Geometric Collisions for Time-Dependent Parametric Surfaces

Brian Von Herzen, Alan H. Barr, and Harold R. Zatz

California Institute of Technology
Pasadena, CA 91125

Abstract

We develop an algorithm to detect geometric collisions between pairs of time-dependent parametric surfaces. The algorithm works on surfaces that are continuous and have bounded derivatives, and includes objects that move or deform as a function of time. The algorithm numerically solves for the parametric values corresponding to coincident points and near-misses between the surfaces of two parametric functions.

Upper bounds on the parametric derivatives make it possible to guarantee the successful detection of collisions and near-misses; we describe a method to find the derivative bounds for many surface types. To compute collisions between new types of surfaces, the mathematical collision analysis is needed only once per surface type, rather than analyzing for each pair of surface types.

The algorithm is hierarchical, first finding potential collisions over large volumes, and then refining the solution to smaller volumes. The user may specify the desired accuracy of the solution. A C-code implementation is described, with results for several non-bicubic and bicubic time-dependent parametric functions. An animation of the collision computation demonstrates collisions between complex parametric functions.

CR Categories: I.3.5—Computational Geometry and Object Modeling; I.3.7—Three-Dimensional Graphics and Realism
Additional Keywords: Collision Detection, Parametric Surfaces, Adaptive Sampling, Simulation, Dynamics, Constraints, Deformations, Computer Modeling.

1 Introduction

In computer animation and physical simulation it is frequently important for objects to interact with one another. One form of interaction between objects is a collision, which is initiated by geometric contacts that arise between two or more bodies. We distinguish the geometric contact of the objects from the forces that influence the motion of the objects after the collision; we call these contacts the *geometric* part of the collision.

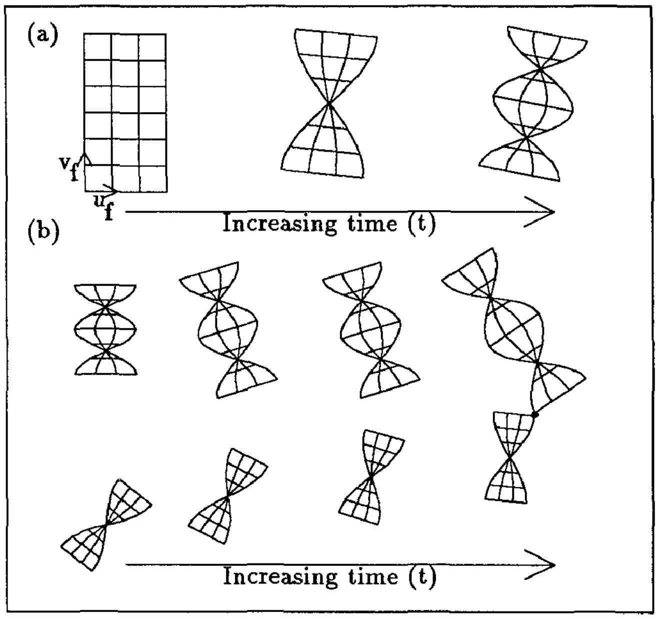

Figure 1: Geometric collision of time-dependent parametric surfaces. (a) A time-dependent parametric surface $\vec{f}(u,v,t)$. (b) A pair of deforming surfaces $\vec{f}(u_g,v_g,t)$ and $\vec{g}(u_g,v_g,t)$ that collide at time $t_{\min}$. The algorithm returns the parameters u_f, v_f, u_g, v_g, and $t_{\min}$ corresponding to the collision point on the two surfaces.

In the physical world, collisions occur when two objects move through space and hit one another. To simulate this behavior in a computer graphics environment, we need a mathematical description of the objects and a corresponding geometric collision procedure to determine that contact has occurred. Many computer graphics objects are composed of polygons; geometric collision algorithms have been developed for these (for example [Moore *et al.* 88].)

Although it is possible to represent virtually any surface with sufficient numbers of polygons, it is sometimes more convenient to use higher-level surface representations. Some people prefer to use bicubic patches for their applications because the patches can be a more compact representation than polygons, and can be easier to work with.

Just as bicubic patches are sometimes more convenient than polygons, there exist higher-level representations of parametric surfaces that at times are more convenient than bicubic patches. Examples include some forms of local and global deformations [Barr 84], [Sederberg *et al.* 86], [Snyder 90]. These surfaces are typically functions of two surface parameters, u, and v, as in $\vec{f}(u,v)$; A few intersection algorithms for these surfaces have been developed [Filip *et al.* 86]. However, no results have been reported for dynamic collisions of general time-dependent parametric functions.

1.1 Overview

In this paper we describe a collision algorithm for *time-dependent* parametric surfaces that are described by parametric functions of three arguments, u, v, and t as in $\vec{f}(u,v,t)$ (see Figures 1 and 2). These types of functions arise frequently in the context of physically-based modeling and simulation, as a body translates, rotates and possibly deforms as a function of time. In this algorithm, we compute the u and v collision parameters at the earliest time of collision $t_{\min}$.

Unfortunately, for arbitrary parametric functions it can be proven that no algorithm (based solely on function evaluation) can be constructed that is guaranteed to find the earliest time of collision (See Section 1.3). A restriction on the functions is needed in order to construct a workable collision algorithm. In this paper, we require the functions to have computable bounds on their regional rates of change. These bounds on the rates of change are called "Lipschitz" values. Such surfaces and functions with computable Lipschitz values will likely become increasingly important for computer graphics rendering, both in terms of software, but also in terms of future computer graphics hardware [Kalra *et al.* 89], [Kaufman 87], and [Von Herzen *et al.* 87].

The potential hardware applications arise from an interesting feature of the algorithm. The reader may be aware that many other algorithms for intersecting parametric surfaces use special cases: a different procedure is needed for each *pair* of surface types. For instance, the reader can imagine an algorithm that computes interactions between spheres and cylinders but does not compute interactions between other surface types (say, ellipsoids and cylinders).

Unlike the case-by-case algorithms in which a different procedure is needed for each pair of surface types, our algorithm works uniformly for all of its available surfaces. Each surface is analyzed *by itself*, to compute bounds on its rates of change (see Appendix B). From these bounds on the surface's rates of change, we can find geometric collisions with other surface types. Thus, we do not need to perform $O(N^2)$ analyses (for each possible pair of N surface types) but instead we can analyze each function once, in isolation. Then it automatically interacts with all of our previously implemented surface types with no extra work.

1.2 Problem Statement

The parametric surfaces may be considered to be vector functions of three parametric variables: $\vec{f}(u_f, v_f, t)$ and $\vec{g}(u_g, v_g, t)$, where u_i and v_i are parametric variables that span each of the surfaces, and t is time. For suitable types of surfaces, we want to find the earliest time $t_{\min}$, within bounds, such that

$$\vec{f}(u_f, v_f, t_{\min}) = \vec{g}(u_g, v_g, t_{\min}). \tag{1}$$

We also want to find u_f, v_f, u_g, and v_g at some point of first collision on each surface. We assume that the surfaces are continuous, and that they are embedded in three spatial dimensions and one temporal dimension.

In practice, we determine when the distance between objects becomes less than a tolerance γ:

$$\left\|\vec{f}(u_f, v_f, t_{\min}) - \vec{g}(u_g, v_g, t_{\min})\right\| < \gamma. \tag{2}$$

This event is termed a *γ-collision*, and includes collisions and near-misses closer than γ.

Most dynamic modeling systems [Baraff 89], [Barzel *et al.* 88] can readily utilize the approximate collision parameters available with large values of γ. The user requests a value of γ that is roughly the largest value satisfactory for the particular application (smaller values would cause the collision-detection algorithm to put in more work than necessary); this value of γ is typically much larger than the machine precision, ϵ. Thus we are able to avoid the problem of finite machine precision by explicitly using a value of γ much larger than ϵ.

