local methods, such as Newton's method, to converge to the actual collision point in each solution region which contains an isolated collision (i.e., for which the interval existence and uniqueness test of Section 5 succeeded). We arbitrarily choose the midpoint as the collision point for the rest of the solution regions (termed singular solutions in Section 3).

[19]This implies, because of the contact constraint, that the surfaces come within a specified constant.

[20]Note that for rigid surfaces we can cache a bounding box on the time-independent rigid surface and compute only a bound over time on the resulting rotated and translated bounding box.

8 Implementing Inclusion Functions

The collision detection algorithm depends on inclusion functions for the time-varying surfaces and their various derivatives. Note that the equality constraint for the parametric surface case (Equation 2) involves derivatives of the time-varying surface mappings $S_i(u_i, v_i, t)$. The interval Newton method then requires an additional derivative of the equality constraint with respect to each of the independent variables. Interval analysis provides the necessary theory for constructing inclusion functions for these functions.

For simple polynomial surfaces (e.g., bicubic patches or algebraic surfaces) interval arithmetic suffices to provide an inclusion function for the time-independent surface. Toth [TOTH85] has presented efficient inclusion functions for Bezier surfaces. Mitchell and Hanrahan have proposed a simple stack-based representation of surfaces which allows generation of inclusion functions for the surface and its derivatives [MITC92]. Inclusion functions for more complicated surfaces and their derivatives can also be constructed. We have used the system described in [SNYD92a,SNYD92b], which automates the construction of inclusion functions (and inclusion functions for the derivatives) of any functions formed by the composition of a quite powerful set of symbolic operators.

For physical simulations, the ODE solver computes a representation of the time behavior of the surfaces. The solver may directly compute a continuous representation or it may be later reconstructed by point sampling the solver's results, typically producing a polynomial. Appendix B discusses a method to bound Chebyshev polynomials.

9 Results

We have successfully tested this method on a series of collision detection examples, including both rigidly moving and deforming objects. For example, Figure 12 shows the results of a difficult collision detection run in which the contact manifold forms a series of disjoint 2D regions. A collection of 59 points was generated in the contact region with a simultaneity threshold of 0.001 and solution separation distance of 0.04, using 28704 iterations and 88.81 CPU seconds.[21] While the running time may seem large, the problem itself is sufficiently difficult that its running time exceeded our threshold of 8 CPU hours without the use of *every* new technique presented in this paper: adding the tangency constraint (rather than using the contact constraint alone), sorting by upper bound in the constrained minimization algorithm (rather than by lower bound), and using Linear_Solve for the interval Newton step (rather than the Krawczyk–Moore operator). Figure 1, 5, and 12–16 show the results of the algorithm for several different time-varying shapes.

The table in Figure 10 compares running times for a second example involving two rotating and translating bumpy parametric surfaces which collide at an isolated point. Several solution methods are compared: LEQN (interval Newton using the linear equation solution techniques of Section 4.2), KM (interval Newton using the Krawczyk–Moore operator), NIN (without interval Newton), and NTAN (without the tangency condition). Since the collision occurs at an isolated point, both the LEQ and KM methods were able to accept a single solution region by verifying the solution existence and uniqueness test. The other methods required an accuracy parameter for acceptance; we used the simple criterion $w(X) < \gamma$.

Running Times		
Example	Iterations	CPU (secs)
LEQN	6331	32.67
KM	10087	148.28
NIN,γ=1e-3	17395	8.58
NIN,γ=1e-4	29921	15.46
NIN,γ=1e-5	40127	21.52
NIN,γ=1e-6	48187	23.25
NIN,NTAN,γ=1e-3	52307	14.59
NIN,NTAN,γ=1e-4	587711	169.87
NIN,NTAN,γ=1e-5	3822605	1207.46

Figure 10: Table of Results for Various Methods: see Section 9

Because we used a prototype system to gather the data, we emphasize the importance of iteration count data over CPU time. Our system requires the traversal of a complicated data structure for each inclusion function evaluation which overwhelms the floating point computation actually needed in the function. The interval Newton methods are sensitive to this bias, since their implementation required many symbolic operators. We believe the iteration counts shown here to be a reasonable measure of expected running time, if the inclusion functions are hand-coded for the surfaces of interest.

