Computer Aided Geometric Design 4 (1987) 3–16
North-Holland

Surface/surface intersection

R.E. BARNHILL *, G. FARIN *, M. JORDAN and B.R. PIPER

Mathematics Department, University of Utah, Salt Lake City, UT 84112, U.S.A.

Presented at Wolfenbuettel 25 June 1986 by R.E. Barnhill
Received 26 September 1986; revised 5 January 1987

Abstract. Finding the intersection of two surfaces is important for many Computer Aided Design tasks concerned with surface modeling. An adaptive algorithm is developed for finding the intersection curve(s) of pairs of rectangular parametric patches which are continuously differentiable. The balance between robustness and efficiency of the algorithm is controlled by a set of tolerances. A suite of examples concludes the paper.

Keywords. Surfaces, geometry, intersection, parametric patches.

1. Introduction

The problem of describing the intersection between two surfaces arises often in engineering applications, for example in geometric modeling, computer aided design/computer aided manufacturing and robotics.

For many CAD applications, the design of surfaces is not sufficient by itself: in order to arrive at the final CAD product, the surfaces need further processing. An example is the description of cutter paths needed for numerically controlled milling. An important tool for such processing is a Surface/Surface Intersection ('SSI') routine. We list several tasks where SSI is necessary:

- Many surfaces are 'overdescribed', i.e., only a part of the designed surface is needed later. The desired part is typically obtained by cutting away pieces of the designed surface by intersection with other surfaces.
- Design verification often involves interference checks of surfaces.
- Cutter paths are often defined by intersecting a surface with a plane or a cylinder.

We consider the (possible) intersection between two rectangular parametric C^1 patches, each defined over a unit square. Each surface is assumed to be defined only by means of a surface evaluator, which, from the input of a parametric (u, v)-value, specifies a point on the surface and the two (parametric) tangent vectors at that point. No special structure, such as a patch being polynomial or of Coons type, is assumed. This is an important non-assumption, because most published SSI algorithms do assume that the patches have polynomial structure (cf. [Farouki '86]). (Of course these particular rectangular patches can be input to our algorithm which is designed for general rectangular patches.) The only published algorithm for problems of comparable generality is [Houghton et al. '85].

Surface algorithms that are as general as our SSI algorithm have the following advantage: suppose a new surface type is to be added to an existing CAD system. If the existing surface algorithms (intersections, offsets, rendering etc.) depend on particular surface types, then they have to be newly created for the new surface type and added to the existing code. This extra

* Present address: Computer Science Dept., Arizona State Univ., Tempe, AZ 85287, U.S.A.

effort is avoided by the use of general surface algorithms. This advantage is somewhat counterbalanced by the fact that all-purpose algorithms tend to be slower than specialized ones.

Our algorithm is not applicable to some important surface types such as triangular patches which have a fundamentally different geometric structure. One standard – and, we believe, incorrect – approach to triangular patches is to consider them as degenerate rectangular patches. Since we recommend against this method of forming patches, we also recommend against using our SSI algorithm on such degenerate patches.

Pratt and Geisow [Pratt, Geisow '86] give an overview of several numerical analysis techniques applied to the surface/surface intersection problem. In this paper we present an algorithm which combines some of these methods with some of our own methods in such a fashion so as to obtain the best attributes of each method. In particular, we shall exploit the techniques of subdivision and marching along an intersection curve to obtain a robust and efficient algorithm.

In Section 2 we describe the particulars of the problem and the tolerances we shall introduce to solve it. Section 3 describes the algorithm and Section 4 considers special cases and extensions. Section 5 shows examples and Section 6 concludes the paper and describes some additional research topics.

The SSI algorithm to be described is an evolving algorithm which will continue to change. This paper can be regarded as a progress report on it.

2. Problem description

2.1. Criteria for success

We first consider the types of situations that can arise when intersecting two surfaces. The intersection of two smooth surfaces is either

(i) empty,
(ii) a collection of points,
(iii) a collection of smooth curves,
(iv) a collection of smooth surfaces,
(v) any combination of (ii), (iii), and (iv).

Two surfaces are said to be in 'general' position if their intersection is of type (i) or (iii) and it is for this type that we address the problem of devising an algorithm to find the intersections. Numerical construction of the intersection between two surfaces not in general position is much more difficult.

In the present paper we shall assume that the surfaces $\mathbf{S}_1$ and $\mathbf{S}_2$ are both C^1 surfaces, defined parametrically on the unit square, for which positions and partial derivatives can be evaluated at any point. We wish to approximate the intersections of type (iii) by piecewise linear curves in each of the parametric domains. The corresponding piecewise linear curve in the range will be called a 'machine curve' and serves as the representation of the intersection curve in $\mathbf{R}^3$. This representation of the intersection curves provides sufficient information to implement these curves in the applications mentioned in the introduction.

The criteria for success of the algorithm are:

1. The algorithm produces reasonable results on realistic examples.
2. The algorithm is reasonably efficient.

It is possible to achieve criterion 1 by producing a method which exhaustively subdivides the surface (cf. [Pratt, Geisow '86]) until a certain global tolerance is met. This unfortunately does not produce an algorithm which satisfies criterion 2. In order to strike a reasonable balance between these two criteria we introduce the following four parameters ('tolerances').

2.2. Tolerances

1. Same Point Tolerance (SPT)

If the distance between two points in $\mathbb{R}^3$ is less than SPT, then the points are considered to be the same. This tolerance will serve as an indication as to whether a pair of points, one on each of the surfaces, represents a true intersection point. SPT is typically chosen to be small (e.g. 1.E − 5).

2. Search Refinement Tolerance (SRT)

The tolerance SRT controls the adaptive subdivision used to obtain initial piecewise linear approximations of the surface. It thus controls the quality of the approximation that is used to find starting points of intersection curves. As this value is decreased, the robustness of the algorithm improves and thus execution time increases.