Eqn. 2 represents a difficult, non-linear, 5-dimensional, root-finding problem. The algorithm based on Eqn. 2 presented in Section 4 can quickly produce results at a coarse tolerance γ, and later produce results at finer tolerances.

Sometimes the γ-collision algorithm terminates after a single sample has been taken from each surface: it becomes computationally trivial to reject potential collisions between distant objects. For additional efficiency, we develop a new method to produce bounding boxes for parametric functions, using a "Jacobian"-style matrix of Lipschitz conditions on the parametric function. This method produces much tighter bounds on the surface than does the standard Lipschitz condition, and enhances the effectiveness of the algorithm for computing collisions between parametric surfaces.

The next subsection describes some of the difficulties in detecting collisions, and potential solution methods. Section 2 describes other work in collision detection. Section 3 and Appendix A develop a new method to form a hierarchy of bounding volumes for parametric surfaces. Section 4 describes the algorithm and computational results, and Section 5 describes methods for computing Jacobian maxima for parametric surfaces, useful in bounding box formation.

1.3 Problems with Arbitrary Surfaces

The collision problem for parametric surfaces can be made arbitrarily difficult for suitably extreme parametric surfaces, such as the spike function of Figure 2. For suitably sharp spikes, finite sets of samples will probably miss the spikes completely. Finding a narrow spike becomes arbitrarily difficult as the parametric width of the spike approaches zero.

The spike problem exists in time as well as space for geometric collisions. If the location of a surface is discontinuous in time then it becomes impossible to detect collisions, because it becomes impossible to know the location of a surface over a time interval. There must be some additional constraint on a parametric surface in order to guarantee that the first collision is detectable.

1.3.1 A Method that Doesn't Work

A simplistic approach for collision detection would be to position two surfaces at time t_1 and see if they intersect, and then move the surfaces to final positions at time t_2 and see if they intersect. We could then split the time difference and sample the two surfaces at time $(t_1 + t_2)/2$, or some other time between t_1 and t_2. Recursing in this manner, we would sample the paths of the two surfaces. The problem with this technique for any finite number of samples is that

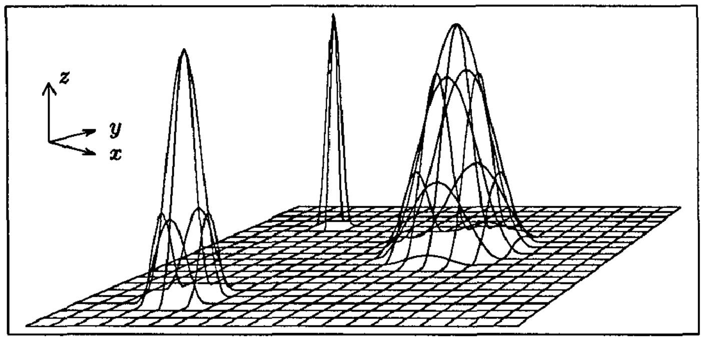

Figure 2: Parametric spike functions can be made arbitrarily sharp, so that their detection is extremely difficult. A fourth spike in this figure is invisible, since it falls between the grid points. Collision detection becomes arbitrarily difficult for such parametric surfaces. We need some other information in addition to the function values at isolated points in order to guarantee the detection of the first intersection.

we have no information about the positions of the surfaces between the sampling times. Without this information, we can never be sure that we have not missed an intersection. The problem is analogous to the spike problem of Figure 2.

1.3.2 A Method that Works

To solve the collision-determination problem, we require a constraint on the maximum velocity of any point on the surface. Similarly, we require constraints on parametric derivatives other than time. If velocity is unconstrained, then the position of a surface may be discontinuous as a function of time, and the collision determination problem is insoluble [Von Herzen 89, Appendix A.5]. With knowledge of the maximum velocity of two surfaces, we can find the first collision of the surfaces.

1.4 Solution using Lipschitz Conditions

We can construct bounding volumes of parametric surfaces with Lipschitz conditions. Given a continuous parametric surface $\vec{f}(\vec{u})$, the **Lipschitz condition** states that

$$\left\|\vec{f}(\vec{u}_2) - \vec{f}(\vec{u}_1)\right\| \le L\,\|\vec{u}_2 - \vec{u}_1\|, \qquad (3)$$

for some finite number L in some region R of $\vec{f}$. The Lipschitz condition is implied if the function $\vec{f}(\vec{u})$ has finite partial derivatives [Lin *et al.* 74, p. 58]. The Lipschitz value L is a generalization of the derivative of $\vec{f}(\vec{u})$. We can also find Lipschitz values for some surfaces that are not differentiable [Von Herzen 89, Appendix B.3]. The Lipschitz condition on a surface is sufficient to create sets of bounding volumes that are guaranteed to bound the parametric surface.

It is possible to develop a similar constraint on the temporal aspects of the collision-determination problem. We can have a moving parametric surface $\vec{f}(\vec{u})$, $\vec{u} = (u, v, t)^T$, that changes as a function of time. We can construct a set of bounding volumes for the changing surface, in a manner analogous to the method for stationary surfaces. In this case, L sets an upper bound for the velocity of the parametric surface as well as for the other parametric derivatives. This inequality is depicted graphically in Figure 3.

Given parametric functions $\vec{f}(\vec{u})$ and $\vec{g}(\vec{u})$, along with their Lipschitz values L_f, and L_g, we have proven in

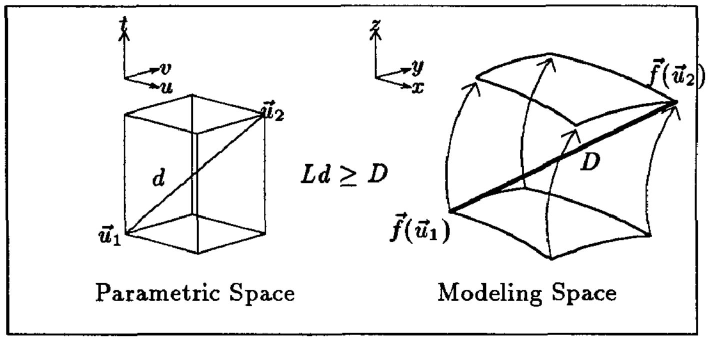

Figure 3: Graphical illustration of the Lipschitz inequality for parametric functions of three variables. If D is the distance from $\vec{f}(\vec{u}_2)$ to $\vec{f}(\vec{u}_1)$, and $d = \|\vec{u}_2 - \vec{u}_1\|$, we have $D \le Ld$, where L is a Lipschitz value for $\vec{f}$, as in Eqn. 3.

[Von Herzen 89] a method to determine the earliest collision between two surfaces. Alternatively, we can confirm that two objects do not collide. In addition, we will generalize the notion of a Lipschitz value so as to provide tighter bounding volumes for the computations.

2 Previous Work

Previous techniques have used velocity and distance bounds for collision detection of rigid objects [Culley *et al.* 86]. Upper bounds on velocity and lower bounds on distance can determine the minimum time until the next collision between objects. Here we extend the work to functions that can deform over time.

There has been some work on determining lower bounds on distance for convex polygons and polyhedra [Schwarz 81], [Cameron *et al.* 86]. A number of collision algorithms have been developed for polyhedra [Moore *et al.* 88], [Canny 84], [Hopcroft *et al.* 83], [Uchiki *et al.* 83], but collision algorithms have not been developed for more general time-dependent parametric surfaces.

Other work has developed techniques to compute the intersections of parametric functions based on derivative bounds [Filip *et al.* 86]. Their work applies to static objects that do not move as a function of time. In Section 4, we describe a method that works for time-dependent surfaces, including deformable surfaces.

The Lipschitz condition has been applied to problems in scan-conversion [Kaufman 87], ray-tracing [Kalra *et al.* 89], and adaptive sampling [Von Herzen *et al.* 87].