10 Conclusions

We have presented a robust interval algorithm that can detect collisions between complex curved surfaces. The algorithm handles a greater range of situations than previous algorithms. It detects both isolated collision points and collision points on contact manifolds. It can avoid detection of points close to a set of tracked points with specified trajectories. It efficiently handles detection of simultaneous collisions between sets of moving objects. The technique is practical for simulations involving large numbers of moving and deforming objects (see Figures 15 and 16).

We draw several conclusions from our experimental results. First, interval methods, such as [VONH90] and [DUFF92], which do not make use of the interval Newton method or the tangency condition soon become impractical as we increase the accuracy parameter (refer to the NIN,NTAN lines of the table in Figure 10). Interval Newton iteration combined with the tangency condition (especially using the interval linear equation approach) is very effective at reducing computation. Second, our method can solve the difficult problem of detecting collision points on a contact manifold. We have found the methods described here to be indispensable, including the idea of the tangency constraint, the constrained minimization algorithm discussed in Section 3, and the interval linear equation approach to interval Newton iteration.

We note that many areas for improvement remain. Sorting by lower bound of the objective function rather than by upper bound is more efficient for isolated point collisions. We have noted an efficiency gain of a factor of from 1 to 10 in using the lower bound for such cases. On the other hand, sorting by lower bound is completely impractical for detecting collisions on a contact manifold. If we know the nature of the collision solution set a priori, we can choose the appropriate method. Alternatively, combining the two approaches, perhaps by "racing" them in parallel on the same problem, may decrease the average running time. We are studying several ways to increase efficiency that involve more optimally

[21]The term *iteration* refers to an evaluation of the inclusion functions $\Box f$ and $\Box F$ (objective function and constraint function) in the constrained minimization algorithm. All CPU times are measured on a HP 9000 Series 750 computer.

choosing the next dimension to subdivide, and determining a subdivision location other than the midpoint.

Acknowledgments

We wish to thank Mark Montague and Allen Corcorran for their production help. Thanks also go to the Siggraph reviewers for their careful reading, suggestions, and bug fixes. This work was supported in part by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219), as part of the NSF/DARPA STC for Computer Graphics and Scientific Visualization. All opinions, findings, conclusions, or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

References

[ALEF83] Alefeld, G., and J. Herzberger, *Introduction to Interval Computations,* Academic Press, New York, 1983.

[BARA89] Baraff, David, "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies," Computer Graphics, 23(3), pp. 223-232, July 1989.

[BARA90] Baraff, David, "Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation," Computer Graphics, 24(4), pp. 19-28, August 1990.

[BARA91] Baraff, David, "Coping with Friction for Non-penetrating Rigid Body Simulation," Computer Graphics, 25(4), pp. 31-39, July 1991.

[BARA92] Baraff, David, and A. Witkin, "Dynamic Simulation of Non-penetrating Flexible Bodies," Computer Graphics, 26(2), pp. 303-308, July 1992.

[DUFF92] Duff, Tom, "Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry," Computer Graphics, 26(2), July 1992, pp. 131-138.

[META92] Metaxas, Dimitri, and D. Terzopoulos, "Dynamic Deformation of Solid Primitives with Constraints," Computer Graphics, 26(2), pp. 309-312, July 1992.

[MITC92] Mitchell, Don, and P. Hanrahan, "Illumination from Curved Reflectors," Computer Graphics, 26(2), July 1992, pp. 283-291.

[MOOR66] Moore, R.E., *Interval Analysis,* Prentice Hall, Englewood Cliffs, New Jersey, 1966.

[MOOR79] Moore, R.E., *Methods and Applications of Interval Analysis,* SIAM, Philadelphia.

[MOOR80] Moore, R.E., "New Results on Nonlinear Systems," in *Interval Mathematics 1980*, Karl Nickel, ed., Academic Press, New York, 1980, pp. 165-180.

[MOOR88] Moore, M. and Wilhelms, J., "Collision Detection and Response for Computer Animation," Computer Graphics, 22(4), pp. 289-298, August 1988.