3. Curve Refinement Tolerance (CRT)

If two points on an intersection curve in three-space are within CRT, then the line segment joining them is taken to be a reasonable approximation to the true intersection curve. We shall use this tolerance to determine when we no longer need to refine our approximation of the intersection curve. As this tolerance is decreased, more points are found on the intersection curve but in many cases these additional points are redundant. Thus, this tolerance may be used to speed up the execution time of the algorithm by eliminating effort spent on examining the intersection curve too closely.

4. Optimization Tolerance (OPT)

The tolerance OPT is used to define an optimal piecewise linear representation of a machine curve, where 'optimal' means fewest number of points. That is, we shall use this tolerance to discard unnecessary points found on the intersection curve so as to produce a more compact result. Here, the term 'unnecessary' refers to points which are nearly collinear with other points on the machine curve.

Each of these tolerances corresponds to a different way of measuring the success of the Surface/Surface Intersection algorithm. Allowing the user to choose different priorities to these measures (by assigning different values for the tolerances) is the key to producing an algorithm that meets both criteria in Section 2.1. For this reason, it is relevant to discuss further the interconnection between these tolerances.

In particular, SPT, CRT and OPT distinguish three important features concerning the quality of the solution curves. A small value of SPT ensures that the points produced on the intersection curve are quite accurate while a small value of CRT ensures that there are a sufficient number of points to approximate the intersection curve. By distinguishing the values OPT and CRT we allow the user to insist that the algorithm produces a very accurate representation of the intersection curve (by making CRT small) without leaving an abundance of unnecessary points on the intersection curve (by making OPT non-zero) where the curve is almost linear.

Distinguishing the quality of the intersection curves produced from the ability to find all intersection curves is done by the introduction of the parameter SRT. A large value for this tolerance allows the user to maintain quality intersection curves (by use of SPT, OPT and CRT) without slowing down the algorithm by spending a long time looking for intersection curves that may not exist.

In general, the tolerances SRT and CRT control robustness and efficiency and how to choose them is a difficult problem. One way of obtaining good choices is to base them on the maximum curvature of the surface. As this value increases these tolerances should be decreased

and vice-versa. This of course requires additional information about the surfaces and we do not pursue this idea here. Rather, we note that for most applications it is reasonable to choose SRT > CRT ≫ SPT and we provide standard default values for these tolerances as well as for the tolerance OPT.

3. The SSI algorithm

3.1. Outline of algorithm

Before describing the details of the algorithm we give a general outline. The algorithm proceeds in two distinct stages. The first is to find one point on some intersection curve. This point is then used as the starting point in 'following' this intersection curve by producing more points in a sequential fashion along the curve.

A fundamental problem associated with both of the stages is the ability to find points on the intersection curve to within the accuracy SPT. The solution to this problem is essential as it must be both fast and robust. We shall achieve this combination by the familiar numerical analysis technique of using a Newton–Raphson iteration for speed and a subdivision technique to get started and in case the Newton–Raphson iteration fails to converge.

A second problem associated with these techniques is ensuring that time is not wasted by following intersection curves twice. We circumvent this problem by keeping enough information about the intersection curves we find so that we may quickly identify when we produce starting guesses on curves already found.

We now consider the details of the algorithm.

3.2. Adaptive mesh generation

The first goal is to find one point on an intersection curve. The idea is the following: It is relatively easy to find the intersection of a line segment with a flat triangle. Therefore, the strategy is to reduce the problem to this situation.

For each surface, we create adaptively an initial rectangular grid in the domain. By splitting each rectangle into two triangles, we obtain a piecewise linear approximation to the surface. The goal is to approximate curved regions by more triangles than flat regions while ensuring that the approximation satisfies the requirements of the tolerance SRT.

We begin by approximating a set (typically three) of isoparametric curves in both the u- and the v-directions by polygons. This is accomplished by recursively subdividing each line at midpoints if a secant-curve-closeness check (against the tolerance SRT) fails. The (u, v) values of these polygons are then superimposed in order to form the rectangular partition of the domain. This superposition is facilitated by the fact that the recursive subdivision always generates parameter values that are multiples of a power of $1/2$.

This rectangular partition is triangulated by drawing in diagonals. The triangulations have a 'rectangular structure' which speeds up subsequent look-up operations. This triangulation as well as the rectangular grid of domain and range points is stored for subsequent use.

3.3. Initial intersection point generation

We first find one starting point (initial intersection point) and then attempt to follow the intersection curve containing it. After the curve corresponding to this starting point is found we shall return and look for more starting intersection points. Thus, once we have an intersection curve, we can save time in the algorithm for finding starting points by not considering regions

near the intersection curves already found. We now describe the details of how starting points on intersection curves are found.

We use the approximations obtained in Section 3.2 and intersect isoparametric curves of surface $\mathbf{S}_2$ with surface $\mathbf{S}_1$. That is, the univariate piecewise linear approximation to an isoparametric curve of $\mathbf{S}_2$ is intersected with the bivariate piecewise linear approximation (over triangles) to the surface $\mathbf{S}_1$.

The univariate piecewise linear approximation that we use depends on the intersection curves we have already found. As we shall describe in Section 3.4, when we are following an intersection curve we form, for each isoparametric line in the original rectangular grids, a list of the places where the intersection curve crosses that isoparametric line. Since there is no need to search regions near these crossings for new starting points, we create a piecewise linear approximation to the isoparametric curve that has these regions deleted. Thus, we enhance the approximation of the isoparametric curve from Section 3.2 by inserting two new points for each crossing place that bracket this crossing point (to within the tolerance CRT). We then ignore the segment between these two points in subsequent processing (see Fig. 1).