Recent developments in constraint methods for flexible models [Platt *et al.* 88] stress the importance of accommodating elastic and moldable objects in a physical simulation. Examples of plastic and inelastic deformations appear in recent work on modeling inelastic deformation [Terzopoulos *et al.* 88]. Collisions between flexible objects are also important for deformable animated characters [Chadwick *et al.* 89], [Going Bananas 88]. The algorithm presented in Section 4 can form a basis for a uniform environment in which varied objects may interact. The environment can accommodate rigid surfaces, bicubic patches, moving surfaces, and deforming surfaces, all within the same framework for collisions and near-misses.

Efficient collision determination involves the adaptive sampling of time-dependent parametric functions. Previous work in adaptive sampling includes [Catmull 75], [Blinn 78], [Lane *et al.* 79], [Lane *et al.* 80], [Schweitzer *et al.* 82], [Schmitt *et al.* 86], [Besl *et al.* 88], and [Von Herzen 85]. It is important to mention that the preceding articles do not deal with time at all, and therefore are not adequate for collisions of deformable time-dependent surfaces. As stated previously, the collision determination problem is insoluble for arbitrary time-dependent surfaces (Section 1.3), but in Section 4 we provide a solution for all surfaces that satisfy the Lipschitz condition, including all differentiable parametric surfaces.

The notion of an upper bound on velocity is generalized to parametric dimensions other than time (see Appendix A). We can automatically find a lower bound on the separation distance between objects, given upper bounds on the parametric derivatives of the functions. The derivative constraints enable us to sparsely sample a parametric function that deforms over time [Barr 83], [Barr 84], [Sederberg *et al.* 86], and determine γ-collisions with other objects.

3 Bounding Volumes for Time-Dependent Parametric Surfaces

We develop a set of bounding volumes for time-dependent parametric surfaces. The method presented here is general enough to determine collisions of flexible objects that change shape over time. We develop a subdivision method over parametric rectangular prisms, and traverse the parametric volumes of two surfaces to verify that they do not collide.

3.1 k-d Trees in Parametric Space

A variety of subdivision mechanisms are possible, including quadtrees of squares or bintrees of triangles [Samet 84], [Von Herzen 89]. We need a method that extends easily to k dimensions, and that controls the aspect ratio of the parametric subregions. We choose to use an alternative to the quadtree, which generalizes to k dimensions, called the k-d tree (for k-dimensional binary search tree, [Bentley *et al.* 79], [Samet 90a], [Samet 90b]). In the k-d tree method, k-dimensional space is divided into k-dimensional boxes, using planes perpendicular to each of the parametric axes. Each subdivision level splits the k-dimensional box along one of the dimensions to form two descendant boxes (Figure 4).

3.2 Lipschitz Bounding Spheres

We can construct bounding spheres from the Lipschitz equation. Figure 5 shows the bounding sphere for a parametric region R, and its corresponding projection in modeling space. The radius r of the bounding sphere in modeling space is given by $r \geq L(\Delta u + \Delta v + \Delta t)$. A sufficient value for L is

$$L \geq \max_R \left(\left\| \frac{\partial \vec{f}}{\partial u} \right\|_2, \left\| \frac{\partial \vec{f}}{\partial v} \right\|_2, \left\| \frac{\partial \vec{f}}{\partial t} \right\|_2 \right). \qquad (4)$$

It is important to emphasize that a hierarchy of bounding spheres is generated from the k-d tree hierarchy. Each subregion in the k-d tree has its own bounding sphere. As subdivision proceeds, the bounding spheres become smaller.

3.3 Jacobian Bounding Boxes

While the Lipschitz spheres will suffice as bounding volumes, we can reduce the size of the bounding volumes, and consequently the average number of interference computations,

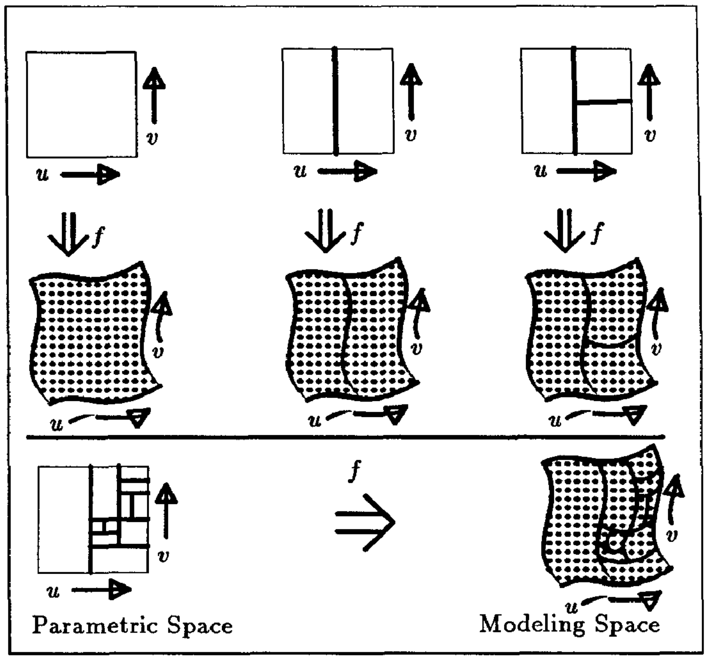

Figure 4: Successive subdivision of a 2-dimensional k-d tree in parametric and modeling space. The function $\vec{f}$ maps parametric space onto modeling space. Each individual subrectangle is called a node of the k-d tree. The aspect ratio of the rectangles may be adjusted by factors of 2.

using the Jacobian of a parametric function. See Appendix A for a derivation of the Jacobian bounding boxes.

To create the Jacobian bounding boxes, we find the maximum of each component of the Jacobian over the region to be bounded, as described in Appendix B. The resulting matrix is called the *rate matrix* M, and places bounds on each of the parametric derivatives over the region R. A sufficient value for the rate matrix M is a constant matrix with all entries set to the value for L in Eqn. 4. Better results are obtained by deriving each component of M separately.

The next section uses the rate matrix M to create bounding boxes for each parametric surface. If the boxes do not overlap, then we confirm that no collision occurs. If they do overlap, then we adaptively subdivide the surfaces to determine if a γ-collision has occurred. As with the bounding spheres, an adaptive hierarchy of bounding boxes is formed based on the k-d tree of each surface.

4 Collision Algorithm

We compute collisions using a bounding volume hierarchy for each parametric surface. The collision algorithm has an important property: parametric surfaces that are far apart will be shown not to collide, using a single sample from each surface. This computation is extremely short, making it trivial to reject collisions between distant objects. Parametric surfaces that do collide will cause the algorithm to adaptively sample each surface near the collision point, using the k-d trees to guide the sampling. In this way the collision is refined until the desired accuracy γ is reached.

To set up the collision algorithm, we are given the parametric functions $\vec{f}(u_f, v_f, t)$ and $\vec{g}(u_g, v_g, t)$. We are also given a function that returns the rate matrix M over a parametric region R.

The task is to compute whether two objects collide, as

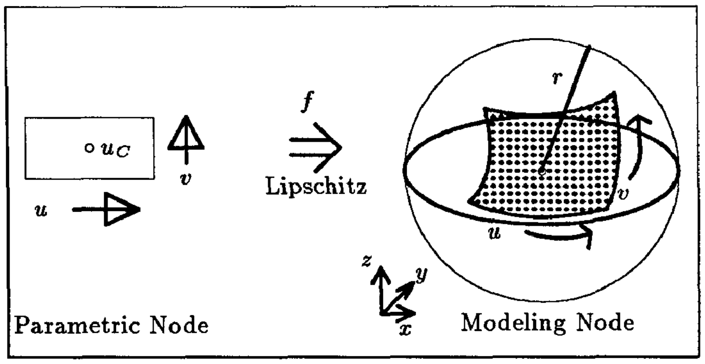

Figure 5: Constructing a bounding sphere about a parametric node. The center of the sphere $\vec{f}(\vec{u}_c)$ in modeling space comes from the center $\vec{u}_c$ in parametric space. The radius r is based on the size of the node in parametric space, and the Lipschitz value for the function. A sphere of radius $r = L(\Delta u + \Delta v + \Delta t)$ bounds the region in modeling space, where Δu, Δv, and Δt represent half-widths of the parametric rectangle.

determined by the loss of separation of the two parametric surfaces. We assume initially that the two objects are disjoint. We are given a threshold distance tolerance γ, below which we should report a collision, including the parameters u_f, v_f, u_g, v_g, and time t.