[PRES86] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes,* Cambridge University Press, Cambridge, England, 1986.

[RATS88] Ratschek, H. and J. Rokne, *New Computer Methods for Global Optimization,* Ellis Horwood Limited, Chichester, England, 1988.

[SCLA91] Sclaroff, Stan, and A. Pentland, "Generalized Implicit Functions for Computer Graphics," Computer Graphics, 25(4), pp. 247-250, July 1991.

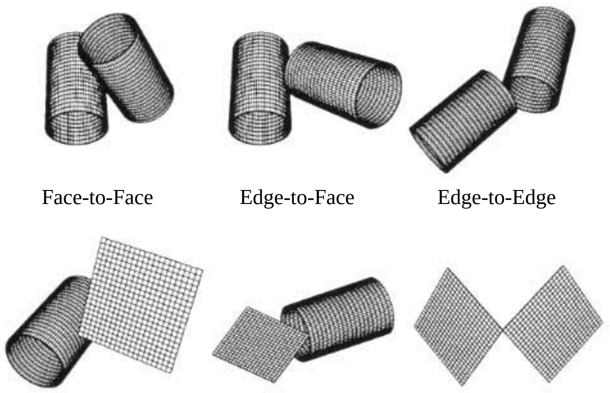

Figure 11: Types of Collision Tangency

[SIX82] Six, H.W., and D. Wood, "Counting and Reporting Intersections of *d*-Ranges," IEEE Transactions on Computers, C-31(3), March 1982, pp. 181-187.

[SNYD92a] Snyder, John, and J. Kajiya, "Generative Modeling: A Symbolic System for Geometric Modeling," Computer Graphics, 26(2), pp. 369-378, July 1992.

[SNYD92b] Snyder, John, *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design Using Interval Analysis*, Academic Press, Cambridge, MA, July 1992.

[SNYD92c] Snyder, John, "Interval Analysis for Computer Graphics," Computer Graphics, 26(2), pp. 121-130, July 1992.

[TOTH85] Toth, Daniel L., "On Ray Tracing Parametric Surfaces," Computer Graphics, 19(3), July 1985, pp. 171-179.

[VONH90] Von Herzen, B., A.H. Barr, and H.R. Zatz, "Geometric Collisions for Time-Dependent Parametric Surfaces," Computer Graphics, 24(4), August 1990, pp. 39-48.

A Collision Constraints for Piecewise Parametric Surfaces

A piecewise surface is composed of a set of smooth *faces*, a set of *edges* where these faces meet, and a set of *vertices* or points where edges meet. Edges form the 1D boundaries over which the surface is not smooth; vertices are the 0D boundaries between smooth edge curves. A data structure containing the faces, edges, and vertices of a solid is called its *boundary representation*. For example, the boundary representation of a cylindrical solid contains three faces: one cylinder and two circular endcaps, two edges where the cylinder and endcap meet, but no vertices.

To detect collisions between two piecewise surfaces, we must search for collisions between each pair of faces, between edges and faces, between vertices and faces, etc. The constraints governing collisions are different in each of these cases, which we call *tangency situations*. There are 6 types of tangency conditions in a collision between piecewise surfaces as shown in Figure 11. Constraints for the face-to-face tangency situation are identical to the constraints discussed in Section 2.1. The following paragraphs discuss the other tangency situations.

We must combine all the constrained minimization problems for the various possible types of tangency situations. For example, if the surfaces are a pair of cylindrical solids, we obtain 25 separate constrained minimization subproblems: 9 face-to-face problem, 12 edge-to-face problems, and 4 edge-to-edge problems. Two tori require only a single face-to-face problem. Each problem is then solved simultaneously using the multiple element constrained minimization algorithm.