If the intersection between this isoparametric polygon and the other piecewise linear surface is non-empty the result is a point that is not, in general, on either surface (see Fig. 2). In fact, the information that will be of more use to us is the domain values corresponding to this point (via the piecewise linear approximation). Thus, we have one point in each domain such that their range values (under the maps $\mathbf{S}_1$ and $\mathbf{S}_2$) are two points close to a true intersection point on a (hopefully) new intersection curve.

Before we try to follow the curve we must first obtain a true intersection point (i.e., two points, one on each surface, that are within SPT). This is done by a curve/surface iteration procedure which we now describe.

We use the pair of domain points found above as the first iterate in a Newton–Raphson curve/surface iteration. That is, we solve the system of three nonlinear equations in the three variables v, s, t given by

$$\mathbf{S}_2(u, v) - \mathbf{S}_1(s, t) = \mathbf{0}$$

where u is fixed by the current isoparametric line. If Newton's Method does not converge (to within tolerance SPT) we subdivide the two piecewise linear approximations in order to improve the first iterate. We continue in this fashion until either we converge to a solution or the piecewise linear approximations (after subdivision) no longer intersect.

If the iteration procedure converges we have a candidate point for a starting point to use in following an intersection curve. However, even once we have this true intersection point we must again check to see if this point is on an intersection curve already found. This is because the piecewise linear approximations may intersect outside all deleted segments while the true intersection point (found after curve/surface iteration) is inside a deleted segment. If the

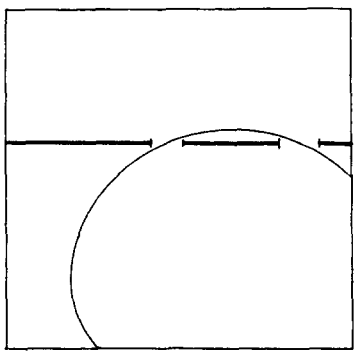


Fig. 1. Gaps are introduced into isoparametric lines around points where intersections were recorded.

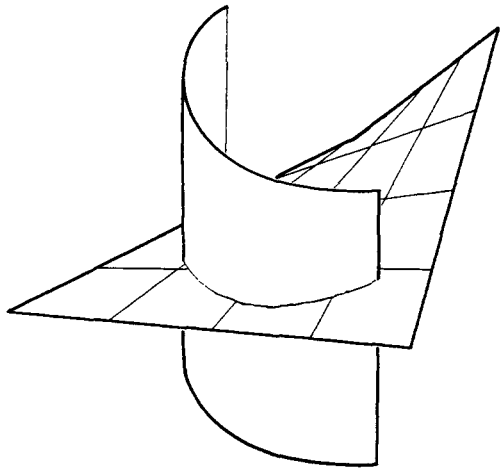


Fig. 2a. A bilinear surface that is intersected with a cylindrical surface.

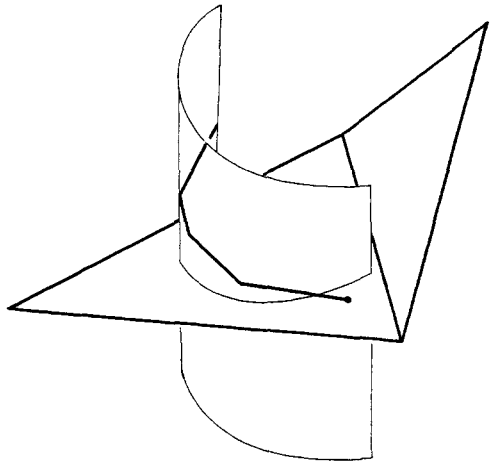


Fig. 2b. The piecewise linear approximation to the bilinear surface and its intersection with a piecewise linear approximation to an isoparametric line of the cylindrical surface. The intersection point is not on either surface.

intersection point is interior to none of the segments on this isoparametric line, we conclude it is not on a previously found curve and use this point as a starting point for the curve following procedure described in Section 3.4.

If the curve/surface iteration does not converge or the point is on a curve already found, we continue looking through the isoparametric curves. We shall consider all the isoparametric curves of $\mathbf{S}_2$ and then interchange the roles of $\mathbf{S}_1$ and $\mathbf{S}_2$. The algorithm is complete when all isoparametric curves have been considered.

3.4. Following an intersection curve

With initial point $\mathbf{S}_1(s_0, t_0) = \mathbf{S}_2(u_0, v_0)$ from Section 3.3 on the intersection curve, we step along the curve. To do this we first find an estimate for the next point on the intersection curve and bring it to a true intersection point. We then refine the linear segment between these two points to meet the tolerance CRT and finally we optimize the result.

We begin by obtaining a new point on the curve. We shall call the current point on the intersection curve $\mathbf{P}_0$ and take a vector step $\mathbf{V}$ in the tangential direction of the intersection curve to obtain a 'guess' point $\hat{\mathbf{P}}_1$ (see Fig. 3). In the general case, $\mathbf{V}$ is determined by intersecting the tangent planes of the two surfaces at $\mathbf{P}_0$ and scaling this vector to be of length proportional to CRT. In case the two tangent planes are parallel, we obtain the direction of the vector $\mathbf{V}$ by using previous points on the intersection curve to approximate a tangent line to the intersection curve at $\mathbf{P}_0$.

Then, by inverting the Jacobians at $\mathbf{P}_0$, we obtain 'guess' points in the domain of each surface corresponding to $\hat{\mathbf{P}}_1$. We then use these domain points as the initial iterate for a surface/surface iteration routine which attempts to find a true intersection point.

The surface/surface iteration routine solves the three non-linear equations

$$\mathbf{S}_1(s, t) - \mathbf{S}_2(u, v) = \mathbf{0} \tag{1}$$

in the four unknowns, s, t, u, v. Given the first iterate, $\hat{s}_1$, $\hat{t}_1$, $\hat{u}_1$, $\hat{v}_1$ say, we can analyze the linearized form of (1), which can be interpreted geometrically as describing the intersection between the tangent plane to $\mathbf{S}_1$ at $(\hat{s}_1, \hat{t}_1)$ and the tangent plane to $\mathbf{S}_2$ at $(\hat{u}_1, \hat{v}_1)$.

