UC Davis
IDAV Publications

Title

An efficient surface-surface intersection algorithm using adaptive surface triangulations and space partitioning trees

Permalink

https://escholarship.org/uc/item/8dc9g25j

Journal

Mathematical Engineering in Industry, 7

Authors

Jean, B. A.
Hamann, Bernd

Publication Date

1998

Peer reviewed

An efficient surface–surface intersection algorithm using adaptive surface triangulations and space partitioning trees

BRIAN A. JEAN [1,*] and BERND HAMANN [2]

[1] *USAE Waterways Exp. Station High Performance Computing Center, CEWES–IH, 3909 Halls Ferry Road, Vicksburg, MS 39180–6199, USA*

[2] *Department of Computer Science and Center for Image Processing and Integrated Computing (CIPIC), University of California, Davis, CA 95616–8562, USA*

Abstract—We present a robust and efficient surface–surface intersection (SSI) algorithm. The algorithm is based on intersecting surface triangulations and improves the resulting linear spline curve approximations of the exact intersection curves by replacing intersection points of the surface triangulations by exact intersection points. We use a specialized data structure, the k-d tree, for the efficient intersection of surface triangulations. The resulting intersection curves are guaranteed to lie within a user-specified tolerance.

1. INTRODUCTION

Robust, accurate and efficient algorithms for the computation of surface–surface intersection (SSI) curves are important for many engineering applications, including grid generation. SSI curves, also called *trimming curves*, represent important features of complex three-dimensional (3D) geometries that must be captured by a surface grid. Grid points must be placed on the intersection curves of multiple surfaces, e.g., the SSI curves resulting from a wing–fuselage intersection. One can find detailed information about grid generation methods in George (1991), Knupp and Steinberg (1993), and Thompson *et al.* (1985). A complex 3D geometry description is most often provided in the form of parametric surfaces, e.g., Bézier, B-spline or

---

NURBS surfaces (see Farin, 1997, Hoschek and Lasser, 1993, Mortenson, 1985 or Piegl and Tiller, 1996).

Even when defined in terms of standard data exchange formats, surface descriptions often do not contain the SSI curve information – or the SSI curves are not in a format suitable for further processing. Thus, grid generation systems (should) have a SSI capability for various standard data formats, parametric and polyhedral surface representations being the most important ones. It is very important that the resulting SSI curves are within prescribed tolerances to allow packing of grid points very close to the exact surface intersections. Flow simulation algorithms typically require that the SSI curves be accurate to at least 10^{-6} units. A practically useful SSI method should be robust and require minimal user input. A user should have to specify only the surfaces to be intersected and the requested tolerance. This is exactly the input to our algorithm. Use in an interactive environment requires that an SSI method be reasonably fast, i.e., the solution of all but the most difficult problems should take only seconds on a state-of-the-art workstation. We use a hierarchical spatial data structure, a *k-d tree*, to represent surface triangulations and can thereby achieve fast processing.

It is beyond the scope of this paper to review the literature on the SSI problem. One can find various "standard" SSI techniques described in Barnhill (1992), Barnhill and Kersey (1990), and Patrikalakis (1993). It should be noted that Houghton and Emnett (1985) contribute an approach similar to ours, however there are some important differences. Most importantly, our algorithm can operate on discrete data with no parametric description. Secondly, we eliminate the sorting step by using a sophisticated solid modeling like data structure that automatically builds an explicit piecewise linear representation of the intersection curves during the triangle intersection phase.

The trimmed surfaces, resulting from any SSI algorithm, cause additional problems in the context of surface rendering, grid generation, and data exchange/conversion. Solutions to these additional problems are discussed in Brunnett (1995), Hamann and Jean (1996), Hamann and Tsai (1996), Hoschek and Schneider (1990), Piegl and Richard (1995), Rockwood *et al.* (1989), and Vries-Baayens and Seebregts (1992).

2. OVERVIEW OF THE SSI ALGORITHM

Our method is used in an interactive grid generation system. The grid generation process for complex 3D geometries, often consisting of several thousand individual surfaces, imposes constraints on the computation of SSI curves. These constraints have driven the development of our algorithm and are as follows:

– **Accuracy.** SSI curves must be highly accurate. Grid generation in the context of fluid flow simulation requires a high degree of precision in order to capture fine detail in high-gradient regions, which usually includes regions close to SSI curves. An SSI algorithm must yield curves within a prescribed tolerance.

- **Efficiency.** In an interactive environment it must be possible to compute SSI curves in a few seconds.

- **Robustness.** An SSI algorithm must correctly determine multiple intersections among multiple surfaces, including SSI curve *bifurcations*.

- **Simplicity.** In an interactive environment the only action required by the user should be the specification of the surfaces to be intersected and the requested tolerance.