Edge-to-Face For the *edge-to-face* case, we have an edge curve, $C(s,t)$, which forms a boundary of a surface, $S_a(u_a, v_a, t)$, and another surface, $S(u,v,t)$. The edge curve is typically formed by evaluating a parametric surface S_a at a specific value for either the u or v parameter, e.g.

$$C(s,t) \equiv S_a(s, v^{\text{fixed}}, t)$$

where v^{fixed} is a constant set at one of the extremes of the v interval over which S_a is evaluated. The contact constraint for edge-to-face collisions is

$$C(s,t) - S(u,v,t) = 0 \tag{8}$$

and the tangency constraint by

$$\frac{\partial C}{\partial s}(s,t) \cdot N(u,v,t) = 0 \tag{9}$$

where N is the time-varying normal to the surface S. The edge-to-face equality constraint can be represented a system of 4 equations in 4 variables.

To define the incoming collision condition, we need to define what "outwardness" means on an edge curve. Assuming all surfaces form the valid boundaries of a closed solid, the edge curve $C(s,t)$ is shared between two surfaces S_a and S_b. We can therefore define two "outward" directions, given by the outward pointing normals to the shared surfaces S_a and S_b. For example, these outward directions may be defined as

$$\begin{aligned} C^{\text{outward-1}}(s,t) &\equiv N_a(s, v^{\text{fixed}}, t) \\ C^{\text{outward-2}}(s,t) &\equiv N_b(u^{\text{fixed}}, s, t) \end{aligned}$$

where N_a and N_b are the outward normal vectors of the respective surfaces.

The incoming constraint forces the relative velocity between the surface and the edge curve to be in the same direction (using a dot product test) as the surface's normal. The surface's normal must also face away from at least one of the edge curve's outward directions. The incoming constraint is:

$$\begin{aligned} &\left(\frac{\partial S}{\partial t}(u,v,t) - \frac{\partial C}{\partial t}(s,t)\right) \cdot N(u,v,t) \geq 0 \quad \text{and} \\ &\Big(-N(u,v,t) \cdot C^{\text{outward-1}}(s,t) \geq 0 \quad \text{or} \\ &\quad -N(u,v,t) \cdot C^{\text{outward-2}}(s,t) \geq 0\Big) . \end{aligned} \tag{10}$$

Edge-to-Edge The *edge-to-edge* case involves two edge curves, $C_1(s_1, t)$ and $C_2(s_2, t)$. For this case, just a contact constraint is sufficient, given by the following system of three equations in three variables:

$$C_1(s_1, t) - C_2(s_2, t) = 0. \tag{11}$$

To define the incoming collision condition, we define two outward directions for each edge curve, as in the previous discussion for the edge-to-face case. The relative velocity between the edge curves must face in the same direction as at least one of the first curve's outward directions:

$$\begin{aligned} &\left(\frac{\partial C_1}{\partial t}(s_1,t) - \frac{\partial C_2}{\partial t}(s_2,t)\right) \cdot C_1^{\text{outward-1}}(s_1,t) \geq 0 \quad \text{or} \\ &\left(\frac{\partial C_1}{\partial t}(s_1,t) - \frac{\partial C_2}{\partial t}(s_2,t)\right) \cdot C_1^{\text{outward-2}}(s_1,t) \geq 0 \end{aligned} \tag{12}$$

Also, at least one of the outward directions on one curve must face away from one of the outward directions of the other curve. A logical combination of 6 inequalities is the result.

Vertex-to-Face, Vertex-to-Edge, Vertex-to-Vertex The *vertex-to-face* case involves a vertex $P(t)$ and a surface $S(u,v,t)$. As in the edge-to-edge case, a contact constraint is sufficient, of the form

$$P(t) - S(u,v,t) = 0. \tag{13}$$

where the point P is formed by evaluating a surface at a fixed point in its (u,v) parameter space, e.g.

$$P(t) \equiv S_a(u^{\text{fixed}}, v^{\text{fixed}}, t)$$

A system of three equations in three unknowns results. Similarly, a system of three equations in two unknowns results for the vertex-to-edge case, and a system of three equations in a single unknown for the vertex-to-vertex case.

The incoming collision condition can be derived by defining a number of outward directions for the colliding vertex, corresponding to the normal vector of each surface containing that vertex. The normal to the surface S must face away from at least one of these outward directions, as in the edge-to-face case. The relative velocity between the surface and the vertex must face in the same direction as the surface's normal, via

$$\left(\frac{\partial S}{\partial t}(u,v,t) - \frac{\partial P}{\partial t}(t)\right) \cdot N(u,v,t) \geq 0.$$

Similar systems of inequalities can be derived for situations where a vertex collides with an edge or another vertex.