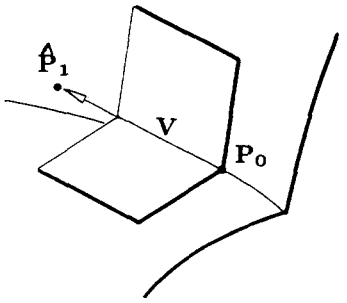


Fig. 3. Starting from a point $\mathbf{P}_0$, a 'guess point' $\hat{\mathbf{P}}_1$ is found in the direction $\mathbf{V}$ determined by the intersection of the two tangent planes at $\mathbf{P}_0$.

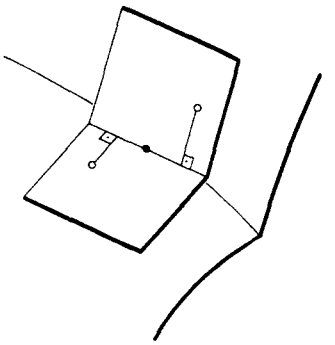


Fig. 4. Having found approximate 'intersection points' on each surface, the next iterate is set to be the midpoint of the projections of these two points onto the line of intersection of their corresponding tangent planes.

A point on this line of intersection is chosen as in Fig. 4 and the next iterate is generated by inverting the Jacobians (at the previous iterates) to obtain points on each domain. As with curve/surface iteration if this iteration diverges we subdivide the surfaces to obtain a better first iterate and try again. If the iteration continues to fail, we return to the point $\mathbf{P}_0$ and generate a new guess point by reducing the length of the vector $\mathbf{V}$ in hopes that this produces a candidate guess point closer to a true point on the intersection curve. If this fails, we stop following the curve in this direction and record the loose end.

If we do in fact find a new true intersection point, then we record it as $\mathbf{P}_1$ and consider the interval between it and $\mathbf{P}_0$. If this segment is of distance larger than CRT we 'refine' the approximation to the intersection curve on this interval. The method is to use a Bézier cubic in each domain to obtain guess points to which we apply the surface/surface iteration algorithm described above.

After refinement, we have a sequence of points between $\mathbf{P}_0$ and $\mathbf{P}_1$ spaced within distance CRT. We optimize this approximation of the intersection curve by discarding points that are collinear to within the tolerance OPT.

We then take the piecewise linear domain curves and record where they cross each of the isoparametric lines in the original rectangular approximation. We form lists for each isoparametric line of all such crossings for use as described in Section 3.3. We also use this list to determine if the curve we are following has collided with another curve. That is, before we record that the curve has crossed an isoparametric line, we check the list for this isoparametric line to see if any other curve has already crossed at this point. If so, we stop following the curve in this direction and record the loose end.

The algorithm stops following the curve when one of the following criteria is met:

(i) The curve hits a boundary of the surface or meets itself.

(ii) The curve hits the endpoint of a previously constructed intersection curve.

(iii) The curve crosses an isoparametric curve at 'nearly' the same point as a previously constructed curve.

(iv) If surface/surface iteration and decreasing the length of $\mathbf{V}$ has failed to produce a new intersection point.

When we stop following the curve, we record the endpoint of the curve and the reason why we stopped.

In cases (iii) and (iv), we record the information as a loose end of the curve to use in detecting if a subsequent curve meets this endpoint as in (ii). Thus, we keep a list of loose ends of intersection curves and check at each interval if the curve we are following hooks up with one of these. If so, we record the two curves as one and remove the loose end from the list of

possible loose ends. This technique handles situations that occur when intersection curves cross, or when we are unable to follow a curve in one direction (due to condition (iv)) but are able to follow it when we approach from the other direction.

After the curve has been followed until one of the preceding stopping criteria is met, the intersection curve is optimized again. This global optimization is necessary because the previous optimization was only local (between $\mathbf{P}_0$ and $\mathbf{P}_1$).

If the starting point is interior, the curve is then followed in the other direction from the starting point. After the curve has been followed in both directions, we return to Section 3.3 to search for starting points on other curves. After all isoparametric lines have been processed as in Section 3.3, the curves are output as sequential lists of points.

3.5. Discussion of the algorithm

As noted earlier, the key to obtaining both robustness and efficiency is to separate the tolerances as in Section 2.2. We utilize several different techniques to incorporate these tolerances into an overall algorithm that produces quality intersection curves quickly and efficiently.

The algorithm works efficiently due to the fact that the majority of effort is used in following the intersection curves. We utilize the marching method to focus on the intersection curves rather than on extraneous portions of the surface. This technique is relatively fast and produces actual points on the intersection curve but does nothing to ensure that all intersection curves are found.

The algorithm is robust because enough initial guesses are generated to obtain all intersection curves to within the tolerance SRT. This is accomplished by creating 'adaptive' initial approximations to the surface. This technique may be used to obtain all intersection curves but is very slow and the points on the intersection curves are not true intersection points.

The key ingredient in combining the above two techniques into an efficient and robust algorithm is to be able to record the information produced by following intersection curves in such a manner that it can be used to 'cull out' regions on the surface which no longer need to be considered. We use the structure of the isoparametric lines on the original approximation to do this and exploit it in subsequent searches for starting points on other intersection curves. We thus obtain the best features of each of the two techniques.