At present, no SSI algorithm possesses all of these properties. This is, in part, due to the fact that an 'optimal algorithm' for a particular intersection problem depends on the type of surfaces involved and the representation of those surfaces, i.e., whether surfaces are described in polyhedral, parametric or implicit form.

Our algorithm can operate on discrete surface representations, i.e., surface triangulations, or parametrically defined surfaces. If the intersection is to be performed for a discrete surface representation – and if this discrete form is the only surface description available – then the intersection curves of the surface triangulations are the final result (these curves are linear spline curves). If a parametric surface description is known, then we perform an additional *refinement* step to relax the intersection points of the surface triangulations to points lying on the exact SSI curves. These are the steps of our algorithm:

1. Generate an initial surface triangulation for all surfaces to be intersected (the triangulations might be given).

2. Store each surface triangulation in an efficient data structure, a spatial tree structure (we use a hierarchical, k-d tree structure).

3. 'Intersect the k-d trees' to obtain lists of potentially intersecting triangles.

4. Intersect individual triangles to obtain a collection of line segments.

5. Sort the line segments, i.e., construct a valid enumeration of the line segments defining an SSI curve approximation.

6. If a parametric surface representation is known, refine the points on the initial curve approximations by mapping triangle intersection points to points on the exact SSI curves (refinement is done to obtain better curve approximations satisfying a specified tolerance).

7. Represent the various branches of the SSI curves in either linear or cubic spline form, in either 3D physical or 2D parameter space.

When more than two surfaces are to be intersected, we iteratively construct all pairs of surfaces to be considered and process each pair individually. In the case of multiple surfaces intersecting each other, one might have to perform additional curve–curve intersections of the resulting SSI curves.
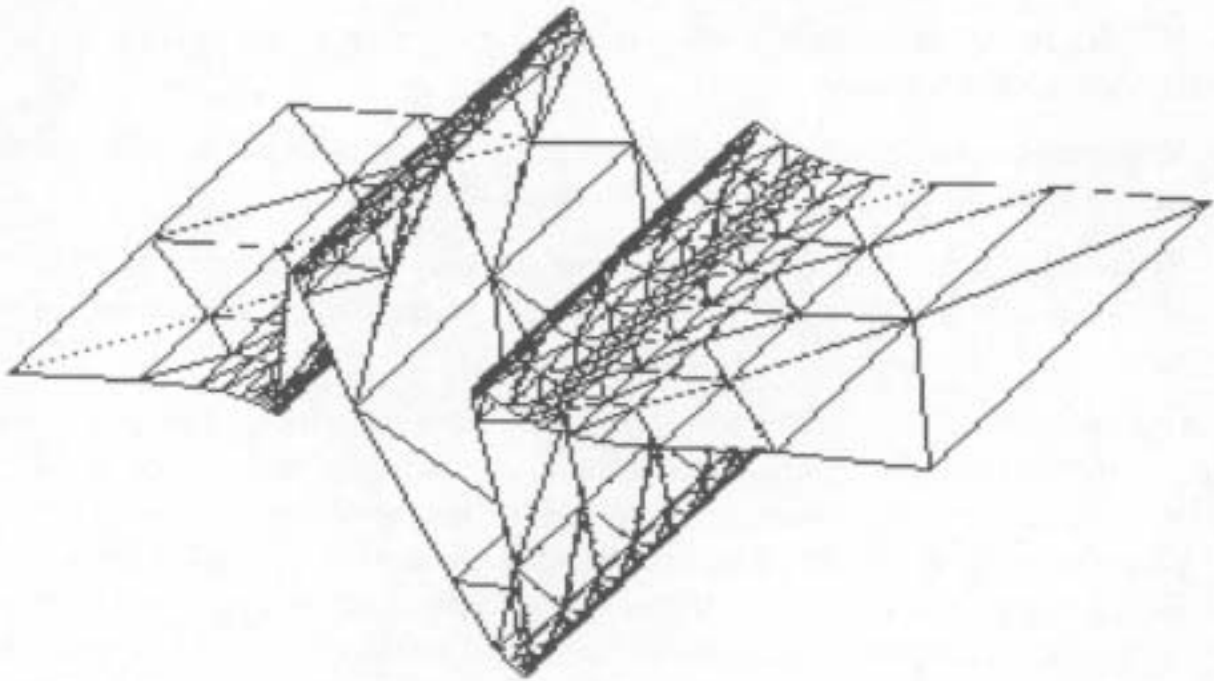
Figure 1. Adaptive surface triangulation.

3. INTERSECTION OF SURFACE TRIANGULATIONS

In most practical cases, intersections of parametric surfaces are the primary concern. We triangulate surfaces of this type by using an *adaptive triangulation* technique based on recursive subdivision (see Anderson *et al.*, 1997, Samet, 1990). This allows us to construct surface triangulations lying within a specified tolerance to underlying parametric surfaces. A triangle lies within a specified tolerance when the shortest distance from its centroid to the surface is no more than the tolerance. An adaptive surface triangulation of a highly oscillating surface is shown in Fig. 1.