4.1 Collision Algorithm Approach

Initially we use one node to represent each surface. We subdivide as necessary to determine if a geometric collision occurs within any particular subregion. Parametric sampling is concentrated where it is needed the most, near potential intersections.

The algorithm must find the earliest collision between two surfaces. This implies that we should traverse the nodes of the k-d trees in forward-time order. We can schedule pairs of nodes (one from each surface) to be compared according to the earliest possible collision time, determined from the minima of the time bounds of the parametric subregions. The two parametric regions cannot collide until they both have come into existence. So the maximum of the two starting times represents the earliest possible collision time. Given the time interval $t_A \pm \Delta t_A$ of node A, and the time interval $t_B \pm \Delta t_B$ of node B, we sort the node-pairs according to the earliest possible intersection time $t_{\min}$:

$$t_{\min} = \max(t_A - \Delta t_A, t_B - \Delta t_B). \qquad (5)$$

We maintain a heap data structure [Knuth 69] of pairs of nodes to be compared, sorted in ascending order, using $t_{\min}$ as the sort-key. The distance between the centers of the nodes is used as a secondary sort-key to focus effort on the most probable collision candidates. We successively pop node-pairs off the heap for comparison, in ascending order, according to $t_{\min}$. The node comparison generates new node-pairs whenever there is an overlap in the bounding volumes. The pairs are pushed onto the heap, and the process continues until all pairs are evaluated. This method guarantees that we will find the earliest collision between the surfaces.

4.2 C Implementation

Figure 6 shows an algorithm written in C for computing the collision between two parametric surfaces. The node

```
typedef double vector[3], matrix[3][3];
  A node is a piece of a parametric surface.
typedef struct node_struct {
   vector parameters; (u,v,t) coords in parametric space.
   vector width; (u,v,t) width in parametric space.
   vector position; (x,y,z) coords in modeling space.
   vector radii; (x,y,z) width in modeling space.
   node child1,child2; The two subregions of this node.
   int split_direction; The splitting axis for the node.
   } *node;

vector surface_collision(fn1, fn2, Jmax1, Jmax2, gamma)
vector fn1(), fn2(); The functions to be collided.
matrix Jmax1(), Jmax2(); The maximum of the Jacobians.
float gamma; Collision tolerance.
{ One node from each function; used for comparison.
  node node1, node2;
  heap_flush(); Empty the heap of nodes.
  Put the initial node pair on the heap for evaluation.
  schedule_node_pair(initial_node(fn1, Jmax1),
                            initial_node(fn2, Jmax2));
  As long as nodes are on the heap, compare them.
  while (heap_pop(&node1, &node2)) {
    if (nodes_collide_within_tolerance(node1,node2,gamma))
        return(collision_info(node1, node2));
    The nodes are too large,
    if (norm(node1->radii) > norm(node2->radii)) {
          node_split(node1, fn1, Jmax1);
          schedule_node_pair(node1->child1, node2);
          schedule_node_pair(node1->child2, node2); }
    else {node_split(node2, fn2, Jmax2);
          schedule_node_pair(node1, node2->child1);
          schedule_node_pair(node1, node2->child2); }
    }
  return(NULL); If there are no nodes left to
  }              compare, the surfaces don't collide.

int schedule_node_pair (node1,node2)
node node1,node2;
      {
      if (!time_overlap(node1, node2)) return;
      if (!space_overlap(node1, node2)) return;
      heap_push(node1, node2); Heap is sorted by tmin.
        }
```

Figure 6: An algorithm and data structure to determine collisions for time-dependent parametric surfaces.

data structure represents a region of a parametric surface. The surface_collision function computes a γ-sphere that contains points from both surfaces, or else confirms that the two surfaces do not collide.

The surface_collision function calls several other functions. The function initial_node computes an initial node for the entire surface at location $(u, v, t) = (0.5, 0.5, 0.5)$. The function schedule_node_pair takes a pair of nodes, sees if they overlap in time and in space, and pushes them onto the heap to be scheduled for evaluation according to $t_{\min}$. The function space_overlap returns false if the minimum distance between two bounding boxes is greater than γ. The operation heap_pop pops a pair of nodes off the heap for evaluation. The function collision_info returns the collision parameters and time if a γ-collision took place. Finally, the function node_split subdivides a node into two smaller nodes along the parametric dimension with the greatest contribution to the bounding-box size. Detailed proofs of the algorithms may be found in [Von Herzen 89, Appendix A].

4.3 Termination Condition

The function **nodes_collide_within_tolerance** determines whether a γ-sphere contains both surfaces. For termination, we compute the smallest isothetic rectangle (a rectangle aligned with the coordinate axes) that contains the two bounding boxes. If the largest dimension of the isothetic rectangle is smaller than the separation tolerance, we report the loss of separation of the two surfaces, down to the tolerance specified. Expressed mathematically, for bounding boxes $(x_A, y_A, z_A) \pm (\Delta x_A, \Delta y_A, \Delta z_A)$ and $(x_B, y_B, z_B) \pm (\Delta x_B, \Delta y_B, \Delta z_B)$, and tolerance γ, we require

$$\max\begin{pmatrix} |x_A - x_B| + \Delta x_A + \Delta x_B, \\ |y_A - y_B| + \Delta y_A + \Delta y_B, \\ |z_A - z_B| + \Delta z_A + \Delta z_B \end{pmatrix} \leq \gamma. \qquad (6)$$

4.4 Complexity for Interacting Spheres

We can test the collision algorithm using two parametric spheres. We would expect that as the separation distance decreases between the two spheres, the number of bounding box comparisons should increase. In particular, if the separation distance drops by a factor of two, we will have to create bounding boxes twice as small to confirm that the surfaces do not intersect. For the parametric k-d tree hierarchy, every halving of the separation distance requires a constant number of additional subdivision levels. Assuming that CPU time should be proportional to the number of subdivision levels, the CPU time t should scale as

$$t \propto \log_2((r+S)/S), \qquad (7)$$

where r is the radius of each sphere, and S is the minimum separation between spheres.

4.5 Results for Interacting Spheres

As an illustration of the relationship between computation time and separation distance S, we determine collisions for two spheres while varying S. The total computation time is a function of the minimum separation distance between the two objects. The graph in Figure 7 shows an example of the computation time as a function of S. For an object of radius r and minimum separation distance $S = 2r$, we require only a few samples to be taken from each surface. As the minimum separation distance decreases, we notice an increase in CPU time proportional to the negative logarithm of the separation distance. In this computation we assume $\gamma < S$.