4. Extensions and special cases of the basic SSI algorithm

4.1. Infinite plane intersection with a patch

An important special case for the SSI algorithm is the intersection of a surface with a plane. Typically, a plane is given in normal form: $ax + by + cz + d = 0$. Such an equation, however, is the implicit form of an infinite surface, whereas SSI can only handle parametric (and finite) patches. We thus convert the normal form to parametric as follows: project the bounding box of the surface onto the given plane and check for non-interference of the box and the plane. In case of interference, form a bounding square (in the given plane) of those eight projected points. Represent that square as a parametric patch

$$\mathbf{S}(u, v) = \mathbf{A} + u \cdot \mathbf{B} + v \cdot \mathbf{C}.$$

This representation of the plane results in a gain in speed: the evaluations for the linear patch $\mathbf{S}$ provides constant u- and v-partials, together with constant normal vectors. Also, $\mathbf{S}$ is subdivided into only two triangles in step 1.

We note here that special algorithms exist to intersect a polynomial surface with a plane, see [Farouki '86]. Such specialized algorithms should provide a more efficient and robust means for the special case of polynomial-plane intersections.

4.2. Surfaces with creases

The basic SSI algorithm can handle continuous surfaces with a finite number of slope discontinuities. However, the curves produced may not correctly mirror the actual intersection curves due to the failure to detect the slope discontinuities. In this case the tolerance CRT can be decreased to improve the approximation. Afterwards, the tolerance OPT can be applied between points of slope discontinuity.

In many cases, it is desirable to produce piecewise linear approximations to intersection curves that have any slope discontinuities identified explicitly. In the remainder of this section we discuss a method for this task.

Let **S** and **T** be two surfaces. Assume that the curve $\alpha:[a, b] \rightarrow \mathrm{dom}(\mathbf{S})$ describes the curve along which the slope discontinuity occurs on **S**. We outline a strategy to find the curves of intersection of **S** and **T** under the added constraint that if a curve of intersection contains a tangent discontinuity, then the point of discontinuity should be found to within the tolerance SPT.

We first apply the SSI algorithm to the surfaces **S** and **T** with the parameters CRT and SRT set as if both surfaces were smooth. We then find all intersections of the range curve $\mathbf{S}(\alpha):[a, b] \rightarrow \mathbb{R}^3$ with the surface **T** and call the set of all such intersection points (in $\mathbb{R}^3$) D. If this set is empty, then no intersection curve has a slope discontinuity and the solution is complete.

Otherwise, if there are points of intersection, they represent exactly those points that we wish to insert on the intersection curves. The problem is then to determine where we should insert these points.

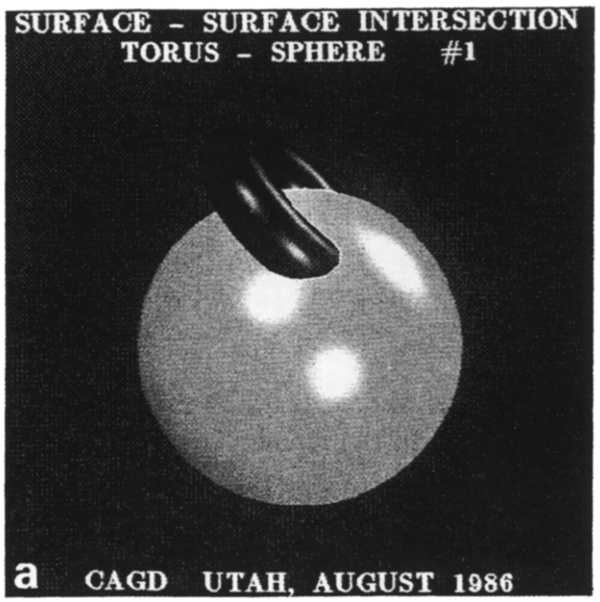
To do this, we find all intersections of the curve α with the domain versions of the intersection curves we constructed originally. (This is a two dimensional problem.) Suppose β is an intersection curve in the domain of **S** that intersects α at a point **p**. We insert into the representation of β the point in the set D that is closest to $\mathbf{S}(\mathbf{p})$. This point is the point of slope discontinuity in the intersection curve. Since a generic curve intersects a surface in isolated points, deciding which point to insert poses no problem.

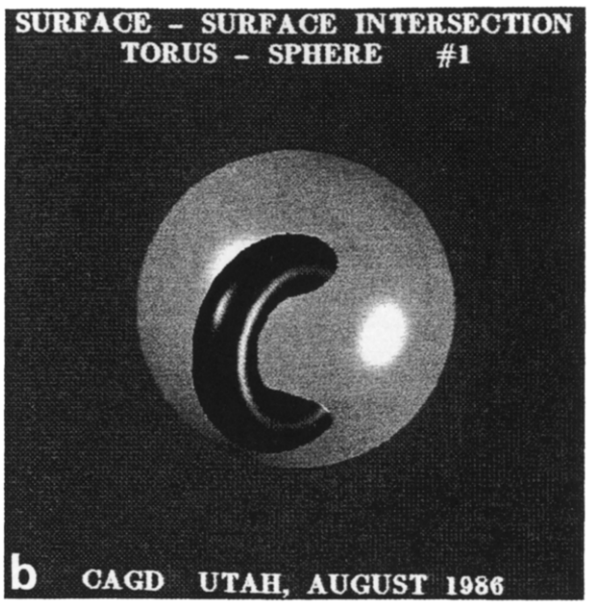
4.3. Self-intersection

Self-intersections of a surface are important in applications such as offset surfaces used in numerically controlled milling. SSI can be used to determine self-intersections by letting both SSI input surfaces be identical.