A triangulation is represented by a list of vertices and a connectivity table. Each vertex is stored only once in order to reduce memory requirements and to eliminate the possibility of slight edge mismatches due to numerical error. In the case of parametric surfaces, separate lists for associated parameter values – we refer to them as (u, v)-parameter values and a (u, v)-connectivity tables – are maintained to allow later refinement of the SSI curves resulting from the intersection of surface triangulations.

Before performing the actual intersection operation for the two planes defined by two triangles, we apply a bounding box test to see if they can possibly intersect. Each triangle bounding box is constructed to be just large enough to contain all three vertices of the triangle and is then expanded by 2ε along each axis (x, y, and z). The test for candidacy is passed if the bounding boxes of the two triangles overlap by ε or more in any direction[1]. If the test is passed, then we intersect the edges of one triangle with the plane defined by the other triangle and *vice versa*. The

---

[1] In the implementation, the tolerance ε is a parameter passed to the intersection routine.

points resulting from these edge–plane intersections are then tested, using triangle sub-areas, to determine if they lie inside the respective triangles. Triangle sub-areas are derived by taking a Euclidean distance vector from the intersection point to each of the triangle vertices; each of the three sub-areas of a triangle are determined by a one side of the triangle and the two distance vectors associated with the sides' end points. If all three sub-areas of the triangle are non-negative, then the point lies inside the triangle or on one of its edges[2]. The triangle–triangle intersection step can yield one of three possible results:

- No intersection point lies in the inside of either of the two triangles, i.e., the triangles do not intersect.
- One intersection point is found, i.e., the triangles 'just touch' at their edges or a single vertex of one triangle lies in the interior or on the edge of the other triangle.
- Two intersection points are found, i.e., the intersection of the two triangles is the line segment defined by the two intersection points.

We consider only those triangle–triangle intersections that result in line segments, i.e., the third case, and store them.

4. INTERSECTION OF SURFACE TRIANGULATIONS AND RELATED DATA STRUCTURES

It is important to use elegant data structures to reduce the amount of time it takes to process the intersection of the surface triangulations. The bounding box test that we employ to find out whether two triangles can or cannot intersect is fast. However, each triangle must be compared with each other triangle, which leads, when using a naive approach, to a processing time that is quadratic in the number of triangles. We therefore store the triangles in a so-called *space-partitioning* tree. The tree partitions the space occupied by the triangles and provides quick access to the set of triangles which inhabit a particular region.

The particular type of space-partitioning tree we use is a k-d tree (see Bentley, 1975, Samet, 1990). Given N triangles, a k-d tree will have at most $2N$ nodes with N leaf nodes, each containing exactly one triangle. A node stores a bounding box and an integer tag. The bounding box is specified by two points in 3D space and is just large enough to contain the bounding boxes of all of its children. The tag identifies, for a leaf node, the triangle which is contained in the bounding box associated with the leaf. We construct a separate k-d tree for each surface. We 'intersect two trees' – one referred to as *base*, the other one as *target* – by performing these steps:

1. Pick a leaf of the base.

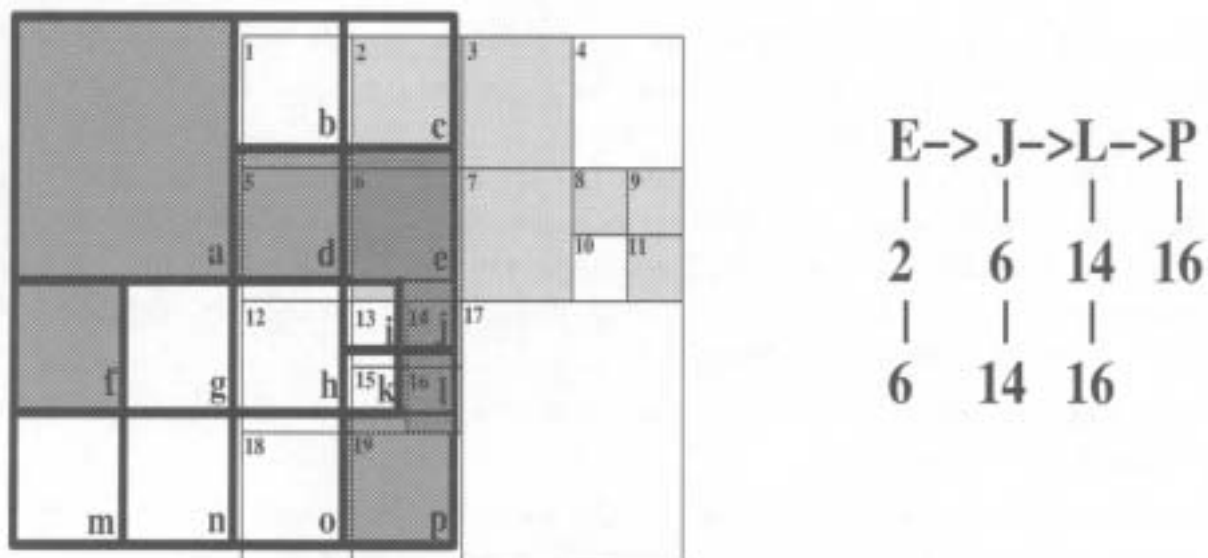[2] A sub-area is non-negative if it is positive or within ε of zero.