4.6 Results for Other Objects

As a demonstration of results for surfaces more complicated than polynomials or quadrics, the collision method is demonstrated for two spiked objects illustrated in Figure 8. The parametric equation for the spike function is

$$\vec{f}(u,v) = \begin{pmatrix} r(u,v)\cos(2\pi u)\sin(\pi v) \\ r(u,v)\sin(2\pi u)\sin(\pi v) \\ -r(u,v)\cos(\pi v) \end{pmatrix}, \qquad (8)$$

where the radius is given by

$$r(u,v) = r_0 + r_1 \sum_{i=0}^{i<n} e^{-((u-u_i)^2+(v-v_i)^2)/w_0^2}. \qquad (9)$$

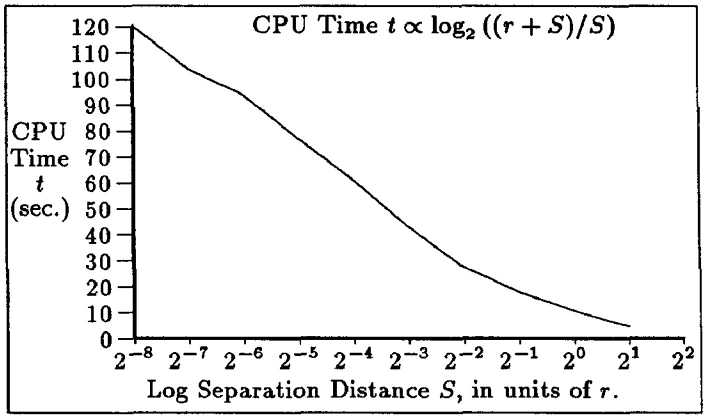

Figure 7: CPU time for two interacting spheres of radius r as a function of $\log S$, where S is the minimum separation between two objects. In this example, $\gamma < S$.

The value n is the number of spikes on the sphere, (u_i, v_i) is the parametric location of the i-th spike, and w_0 determines the radius of the spikes.

Without knowing something about the parametric derivatives of the spike function, it would be very difficult to solve the collision problem for two moving spike functions. As it is, we are able to construct a set of bounding volumes as the computation requires, in order to verify the paths of the two objects.

We illustrate the results of the collision computation. In Figure 8.a, we see two spherical spike functions approaching each other. In Figure 8.b, the algorithm computes a collision between two of the spikes. A physical simulation program computes the recoil as shown in Figure 8.c (see Section 6). This collision computation would have been very difficult to solve without knowing the rate matrix M for the spike function. With this information, we can solve difficult collision problems, using a straightforward application of the collision algorithm of Figure 6. The appendices discuss the creation of M.

5 Constraints on Jacobians

For the collision technique to be most useful, we need to determine constraints on the Jacobian of the parametric functions (See Appendix A). A variety of methods are possible.

The simplest approach is to compute the maximum of any component of the Jacobian over the entire surface, and then to set each entry of the rate matrix M equal to the maximum value. This does not provide particularly tight bounds on the parametric surface, but is sufficient to compute collisions.

Alternatively, we can compute a global maximum for each parametric variable (u, v, t). It is common for the time derivatives, such as $\partial x/\partial t$, to have separate scaling from the spatial parametric derivatives, such as $\partial x/\partial u$ and $\partial x/\partial v$. It is also common for the u and v derivatives to have separate scalings. If we define

$$w_u \equiv \max_R\left(\left|\frac{\partial x}{\partial u}\right|, \left|\frac{\partial y}{\partial u}\right|, \left|\frac{\partial z}{\partial u}\right|\right), \qquad (10)$$

and w_v and w_t similarly, then the following matrix constrains the Jacobian of the parametric surface:

$$\mathrm{M}(R) = \begin{pmatrix} w_u & w_v & w_t \\ w_u & w_v & w_t \\ w_u & w_v & w_t \end{pmatrix}. \qquad (11)$$

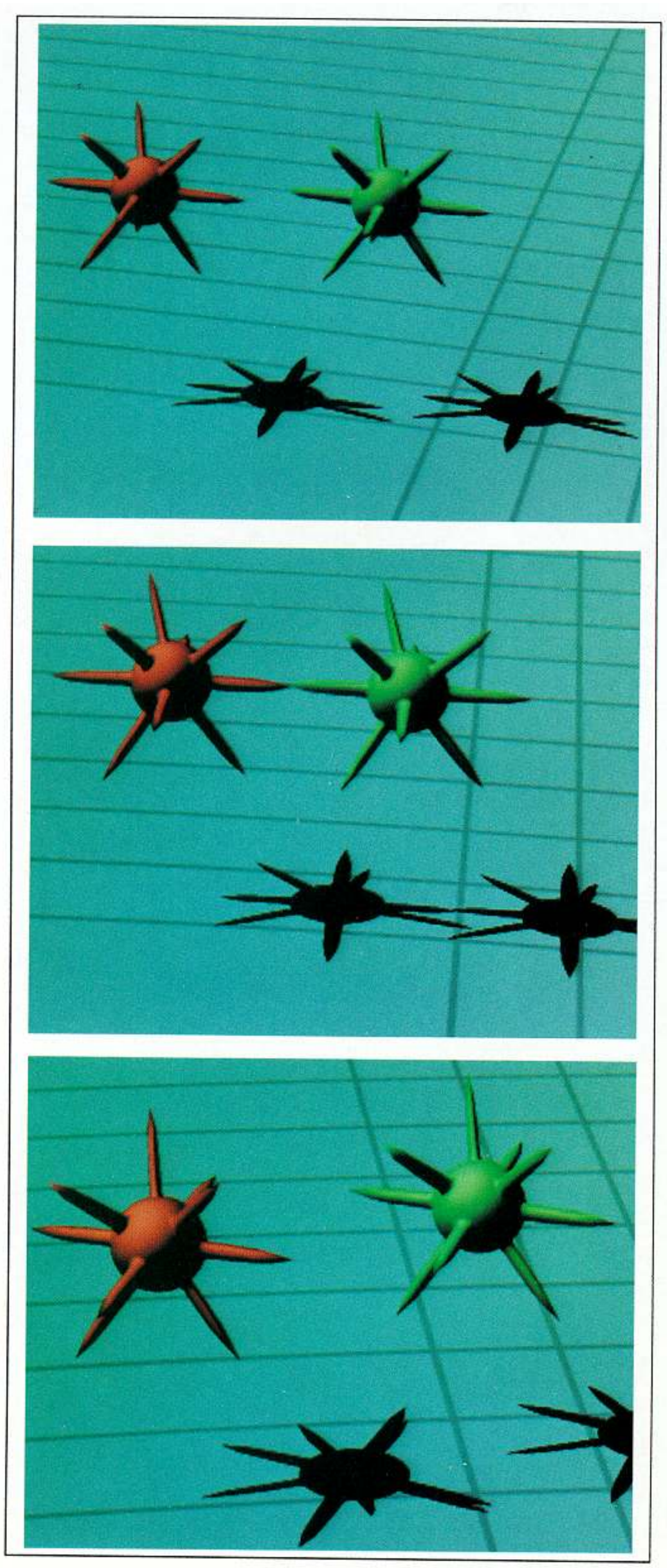

Figure 8: A pair of spike functions before, during, and after a collision.

Each column of M has a separate entry: either a constant for the whole surface, or a function of subregion R. We obtain a tighter set of bounding volumes than with the approach using a single constant.

Perhaps the most general and flexible way to compute constraints on the Jacobian matrix is to create a special function that computes maxima of the derivatives of each parametric function. Frequently we can find analytical expressions of the Jacobian of the parametric function, and the maximum of every component in the Jacobian over some parametric range. In these cases we can produce very tight bounds around a surface. It is frequently possible to find an exact analytic solution to the M function. In other cases we may need to use approximation rules to the various components (Appendix B). We must satisfy only the condition that

$$m_{ij} \geq \max_R |J_{ij}| . \tag{12}$$

In this case, R may be any subregion of the parametric domain of the function. Note that if m_{ij} is much larger than J_{ij}, the algorithm will still work, but will take longer to terminate.

Composition rules such as the triangle inequality in Eqn. 23 can simplify the computation of Jacobian maxima. The rate matrix M for several objects is computed in [Von Herzen 89], along with identities for simplifying the analysis (Appendix B).

6 Potential Application to Physically-Based Simulators

Many physically-based modeling systems need to have an implicit function to tell when a pair of objects come together in a collision. The function should be positive when the two objects do not interfere, negative when the objects overlap, and zero when the objects are just barely in contact. In addition, the function should be continuous.