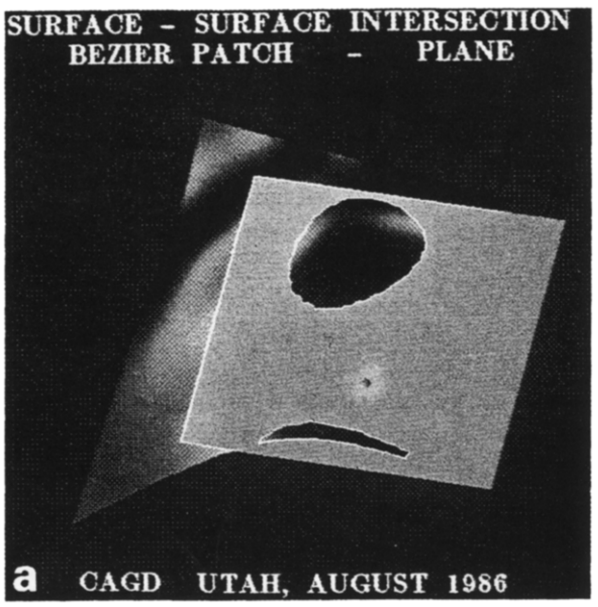
That is, we define the intersection of a surface with itself as a pair of *distinct* points in the domain that map to the same point in the range to within the tolerance SPT. Thus, we apply the SSI algorithm to two surfaces that are actually both the same surface and discard any intersection points we find that have the same domain values. This produces curves where the surface intersects itself.

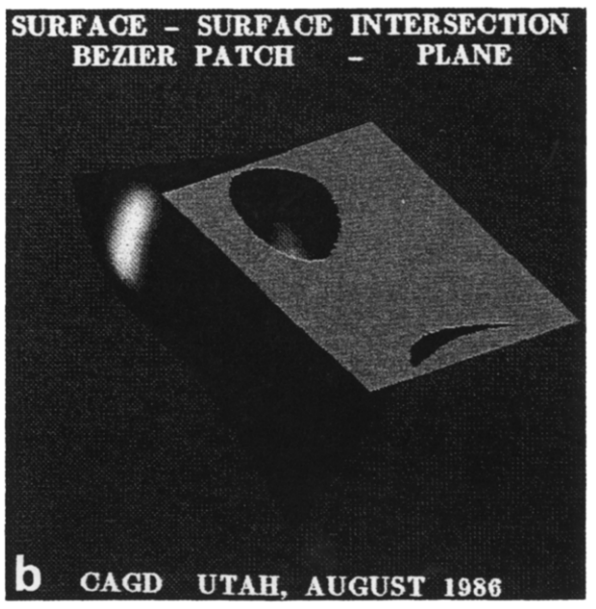
The discarding of intersection points found that have the same domain location is done primarily at the stage of finding initial intersection points. Thus, the algorithm does not spend too much extra time dealing with extraneous intersections. We must, of course, also check to make sure that the domain points remain distinct as we are following the curve. If they do not, we handle it as in case (iii) of the stopping criteria for the SSI algorithm



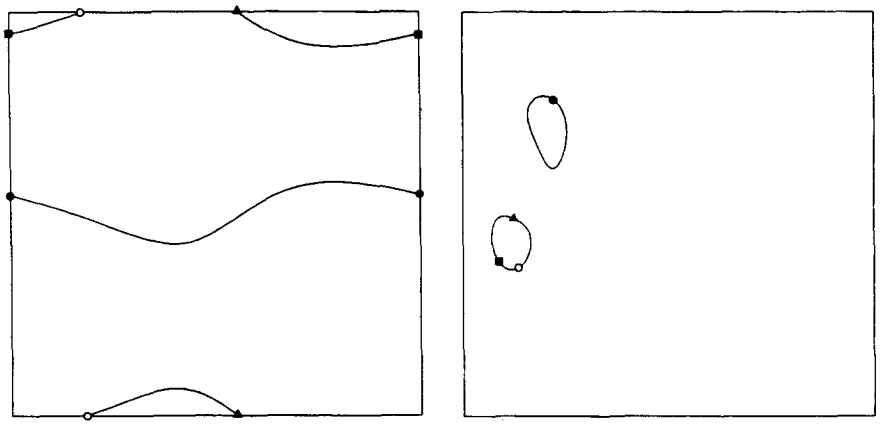


Figs. 5a, 5b. The torus/sphere intersection problem. The intersection is two disjoint closed curves.





Figs. 6a, 6b. The Bézier patch/plane intersection problem. In these figures the intersection curves appear very jagged due to the resolution of the z-buffer.



Figs. 5c, 5d. The pre-images of the intersection curves in the domains of the torus and the sphere respectively. The points marked by common symbols map to the same locations in $\mathbf{R}^3$.

We also note that we may save storage by keeping only one copy of the domain and initial approximation for the surface. In fact, it is necessary to keep only one copy of the domain of the surface to prevent finding two copies of each intersection curve. That is, we avoid finding the same intersection curve twice by recording both domain curves corresponding to one intersection curve in the same domain.

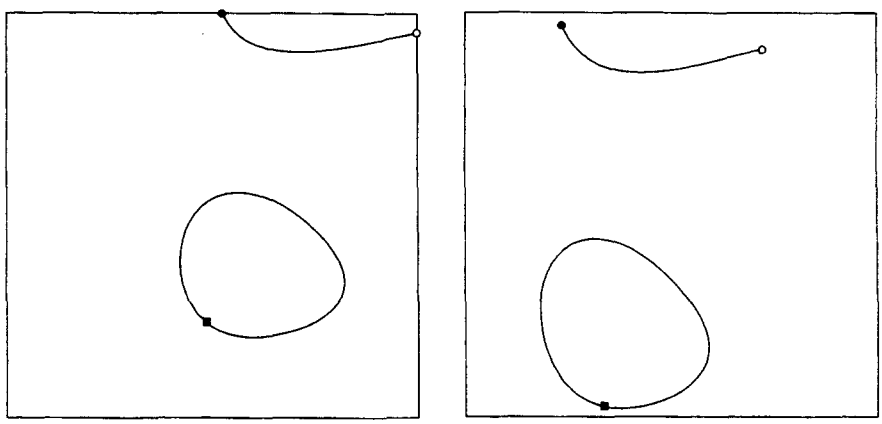
5. Examples

The basic SSI algorithm is illustrated with several examples. Each example has these elements:

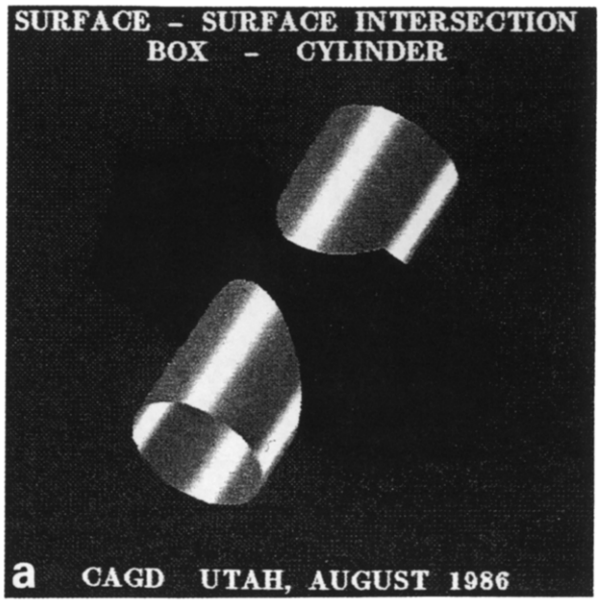
(1) Color pictures of the surfaces viewed in two different orientations.

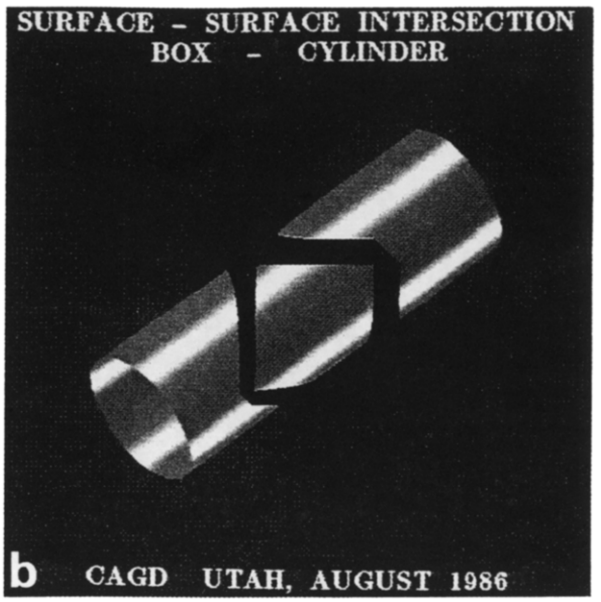
(2) The intersection curves in the two parametric domains.

The color pictures are rendered using several point light sources and Phong shading; see [Foley, Van Dam '82, pp. 577–584]. The intersections that appear in the color pictures are a result of z-buffering the image; see [Foley, Van Dam '82, pp. 506–561]. This technique does not in general produce the entire intersection curve. Also, due to the resolution of the device, the portions of the intersection curves that are detected are not always to within desired accuracy (cf. Fig. 6b).

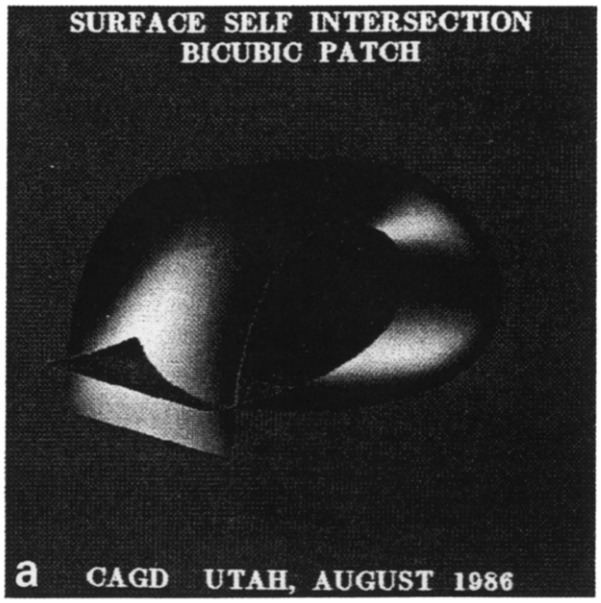


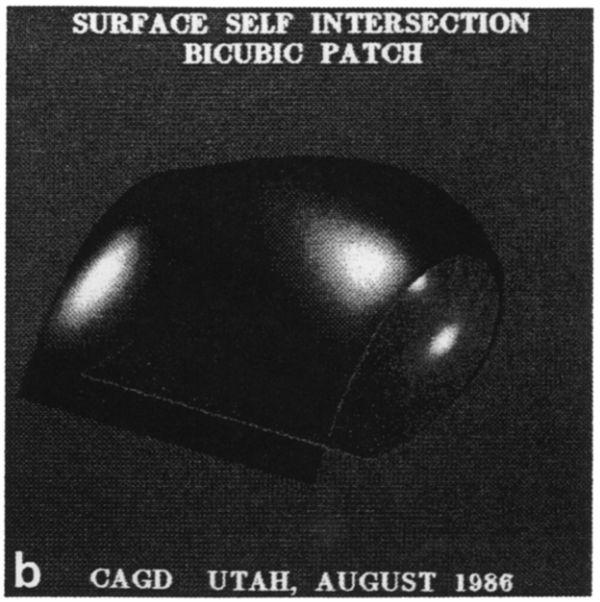
Figs. 6c, 6d. The pre-images of the intersection curves in the domains of the Bézier patch and the plane respectively. The points marked by common symbols map to the same locations in $\mathbf{R}^3$.