Figure 2. Intersection of quadtrees and resulting node association list (shaded squares being non-empty, geometry-containing areas).

2. Intersect the base leaf with the target using recursive bounding box tests.

3. Link the base leaf to each target leaf that intersects it. If the base leaf does not intersect any target leaves, then it is no longer considered.

4. Repeat this procedure for each leaf in the base.

The set of links resulting from step 3 of this algorithm is the desired output. This set encompasses all possible triangle intersections for the two surfaces. Target leaves may be associated with multiple base leaves. However, each base leaf appears only once. Figure 2 is an illustration of a 2D example. The k-d tree is a binary tree and can be searched in logarithmic time. Hence, we can intersect two surfaces, represented by M and N triangles, in $M \log_2 N$ time.

5. THE TOPOLOGY OF THE SSI CURVES AND RELATED DATA STRUCTURES

The result of the intersection of the adaptive surface triangulations is a set of unorganized line segments which, when sorted properly, provide linear spline approximations of the SSI curves. We will later refine the points constituting these approximations by mapping them onto the exact SSI curves. Besides computing all the line segments, we need to determine the topology of the intersection curve 'network'. This requires ordering all line segments and identifying particular branches, including closed curve branches.

The detection of individual intersection curve branches, referred to as *loops* in the following, utilizes special data structures as well. Identifying the correct topology of the SSI curve 'network' is one of the most involved steps of SSI computations. There might be closed curves, intersecting curves, bifurcating curves, and various other degenerate cases. We describe a data structure that we have found to be very useful in the context of topology determination. The data structure stores the connectivity between the line segments of a linear spline approximation of an SSI

curve and allows us to trace and identify the individual branches of entire curve 'network'.

The *Point* structure stores the x, y, and z coordinates for each point and, in the parametric case, its associated parameter values for each of the two surfaces. For each Point, we store an associated circular-linked list of *PointUse* structures. PointUse structures contain connectivity and other topological information of a Point. Each Point is unique: If a Point is computed having the same coordinates as an existing Point[3], then a PointUse with the appropriate information is added to the list of PointUses for the already existing Point. The Point and PointUse structures, along with the segment structure, explained below, define the topology of the SSI curves. These entities are adaptations of similar structures devised by Weiler (1986) for *solid modeling* applications.

The PointUse structure contains topological information about a Point and a *Segment* (=line segment) structure. The Segment structure provides Point connectivity information based on PointUse structures. A Point shared by two Segment structures has two associated PointUse structures. Since both PointUse structures refer to the same Point, they are linked and hence the Segment structures are linked as well. We also store the two triangles defining a Segment for use in refinement of the SSI curves.

Figure 3 illustrates our topology data structure. We depict the intersection of four triangles, belonging to two different surfaces, resulting in four intersection points, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$. This example yields one loop whose two end points are $\mathbf{p}_1$ and $\mathbf{p}_4$.

The PointUse structure contains a *Location* field indicating where the PointUse is located on its associated Segment. The Location field is either zero or one indicating the 'start' or 'end' point of the line segment. A *P* field and an *SSISegPtr* field provide links to the associated Point and Segment structures. A *Prev* and a *Next* field link a PointUse to others (if any) in the circular-linked list of PointUse structures.

We define an *end point* as a Point with a number of associated PointUse structures not equal to two (closed curves are a special case where all Point structures have two associated PointUse structures). If more than two PointUse structures are associated with an end point, we are dealing with a point where three or more SSI curves come together. We automatically split SSI curves at such points, which include points where curve bifurcations occur.

In summary, we determine the topology of the SSI curves by performing these steps:

1. Find a Point with a number of associated PointUse structures different from two and at least one associated PointUse structure with its *InUse* flag, a Boolean value, being FALSE. If there is no such Point, stop.

---

[3]Points are considered to be coincident if the Euclidean distance between them is ε or less.