B Inclusion Functions for Chebyshev Polynomials

Chebyshev polynomials are a good basis for a continuous representation of time behavior. They allow simple control of approximation error, and can be differentiated using a simple method to produce a Chebyshev representation of the derivative (see [PRES86, pages 158–165] for a discussion of the advantages of Chebyshev polynomials, their properties, and algorithms for their manipulation). The basis functions for a Chebyshev polynomials are

$$T_n(x) \equiv \cos(n \arccos(x))$$

which expand to a series of polynomials of the form

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ &\vdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \quad n \geq 1 \end{aligned}$$

The function $T_n(x)$ has $n+1$ extrema with values of ± 1 at the locations

$$x_i \equiv \cos\left(\frac{\pi i}{n}\right) \quad i = 0, 1, \ldots, n$$

The i-th extremum of the basis function T_n is either a minimum or maximum according to the rules

$$T_n(x_i) = \begin{cases} -1, & \text{if } (i+n) \equiv 1 \bmod 2 \\ +1, & \text{if } (i+n) \equiv 0 \bmod 2 \end{cases}$$

A Chebyshev approximation of order N is given by specifying N coefficients c_i, $i = 0, 1, \ldots, N-1$, which determine the polynomial

$$C(x) = \sum_{i=1}^{N-1} c_i T_i(x) + \frac{c_0}{2}$$

Given the order of the Chebyshev approximation function $C(x)$, N, we can easily compute an inclusion function for $C(x)$. Let the interval over which we are to bound $C(x)$ be given by $X = [x_0, x_1]$. As a preprocessing step, we first tabulate the locations of the extrema of the basis functions, up to some maximum order. (Note that the results can then be used for any approximating polynomial.) For each Chebyshev basis function, $T_i(x)$, $i = 0, 1, \ldots, N-1$, we first evaluate $T_i(x_0)$ and $T_i(x_1)$. We then determine whether any extrema of $T_i(x)$ occur in $[x_0, x_1]$ using the tabulated locations of the extrema. A lower bound on the basis function over $[x_0, x_1]$, b_i^0 is

$$b_i^0 \equiv \begin{cases} \min(T_i(x_0), T_i(x_1), -1), & \text{if min of } T_i(x) \in [x_0, x_1] \\ \min(T_i(x_0), T_i(x_1)), & \text{otherwise.} \end{cases}$$

Similarly, an upper bound is

$$b_i^1 \equiv \begin{cases} \max(T_i(x_0), T_i(x_1), 1), & \text{if max of } T_i(x) \in [x_0, x_1] \\ \max(T_i(x_0), T_i(x_1)), & \text{otherwise.} \end{cases}$$

The final inclusion function is then

$$\Box C(X) \equiv \sum_{i=1}^{N-1} c_i [b_i^0, b_i^1] + \frac{c_0}{2}$$

where operations are computed with interval arithmetic.

Scenes from test animations: In the following figure pairs, the upper image is the scene immediately before the collision, while the bottom image is the scene at the collision time. Points of contact are shown as white dots, which are uniformly distributed over regions where there are line and surface contacts. At the time of collision, surfaces become transparent to make the dots visible.

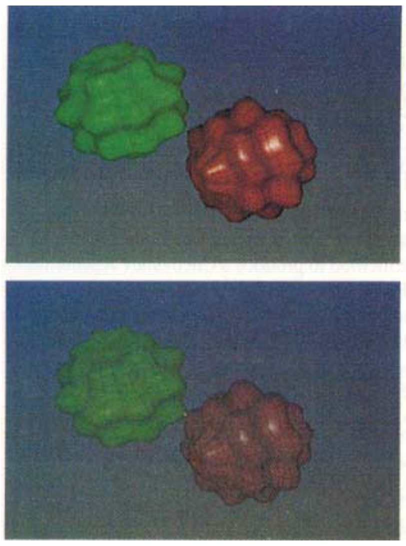
Figure 12: Two bumpy objects collide at one point.