A simple solution is $h(t) = t_0 - t$, where t_0 is the collision time [Platt 89]. Before the collision, $h(t)$ is positive, and after the collision, $h(t)$ is negative. The function is linear in time, which is very helpful for numerical analysis of physically-based modeling systems. The value of t_0 is computed by the algorithm presented in this paper, whereupon the forces of the collision are computed by the physical simulation system (See, for example [Barzel *et al.* 88]).

7 Conclusion

We have demonstrated a method to determine collisions between time-dependent parametric functions. The method is guaranteed to find the earliest collision for those functions with computable bounds on parametric derivatives. The collision theory and algorithms developed here may potentially apply to robotics and to ray-tracing problems as in [Kalra *et al.* 89]. Even for such difficult functions as the spike functions of Figure 8, the method is practical and robust and easily determines potential collisions between objects.

7.1 Advantages of the Method

In summary, the collision algorithm presented here has the following advantages:

- robust method
- works for deforming time-dependent surfaces
- computes to user-specified accuracy
- finds the earliest collision or near-miss

- works with many types of surfaces, including patches
- interfaces to physical modeling systems
- needs analysis only once per surface type, vs. $O(N^2)$ comparisons between all pairs of surface types

7.2 Disadvantages

Disadvantages of the algorithm include:

- must analyze derivatives for each surface type
- can't guarantee collisions for surfaces with unbounded derivatives

Acknowledgments

We would like to thank Carolyn Collins and Pete Wenzel for their assistance. The work presented in this paper was sponsored in part by International Business Machines, Inc., Hewlett-Packard Co., Apple Computer, Inc., and the Fannie and John Hertz Foundation.

References

[Baraff 89] David Baraff, "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies," *Computer Graphics 23,* 3, July 1989, 223–232.

[Barr 83] Alan H. Barr, *Geometric Modeling and Fluid Dynamic Analysis of Swimming Spermatozoa,* Ph.D. Dissertation, Rensselaer Polytechnic Institute, 1983.

[Barr 84] Alan H. Barr, "Local and Global Deformations of Solid Primitives," *Computer Graphics 18,* 3, July 1984, 21–30.

[Barzel *et al.* 88] Ronen Barzel and Alan H. Barr, "A Modeling System Based on Dynamic Constraints," *Computer Graphics 22,* 4, August 1988, 179–188.

[Bentley *et al.* 79] Jon L. Bentley and Jerome H. Friedman, "Data Structures for Range Searching," *ACM Computing Surveys 11,* 4, December 1979, 397–409.

[Besl *et al.* 88] Paul J. Besl and Ramesh C. Jain, "Segmentation through Variable-Order Surface Fitting," *IEEE Transactions on Pattern Analysis and Machine Intelligence 10,* 2, March 1988, 167–192.

[Bezier 74] Pierre Bezier, "Mathematical and Practical Possibilities of UNISURF," in *Computer-Aided Geometric Design,* edited by Robert E. Barnhill and Richard F. Riesenfeld, Academic Press, New York, 1974, pp. 127–152.

[Blinn 78] Jim Blinn, *Computer Display of Curved Surfaces,* Ph.D. Dissertation, University of Utah, 1978.

[Cameron *et al.* 86] S. A. Cameron and R. K. Culley, "Determining the Minimum Translational Distance Between Two Convex Polyhedra," *IEEE International Conference on Robotics and Automation,* 1986.

[Canny 84] John Canny, "Collision Detection for Moving Polyhedra," *MIT Artificial Intelligence Lab Memo 806,* October, 1984.

[Catmull 75] Catmull, Ed, "Computer Display of Curved Surfaces," *IEEE Conference Proceedings on Computer Graphics, Pattern Recognition and Data Structures,* May 1975, 11.

[Chadwick *et al.* 89] John E. Chadwick, David R. Haumann and Richard E. Parent, "Layered Construction for Deformable Animated Characters," *Computer Graphics 23,* 3, July 1989, 243–252.

[Culley *et al.* 86] R. K. Culley and K. G. Kempf, "A Collision Detection Algorithm Based on Velocity and Distance Bounds," *Proceedings 1986 IEEE International Conference on Robotics and Automation,* Volume 2, pp. 1064–1069.

[Filip *et al.* 86] Daniel Filip, Robert Magedson and Robert Markot, "Surface Algorithms using Bounds on Derivatives," *Computer Aided Geometric Design 3,* 1986, 295–311.

[Gear 71] C. William Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971, p. 55.

[Going Bananas 88] John Snyder, Jed Lengyel, Devendra Kalra, Ronen Barzel, John C. Platt, Alan H. Barr and Brian Von Herzen, *Going Bananas,* 1988 Siggraph Film Show.

[Hopcroft *et al.* 83] J. E. Hopcroft, J. T. Schwartz and M. Sharir, "Efficient Detection of Intersections among Spheres," *The International Journal of Robotics Research 2,* 4, Winter 1983, 77–80.

[Kalra *et al.* 89] Devendra Kalra and Alan H. Barr, "Guaranteed Ray Intersections with Implicit Surfaces," *Computer Graphics 23,* 3, July 1989, 297–306.

[Kaufman 87] Arie Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Computer Graphics 21,* 4, July 1987, 171–180.

[Knuth 69] Donald Knuth, *The Art of Computer Programming; Vol. 1, Fundamental Algorithms,* Addison-Wesley, Menlo Park, CA, 1969, Section 2.2.4.

[Lane *et al.* 79] Jeff Lane and Loren Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing 11,* 1979, 290.

[Lane *et al.* 80] Jeff Lane and Richard F. Riesenfeld, "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence 2,* 1, January 1980, 35–46.

[Lee *et al.* 84] D. T. Lee and Franco P. Preparata, "Computational Geometry— A Survey," *IEEE Transactions on Computers C-33,* 12, December 1984, 1072.

[Lin *et al.* 74] C. C. Lin and L. A. Segel, *Mathematics Applied to Deterministic Problems in the Natural Sciences,* Macmillan Publishing Co., Inc., New York, 1974, pp. 57–58.

[Moore *et al.* 88] Matthew Moore and Jane Wilhelms, "Collision Detection and Response for Computer Animation," *Computer Graphics 22,* 4, August 1988, 289–298.

[NAG] NAG Fortran Library, Numerical Algorithms Group, 1400 Opus Place, Suite 200, Downers Grove, IL 60515 (312) 971-2337.

[Platt *et al.* 88] John C. Platt and Alan H. Barr, "Constraint Methods for Flexible Models," *Computer Graphics 22,* 4, August 1988, 279–288.

[Platt 89] John C. Platt, personal communication.

[Samet 84] Hanan Samet, "The Quadtree and Related Hierarchical Data Structures," *Computing Surveys 16,* 2, June 1984, 187–260.

[Samet 90a] Hanan Samet, *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, Menlo Park, CA, 1990, Section 2.4, pp. 66–80.

[Samet 90b] Hanan Samet, *Applications of Spatial Data Structures,* Addison-Wesley, Menlo Park, CA, 1990, Section 1.3, pp. 15–16.

[Schmitt *et al.* 86] Francis Schmitt, Brian Barsky and Wen-Hui Du. "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data," *Computer Graphics 20,* 4, August 1986, 179–188.

[Schwarz 81] J. T. Schwarz, "Finding the Minimum Distance Between Two Convex Polygons," *Information Processing Letters 13,* 4, 1981, 168–170.

[Schweitzer *et al.* 82] D. Schweitzer and E. S. Cobb, "Scanline Rendering of Parametric Surfaces," *Computer Graphics 16,* 3, July 1982, 265.

[Sederberg *et al.* 86] Tom Sederberg and Scott Parry, "Free-Form Deformation of Solid Geometric Models," *Computer Graphics 20,* 4, August 1986, 151–160.

[Snyder 90] John Snyder *Generative Models,* Ph.D. Dissertation, California Institute of Technology, in progress.