P: Point
PU: Point Use
S: Segment

$$PU \longleftrightarrow S_1 \longleftrightarrow PU \longleftrightarrow PU \longleftrightarrow S_2 \longleftrightarrow PU \longleftrightarrow PU \longleftrightarrow S_3 \longleftrightarrow PU$$
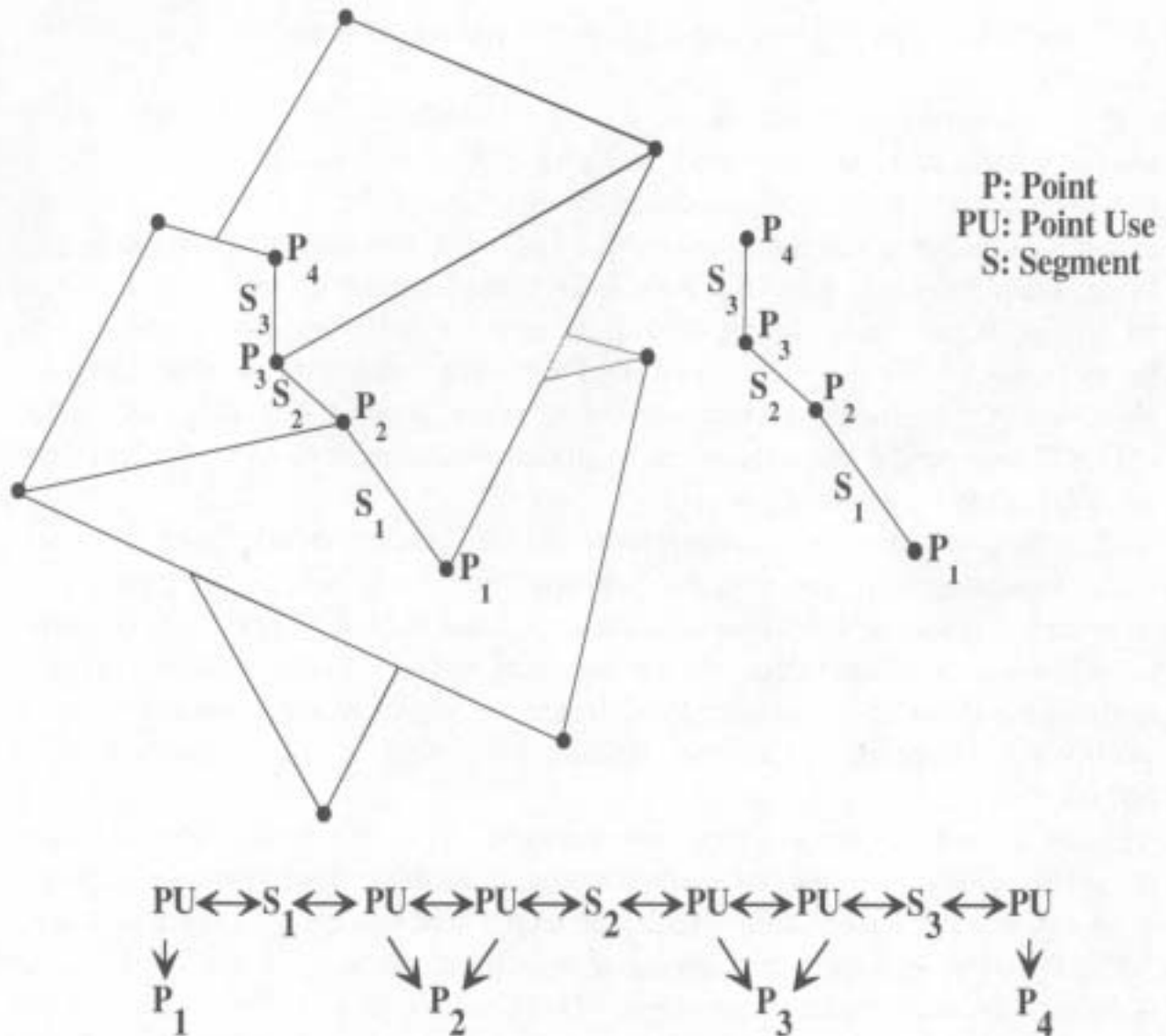
$$P_1 \qquad P_2 \qquad P_3 \qquad P_4$$

Figure 3. Data structure used to represent SSI curve topology.

2. For the PointUse found in step 1, set its InUse flag to TRUE and add the Point to the ordered list of Points on this SSI curve.

3. Identify the Segment associated with this PointUse and set its InUse flag to TRUE.

4. Identity the 'opposite' PointUse of this Segment.

5. Regarding this 'opposite' PointUse, set its InUse flag to TRUE and add its associated Point to the list of ordered Points on this SSI curve.

6. If the number of PointUse structures associated with the current Point is two, consider the other PointUse associated with the Point and continue with step 3 otherwise, continue with step 7.

7. Store the list of ordered Points of subsequent SSI curve refinement.

8. Repeat this process until the topology of the entire SSI curve 'network' is defined.

Closed curves require a special case treatment. When closed curves (or closed curve branches) are present, the above algorithm will leave certain Segment struc-

tures 'unused'. We deal with closed curves by picking a random 'starting' PointUse from the 'unused' PointUse structures and otherwise apply the same algorithm. In general, a Point can have an arbitrary number of PointUse structures associated with it. This tracing algorithm suffices to handle all possible topological cases[4].