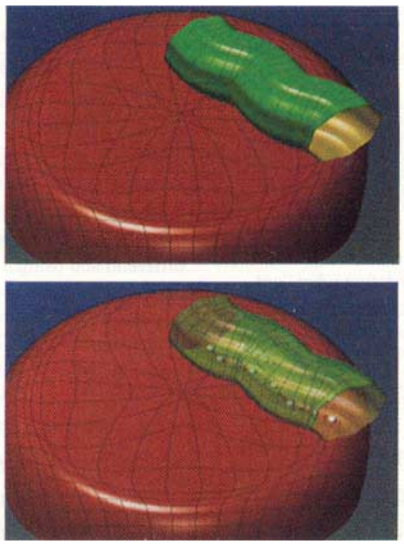
Figure 13: A time-varying tube contacts a cushion along a curve.

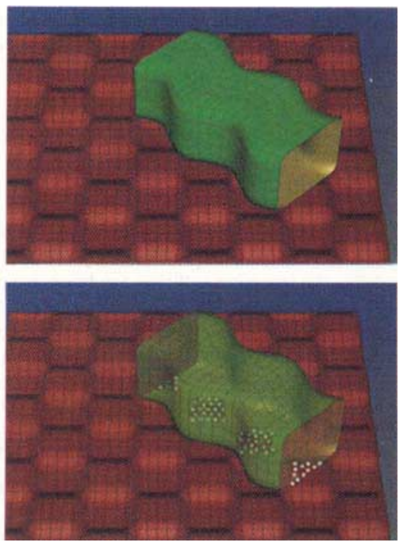
Figure 14: A wavy object contacts a raised checkerboard floor in several flat patches.

Scenes from "Fruit Tracing": This animation shows the results of collision detection for a more complicated setting involving hundreds of colliding objects. In this animation, moving parametric surfaces representing fruit are collided with a static lobster shape, defined as an implicit surface. (Lobster data generated by David Laidlaw, Matthew Avalos, Caltech, and Jose Jimenez, Huntington MRI Center.)

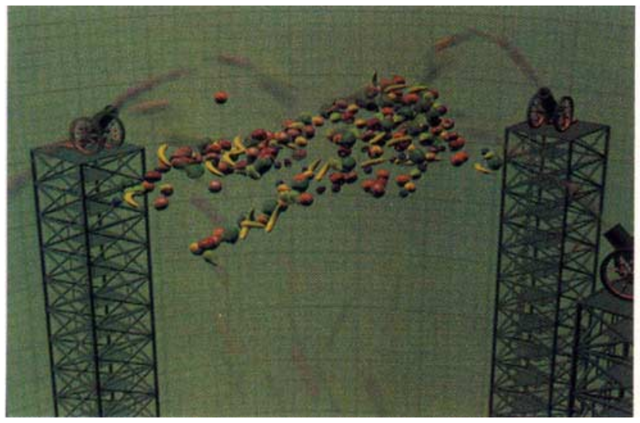
Figure 15: Colliding dynamic fruits.

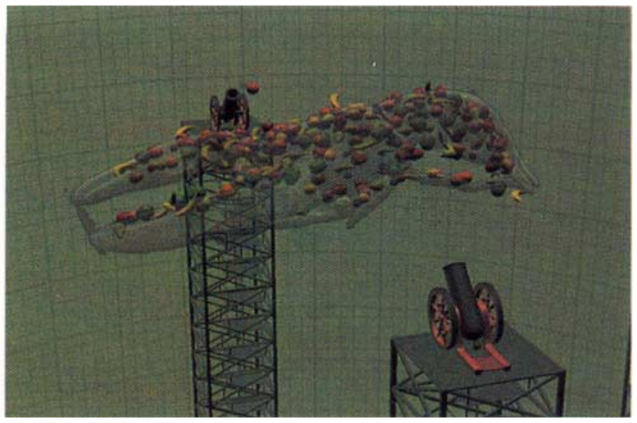
Figure 16: Scene showing lobster shape.