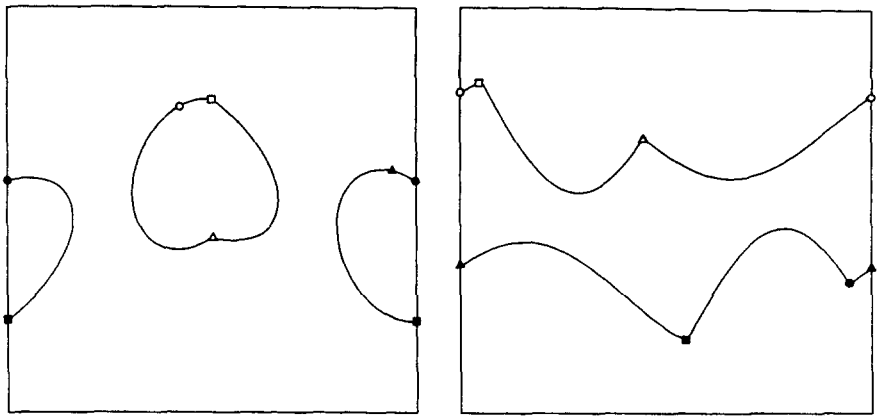


Figs. 7a, 7b. The box/cylinder intersection problem. The intersections are two closed curves with four slope discontinuities on each curve.





Figs. 8a, 8b. A bicubic patch which intersects itself.



Figs. 7c, 7d. The pre-images of the intersection curves in the domain of the box and the cylinder respectively. The points marked by common symbols map to the same locations in $\mathbf{R}^3$.

Thus, the color pictures of the surfaces do not solve the intersection problem. Rather they should be used as guidance to what to expect in the domain pictures. The domain pictures contain the actual solutions.

In each pair of domain pictures we mark by common symbols those points whose images agree (to within SPT) in the range. This illustrates the relationship between the two pre-images of the intersection curve.

We consider one example of the basic SSI algorithm and one each of the special applications listed in Section 4.

In Figs. 5a and 5b, two views of a torus and a sphere are given. The intersection curves in this case are two closed curves in $\mathbf{R}^3$. The domain intersection curves are illustrated in Figs. 5c and 5d. Notice that due to the parameterization, each intersection curve hits the boundary of the domain of the torus while they are closed curves in the domain of the sphere.

The second example (cf. Figs. 6a and 6b) illustrates a Bézier patch intersecting a plane. In this case we actually render only a finite piece of the infinite plane described in Section 4.1. The solutions are shown in Figs. 6c and 6d.

The third example is illustrated in Figs. 7a and 7b. In this example, we intersect a box with a cylinder. The resulting intersection curves have slope discontinuities apparent in the domain pictures in Figs. 7c and 7d.

The last example shows the case of intersecting a bicubic patch with itself. The bicubic patch is shown in Figs. 8a and b. We display in Fig. 8c the two pre-image curves giving rise to the curve of self intersection.

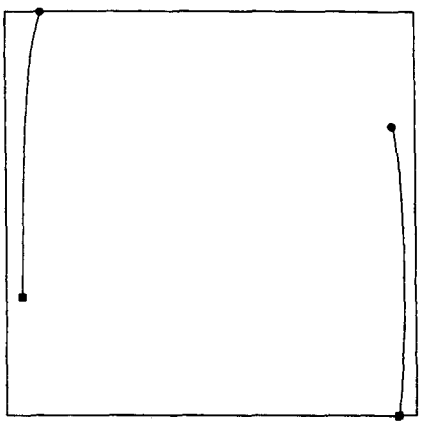


Fig. 8c. The two pre-images of the intersection curve in the domain of the bicubic patch. The points marked by common symbols map to the same locations in $\mathbf{R}^3$.

6. Conclusions and future research

We have presented a general surface/surface intersection algorithm for rectangular parametric C^1 patches each defined over a unit square. The algorithm has performed well on a variety of examples. However, we do not claim infallibility: any algorithm involving tolerances can be defeated by sufficiently ill-chosen examples. One class of examples difficult for SSI algorithms (cf. [Houghton et al. '85]) are surfaces which are tangent to each other. These types of surfaces frequently give rise to *surfaces* of intersection. However, we have used our SSI algorithm's output to detect this tangency case, after which counter-measures can be taken.

We listed a subset of the possible extensions and special cases of the algorithm in Section 4. There are, of course, many other possible uses for the ideas contained in the algorithm. In particular, we have extracted a curve/curve and a curve/surface intersection routine from the algorithm. In addition, we have applied a similar algorithm to the problem of finding the shortest distance from a point to a surface.

Future research could include a study of automatic choices of the tolerances, the use of curvature analyses to determine surface subdivision and application of the self-intersection algorithm to define offset surfaces.

Acknowledgments

This research arose from a collaboration with Dr. Rosemary Chang, Control Data Corporation, Arden Hills, Minnesota. We thank her and her coworkers at CDC, especially Mr. Bruce Brooks. The support of the Math CAGD Group at Utah was valuable, as always.

The authors acknowledge Control Data Corporation for their support of this work performed under a research/software development/consulting contract with them.

References

Farouki, R. (1987), Direct surface section evaluation, in: Farin, G., ed., *Geometric Modeling*, SIAM, Philadelphia, PA.

Foley, J.D. and Van Dam, A. (1982), *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA.

Houghton, E.G., Emnett, R.F., Factor, J.D., and Sabharwal, C.L. (1985), Implementation of a divide-and-conquer method for intersection of parametric surfaces, in: Barnhill, R.E. and Boehm, W., eds., *Surfaces in Computer Aided Geometric Design '84*, North-Holland, Amsterdam.

Pratt, M.J. and Geisow, A.D. (1986), Surface/surface intersection problems, in: Gregory, J.A., ed., *The Mathematics of Surfaces*, Oxford University Press, Oxford.