6. ADAPTIVE SSI CURVE REFINEMENT

The result of the above curve tracing method is a complete description of the SSI curve topology. When dealing with parametric surfaces, the points defining the linear spline approximations to all SSI curves do not lie on the parametric surfaces. In the following, we assume that we know a parametric surface definition and use it for refinement. This means that we map the points of our initial piecewise linear SSI curve approximations onto the exact, underlying surfaces.

Denoting a point lying on an linear spline approximation by $\mathbf{a}$, we 'project' it onto the associated parametric surfaces by considering the barycentric coordinates of $\mathbf{a}$ with respect to the vertices of the triangles containing $\mathbf{a}$. We know the (u, v)-parameter values for each vertex in the surface triangulations; if the barycentric coordinates of $\mathbf{a}$ with respect to a particular triangle T are α_1, α_2 and α_3, we associate the tuple $(\bar{u}, \bar{v}) = \sum_{i=1}^{3} \alpha_i (u_i, v_i)$ with $\mathbf{a}$, where T's vertices have parameter values (u_i, v_i). We 'project' $\mathbf{a}$ onto the underlying surface by computing the surface point for the parameter tuple $(\bar{u}, \bar{v})$.

Having 'projected' all points of all initial SSI curve approximations onto the associated, underlying parametric surfaces, we are in a position to refine the projections. When 'projecting' an intersection point of two triangles, the resulting two projections onto the two underlying surfaces are, in general, not the same. Refinement is the process of using the parametric definition of the surfaces to compute an exact intersection point, a point lying on the exact intersection curve of the two surfaces. We use a so-called *auxiliary plane* approach for this purpose (see Barnhill and Kersey, 1990, Hosaka, 1992). Figure 4 illustrates the data needed for the refinement process.

We denote the 'projections' of an intersection point of two triangles onto the two underlying surfaces $\mathbf{r}(w, t)$ and $\mathbf{s}(u, v)$ by $\mathbf{p}_0 = \mathbf{r}(\bar{w}, \bar{t})$ and $\mathbf{q}_0 = \mathbf{s}(\bar{u}, \bar{v})$. Let F_p and F_q be the tangent planes at $\mathbf{p}_0$ and $\mathbf{q}_0$, and let $\mathbf{n}_p$ and $\mathbf{n}_q$ be the respective (outward) unit normal vectors at $\mathbf{p}_0$ and $\mathbf{q}_0$. The shortest (perpendicular) distances d_p and d_q from the (global) origin to F_p and F_q are

$$d_p = \mathbf{n}_p \cdot \mathbf{p}_0 \quad \text{and} \quad d_q = \mathbf{n}_q \cdot \mathbf{q}_0. \tag{1}$$

---

[4] As noted later, the SSI algorithm can fail to capture intersection curves near points where the two surfaces are touching tangentially. This is not due to a failure of the tracing algorithm; it results from the use of triangle intersections to determine the piecewise linear approximation of the intersection curves.
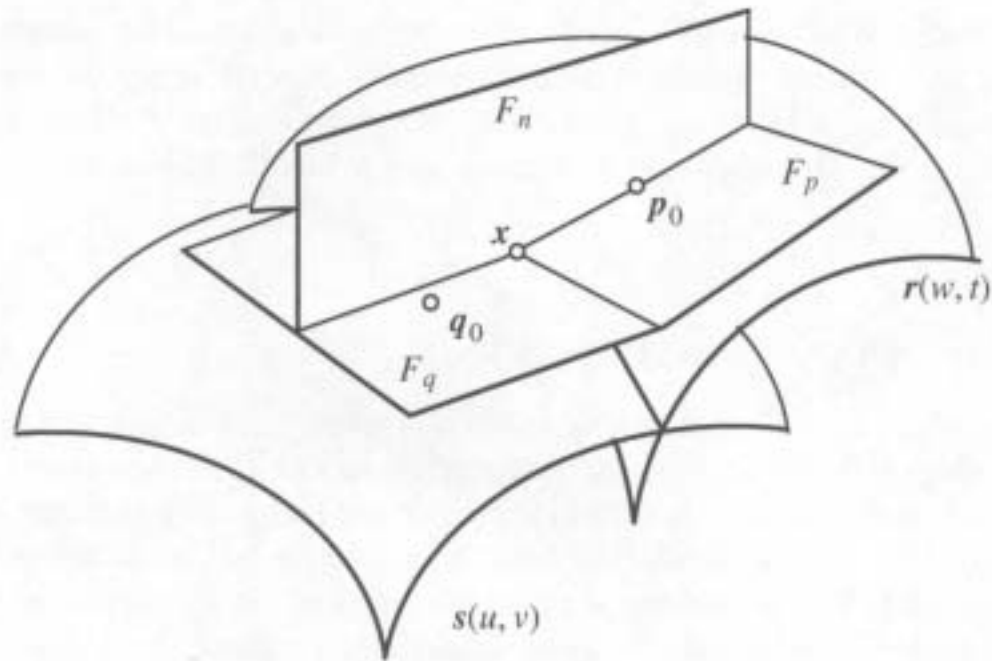
Figure 4. Data needed for iterative refinement of SSI intersection points.