[Terzopoulos *et al.* 88] Demetri Terzopoulos and Kurt Fleischer, "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," *Computer Graphics 22,* 4, August 1988, 269–278.

[Uchiki *et al.* 83] Tetsuya Uchiki, Toshiaki Ohashi and Mario Tokoro, "Collision Detection in Motion Simulation," *Computers and Graphics 7,* 3, 1983, 285–293.

[Von Herzen *et al.* 87] Brian Von Herzen and Alan H. Barr, "Accurate Triangulations of Deformed, Intersecting Surfaces," *Computer Graphics 21,* 4, July 1987, 103–110.

[Von Herzen 85] Brian Von Herzen, *Sampling Deformed, Intersecting Surfaces with Quadtrees,* Masters Thesis, California Institute of Technology, Computer Science Dept., 5179:TR:85, 1985.

[Von Herzen 89] Brian Von Herzen, *Applications of Surface Networks to Sampling Problems in Computer Graphics,* PhD. Dissertation, California Institute of Technology, Computer Science Dept., Caltech-CS-TR-88-15, 1989.

A Appendix: Jacobian Bounding Boxes

Here we derive a set of bounding boxes for parametric functions using the Jacobian of the function. These boxes frequently produce tighter bounds on a parametric surface than the Lipschitz bounding spheres. We start with the original definition of the Lipschitz condition for parametric functions ([Gear 71]):

$$\left\|\vec{f}(\vec{u}) - \vec{f}(\vec{u}_c)\right\| \leq L\left\|\vec{u} - \vec{u}_c\right\|. \tag{13}$$

Assume that the condition holds over some parametric subregion $R : u_1 \leq u \leq u_2$, $v_1 \leq v \leq v_2$, and $t_1 \leq t \leq t_2$. We define parametric coordinates $\vec{u}_c = (u_c, v_c, t_c)$ at the center of region R, and modeling space coordinates $(x_c, y_c, z_c) = \vec{f}(\vec{u}_c)$. We choose an L_1 norm for the right side of Eqn. 13, and we apply the condition to each component of $\vec{f}$ separately:

$$\begin{array}{rcl} |x - x_c| & \leq & L_x\left(|u - u_c| + |v - v_c| + |t - t_c|\right), \\ |y - y_c| & \leq & L_y\left(|u - u_c| + |v - v_c| + |t - t_c|\right), \\ |z - z_c| & \leq & L_z\left(|u - u_c| + |v - v_c| + |t - t_c|\right), \end{array} \tag{14}$$

for some suitable values of L_i. We distribute the values L_i and rename them to arrive at a more general inequality:

$$\begin{array}{rcl} |x - x_c| & \leq & M_{xu}\,|u - u_c| + M_{xv}\,|v - v_c| + M_{xt}\,|t - t_c|, \\ |y - y_c| & \leq & M_{yu}\,|u - u_c| + M_{yv}\,|v - v_c| + M_{yt}\,|t - t_c|, \\ |z - z_c| & \leq & M_{zu}\,|u - u_c| + M_{zv}\,|v - v_c| + M_{zt}\,|t - t_c|. \end{array} \tag{15}$$

We can solve for each M_{ij} by choosing appropriate values of (u, v, t). We illustrate with M_{xu}:

$$|x(u, v, t) - x(u_c, v, t)| \leq M_{xu}\,|u - u_c|, \tag{16}$$

or

$$\left|\frac{x(u, v, t) - x(u_c, v, t)}{u - u_c}\right| \leq M_{xu}, \quad u \neq u_c. \tag{17}$$

Assuming that $x(u, v, t)$ is differentiable, a sufficient value of M_{xu} is

$$M_{xu} \equiv \max_R\left|\frac{\partial x(u, v, t)}{\partial u}\right|. \tag{18}$$

M_{xu} is an upper bound on the parametric derivative over the region R. In general, a sufficient value of the **rate matrix** M is:

$$\mathbf{M} \equiv \begin{pmatrix} \max\limits_R\left|\frac{\partial x}{\partial u}\right| & \max\limits_R\left|\frac{\partial x}{\partial v}\right| & \max\limits_R\left|\frac{\partial x}{\partial t}\right| \\ \max\limits_R\left|\frac{\partial y}{\partial u}\right| & \max\limits_R\left|\frac{\partial y}{\partial v}\right| & \max\limits_R\left|\frac{\partial y}{\partial t}\right| \\ \max\limits_R\left|\frac{\partial z}{\partial u}\right| & \max\limits_R\left|\frac{\partial z}{\partial v}\right| & \max\limits_R\left|\frac{\partial z}{\partial t}\right| \end{pmatrix}. \tag{19}$$

Just as the Lipschitz value L is a generalization of the derivative, so the rate matrix $\mathbf{M}$ is a generalization of the Jacobian matrix for parametric vector functions of several variables [Lin *et al.* 74, p. 355]. The matrix $\mathbf{M}$ consists of upper bounds on all the parametric derivatives of all the components of vector function $\vec{f}$.

We define $\Delta u \equiv |u_2 - u_c|$, $\Delta v \equiv |v_2 - v_c|$, and $\Delta t \equiv |t_2 - t_c|$. Since $u_1 \leq u \leq u_2$, we have $|u - u_c| \leq \Delta u$. Similarly, $|v - v_c| \leq \Delta v$, and $|t - t_c| \leq \Delta t$. Substituting into Eqn. 15, we have the **rate condition:**

$$\begin{array}{rcl} |x - x_c| & \leq & M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ |y - y_c| & \leq & M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ |z - z_c| & \leq & M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t. \end{array} \tag{20}$$

We define the **bounding box radii** to be

$$\begin{array}{l} \Delta x \equiv M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ \Delta y \equiv M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ \Delta z \equiv M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t. \end{array} \tag{21}$$

Now we can construct a bounding volume in modeling space from the bounding box radii, based on Δu, Δv, Δt, and the rate matrix. We form a rectangular prism that is aligned with the x, y, and z axes, centered about modeling coordinates (x_c, y_c, z_c). Combining Eqn. 21 with Eqn. 20, we get the **bounding box inequality:**

$$\begin{array}{rcl} |x - x_c| & \leq & \Delta x \\ |y - y_c| & \leq & \Delta y \\ |z - z_c| & \leq & \Delta z. \end{array} \tag{22}$$

Such a rectangular region is called an *isothetic rectangle,* a rectangle whose sides are parallel to coordinate axes [Lee *et al.* 84]. The set of points satisfying Eqn. 22 form a bounding box containing the parametric region. We now have an efficient bounding box useful for computing collisions between moving parametric surfaces. We are free to compute the Jacobian maxima over the entire surface, thereby computing with a single-valued constant matrix across the surface. Alternatively, we may compute the Jacobians over subregions in order to tailor the bounding volumes more closely to particular variations in the surface. These boxes frequently produce tighter bounds on the parametric functions than does the Lipschitz condition of Eqn. 3.

B Appendix: Bounds on Parametric Derivatives

Here we describe how to compute the entries in the matrix $\mathbf{M}$ from Eqn. 19. In addition to the differentiable surfaces, some non-differentiable surfaces also have computable Lipschitz values from which to derive rate matrices ([Von Herzen 89, Appendix B.3]). In this section we will focus our attention on differentiable parametric surfaces.

B.1 Maxima of scalars

We frequently have a closed-form description of $x(u, v, t)$ that permits us to compute the derivative $x'(u, v, t)$ directly. Then we can use the following identities to compute the maxima of functions:

Eqn. 23 is known as the triangle inequality. It is equivalent to the law that the length of the longest side of a triangle must be less than the lengths of the two shorter sides added together:

$$\max_R|\vec{f}(R) + \vec{g}(R)| \leq \max_R|\vec{f}(R)| + \max_R|\vec{g}(R)|. \tag{23}$$

Similar laws hold for the operations of subtraction, multiplication, and division of functions.

$$\max_R|\vec{f}(R) - \vec{g}(R)| \leq \max_R|\vec{f}(R)| + \max_R|\vec{g}(R)|, \tag{24}$$

$$\max_R|\vec{f}(R)\vec{g}(R)| \leq \max_R|\vec{f}(R)|\max_R|\vec{g}(R)|, \tag{25}$$

$$\max_R|\vec{f}(R)/\vec{g}(R)| \leq \frac{\max\limits_R|\vec{f}(R)|}{\min\limits_R|\vec{g}(R)|}, \tag{26}$$

for all $\vec{f}(R)$ and $\vec{g}(R)$.