We construct a plane F_n that is orthogonal to both F_p and F_q and passes through $\mathbf{p}_0$. The unit normal $\mathbf{n}_n$ of F_n and its shortest (perpendicular) distance d_n from the origin are

$$\mathbf{n}_n = \frac{\mathbf{n}_p \times \mathbf{n}_q}{\|\mathbf{n}_p \times \mathbf{n}_q\|} \quad \text{and} \quad d_n = \mathbf{n}_n \cdot \mathbf{p}_0, \tag{2}$$

where $\| \ \|$ denotes the Euclidean norm. The intersection point $\mathbf{x}$ of the planes F_p, F_q, and F_n is

$$\mathbf{x} = \frac{d_p(\mathbf{n}_q \times \mathbf{n}_n) + d_q(\mathbf{n}_n \times \mathbf{n}_p) + d_n(\mathbf{n}_p \times \mathbf{n}_q)}{[\mathbf{n}_p, \mathbf{n}_q, \mathbf{n}_n]}, \tag{3}$$

where $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ is the *scalar triple product* $(\mathbf{v}_1 \times \mathbf{v}_2) \cdot \mathbf{v}_3$ of three 3D vectors (see Hosaka, 1992). The point $\mathbf{x}$ defines the next approximation to an exact intersection point of $\mathbf{r}(w, t)$ and $\mathbf{s}(u, v)$*. In general, $\mathbf{x}$ does not lie on either of the two surfaces.

Denoting the partial derivative vectors (not normalized) at $\mathbf{p}_0$ by $\mathbf{r}_w$ and $\mathbf{r}_t$ and at $\mathbf{q}_0$ by $\mathbf{s}_u$ and $\mathbf{s}_v$, one can express the difference vectors $\delta \mathbf{p}_0 = \mathbf{x} - \mathbf{p}_0$ and $\delta \mathbf{q}_0 = \mathbf{x} - \mathbf{q}_0$ as linear combinations of these partial derivative vectors. For infinitesimally small values the two equations

$$\mathbf{r}_w \delta w + \mathbf{r}_t \delta t = \delta \mathbf{p}_0 \quad \text{and} \quad \mathbf{s}_u \delta u + \mathbf{s}_v \delta v = \delta \mathbf{q}_0 \tag{4}$$

hold. In order to determine increments in the parameter spaces of the two underlying surfaces, we need to compute four vectors which are orthogonal to the partial derivative vectors and the normal vectors at the two respective surface points. These

---

*In the implementation, the solution of Eqn (3) requires special case logic to deal with the possibility of a denominator that approaches zero.

four vectors are

$$\acute{\mathbf{r}}_w = \mathbf{r}_w \times \mathbf{n}_p, \quad \acute{\mathbf{r}}_t = \mathbf{r}_t \times \mathbf{n}_p, \quad \acute{\mathbf{s}}_u = \mathbf{s}_u \times \mathbf{n}_q \quad \text{and} \quad \acute{\mathbf{s}}_v = \mathbf{s}_v \times \mathbf{n}_q. \quad (5)$$

Based on Eqns (4) and (5) we obtain the desired increments in the two parameter spaces,

$$\delta w = \frac{\acute{\mathbf{r}}_t \cdot \delta \mathbf{p}_0}{\acute{\mathbf{r}}_t \cdot \mathbf{r}_w}, \quad \delta t = \frac{\acute{\mathbf{r}}_w \cdot \delta \mathbf{p}_0}{\acute{\mathbf{r}}_w \cdot \mathbf{r}_t}, \quad \delta u = \frac{\acute{\mathbf{s}}_v \cdot \delta \mathbf{q}_0}{\acute{\mathbf{s}}_v \cdot \mathbf{s}_u}, \quad \text{and} \quad \delta v = \frac{\acute{\mathbf{s}}_u \cdot \delta \mathbf{q}_0}{\acute{\mathbf{s}}_u \cdot \mathbf{s}_v}. \quad (6)$$

Thus, the updated values of $\mathbf{p}_0$ and $\mathbf{q}_0$ are

$$\mathbf{p}_0 = \mathbf{r}(\overline{w} + \delta w, \overline{t} + \delta t) \quad \text{and} \quad \mathbf{q}_0 = \mathbf{s}(\overline{u} + \delta u, \overline{v} + \delta v). \quad (7)$$

We repeat these steps until the value $\|\mathbf{p}_0 - \mathbf{q}_0\|$ is within a specified tolerance ε_2. We have observed that this method is extremely fast and robust. The SSI curve approximations resulting from the intersection of the initial surface triangulations may or may not meet the requested tolerance ε_2. If, after refinement, the distribution of exact SSI points is too coarse (or too fine) we add (or delete) points as required. To add points we compute the midpoints of two consecutive parameter tuples of exact intersection points, compute the corresponding two points in 3D space, and refine the two points to another exact intersection point.