B.2 Maxima of polynomials

Given $h(t)$, a polynomial function of degree $n = 2, 3$, or more, we want to maximize its value over a range $t_a \leq t \leq t_b$. The polynomial $h(t)$ is assumed to be of the form $h(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + \ldots$. The maximum in $h(t)$ occurs either at 0, t_0, or at the points of solution for $h'(t) = 0$.

We take the derivative analytically and then solve the resulting polynomial equation using any one of a variety of numerical analysis programs (see [NAG]) for t, to get a set of values $t = t_1, \ldots, t_N$. Add 0 and t_0 to the set to get $0, t_1, \ldots, t_N, t_0$.

Then we substitute these values into the definition for $h(t)$. and pick the maximum value of $h(0)$ or $h(t_i)$, for $0 \leq i \leq N$. This is the maximum value for the whole interval, $t_a \leq t \leq t_b$.

For any interval $t_a \leq t \leq t_b$ we only need to evaluate $h(t)$ at the endpoints t_a and t_b and any values in the solution set between t_a and t_b. This recalculation will reduce the magnitude of the Lipschitz value as the interval decreases with further iterations.

Similar solutions are possible for polynomial patches that use a rational cubic representation in one parametric direction ([Filip *et al.* 86, p. 307]). It is straightforward to extend these results to several dimensions.

B.3 Product surfaces

Product surfaces include superquadrics, spheres, profile surfaces, translational sweeps, and spherical products [Barr 83]. These surfaces take the mathematical form:

$$x_i(u,v) = k(v)c_i(u) + d_i(v), \tag{27}$$

where $i = 1,2,3$, and subscripts 1,2 and 3 correspond to components x, y, and z.

The partial derivative of this surface with respect to u is:

$$\frac{\partial x_i(u,v)}{\partial u} = k(v)\frac{\partial c_i(u)}{\partial u} + d_i(v). \tag{28}$$

Sufficient values of the entries of the rate matrix M are:

$$m_{iu} = \max_R |k(v)| \max_R \left|\frac{\partial c_i(u)}{\partial u}\right| + \max_R |d_i(v)|, \tag{29}$$

$i = 1,2,3$. The rate matrix entries with respect to parameter v are given by:

$$m_{iv} = \max_R \left|\frac{\partial k(v)}{\partial v}\right| \max_R |c_i(u)| + \max_R \left|\frac{\partial d_i(v)}{\partial v}\right|. \tag{30}$$

Finally, all the time derivatives are zero: $m_{it} = 0$. Given differentiable scalar functions for $k(v), c_i(u)$, and $d_i(v)$, we can find the rate matrix for the product surface.

B.4 Surfaces with Translational Motion

Assuming that we can compute the rate matrix for a stationary surface $\vec{f}(u,v)$, how can we compute the rate matrix for the same surface that is translating as a function of time? ([Von Herzen 89, Appendix B.2]). We define the translation function to be $\vec{s}(t)$. The translating surface is given by function $\vec{g}(u,v,t) = \vec{f}(u,v) + \vec{s}(t)$. If the value m_{ij} represents the rate matrix for $\vec{f}$, then the new rate matrix Mg for the moving surface $\vec{g}(u,v,t)$ is

$$\mathrm{Mg} = \begin{pmatrix} m_{xu} & m_{xv} & \max_R \left|\frac{\partial s_x(t)}{\partial t}\right| \\ m_{yu} & m_{yv} & \max_R \left|\frac{\partial s_y(t)}{\partial t}\right| \\ m_{zu} & m_{zv} & \max_R \left|\frac{\partial s_z(t)}{\partial t}\right| \end{pmatrix}. \tag{31}$$

B.5 Surfaces with Rotational Motion

We now examine rotational motion for rigid objects. Given a function $\vec{f}(u,v)$, and a rotation matrix $\mathrm{R}(t)$ as a function of time, we have $\vec{g}(u,v,t) = \mathrm{R}(t)\vec{f}(u,v)$. The parametric derivatives of $\vec{g}$ are given by

$$\frac{\partial \vec{g}(u,v,t)}{\partial u} = \mathrm{R}(t)\frac{\partial \vec{f}(u,v)}{\partial u}, \tag{32}$$

$$\frac{\partial \vec{g}(u,v,t)}{\partial v} = \mathrm{R}(t)\frac{\partial \vec{f}(u,v)}{\partial v}, \tag{33}$$

$$\frac{\partial \vec{g}(u,v,t)}{\partial t} = \frac{\partial \mathrm{R}(t)}{\partial t}\vec{f}(u,v). \tag{34}$$

B.6 Example of a deformation

As an example of computing the rate matrix for a deforming function, we illustrate how to compute the rate matrix for an object with a variable taper as given in [Barr 84], assuming we have the rate matrix for the undeformed object. Let $\vec{f}(u,v)$ be the undeformed object with components (x,y,z). The deformed coordinates are given by $X = r(z,t)x$ for the x component, $Y = r(z,t)y$ for the y component, and $Z = z$ for the z component, where $r(z,t)$ is the tapering function that varies over time. Then the derivatives for the deformed coordinates are:

$$\frac{\partial X}{\partial u} = \frac{\partial r}{\partial z}\frac{\partial z}{\partial u}x + \frac{\partial x}{\partial u}r(z,t), \tag{35}$$

$$\frac{\partial X}{\partial v} = \frac{\partial r}{\partial z}\frac{\partial z}{\partial v}x + \frac{\partial x}{\partial v}r(z,t), \tag{36}$$

$$\frac{\partial X}{\partial t} = \frac{\partial r}{\partial t}x + \frac{\partial x}{\partial t}r(z,t). \tag{37}$$

The equations are analogous for the Y component. All of the derivatives for Z are equal to the derivatives for z.

A typical taper function $r(z,t)$ is a piecewise linear function that tapers from r_1 to r_2 starting at z_1 and ending at z_2. We can make the ending values of the taper vary as a function of time, $r_1(t)$ and $r_2(t)$. The function $r(z,t)$ is given by

$$r(z,t) = \begin{cases} r_1(t) & z < z_1, \\ \dfrac{(z-z_1)r_2 + (z_2 - z)r_1}{z_2 - z_1} & z_1 \le z \le z_2, \\ r_2(t) & z > z_2. \end{cases} \tag{38}$$

The derivatives of $r(z,t)$ are given by

$$\frac{\partial r}{\partial z} = \begin{cases} 0 & z < z_1, \\ \dfrac{r_2 - r_1}{z_2 - z_1} & z_1 \le z \le z_2, \\ 0 & z > z_2. \end{cases} \tag{39}$$

The temporal derivative is given by

$$\frac{\partial r}{\partial t} = \begin{cases} \dfrac{\partial r_1(t)}{\partial t} & z < z_1, \\ \dfrac{(z-z_1)\frac{\partial r_2}{\partial t} + (z_2 - z)\frac{\partial r_1}{\partial t}}{z_2 - z_1} & z_1 \le z \le z_2, \\ \dfrac{\partial r_2(t)}{\partial t} & z > z_2. \end{cases} \tag{40}$$

Eqn. 40 is valid for dynamic tapers of static objects. The differentiation rule for products leads to the equation for tapers of distorting objects. These equations may be substituted directly into Eqn. 19 for the rate matrix to obtain derivative bounds on parametric surfaces tapering as a function of time.