The final representation of the SSI curve approximations curve depends on the requirements of a particular application. When desired, we compute a cubic spline representation of the SSI curves by interpolating all exact SSI points. We can represent SSI curves in both 3D physical and 2D parameter space.

7. EXAMPLES

Figures 5–10 show results of our algorithm. Figure 5 shows the intersection of two cubic B-spline surfaces, Fig. 6 the intersection of two tori, Fig. 7 the intersection of a plane and a B-spline surface, Fig. 8 a wing-fuselage intersection, Fig. 9 the intersection of the sea surface and some of the Hawaiian islands, and Fig. 10 a cylinder-torus intersection (with bifurcating SSI curves). All surfaces are defined as non-uniform rational B-spline (NURBS) surfaces. The various geometries are rendered as Gouraud-shaded triangulations with superimposed SSI curves.

For each example we provide a *maximal error*, which we define as the maximum of the *relative errors* associated with each SSI curve pair (each SSI curve is a pair of curves – one curve associated with each one of the two surfaces being intersected). For a particular curve (pair), we obtain the relative error by dividing the maximal Euclidean distance between the pair by the average of their respective arclengths.

In addition to the maximal errors, we list the arc length of the curve having the largest relative error, the total number of triangles used to obtain the initial approximations, and the processing times needed to compute the final SSI curves from the given B-spline surface definitions. The times are based on our implementation running on an SGI IRIS Indigo R4000 workstation.
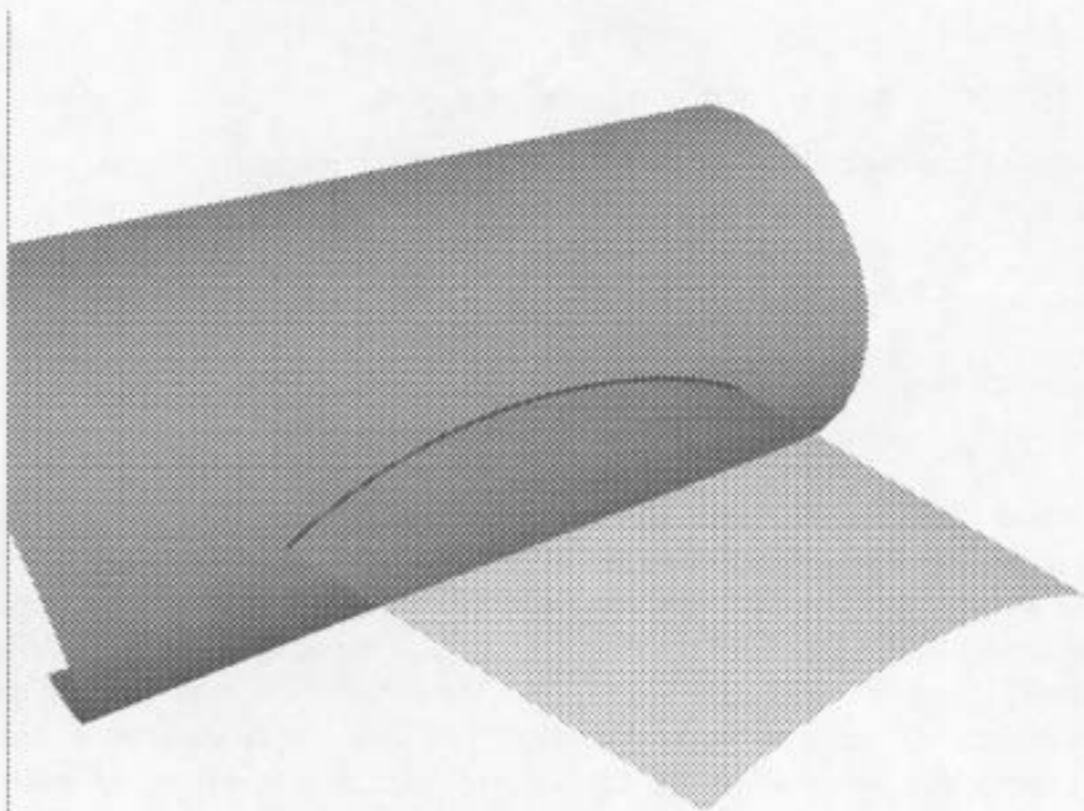
Figure 5. Intersection of two B-spline surfaces (max. error: 2.51576e−06; arc length: 6.00617; number of triangles: 800; time: 0.31 s).
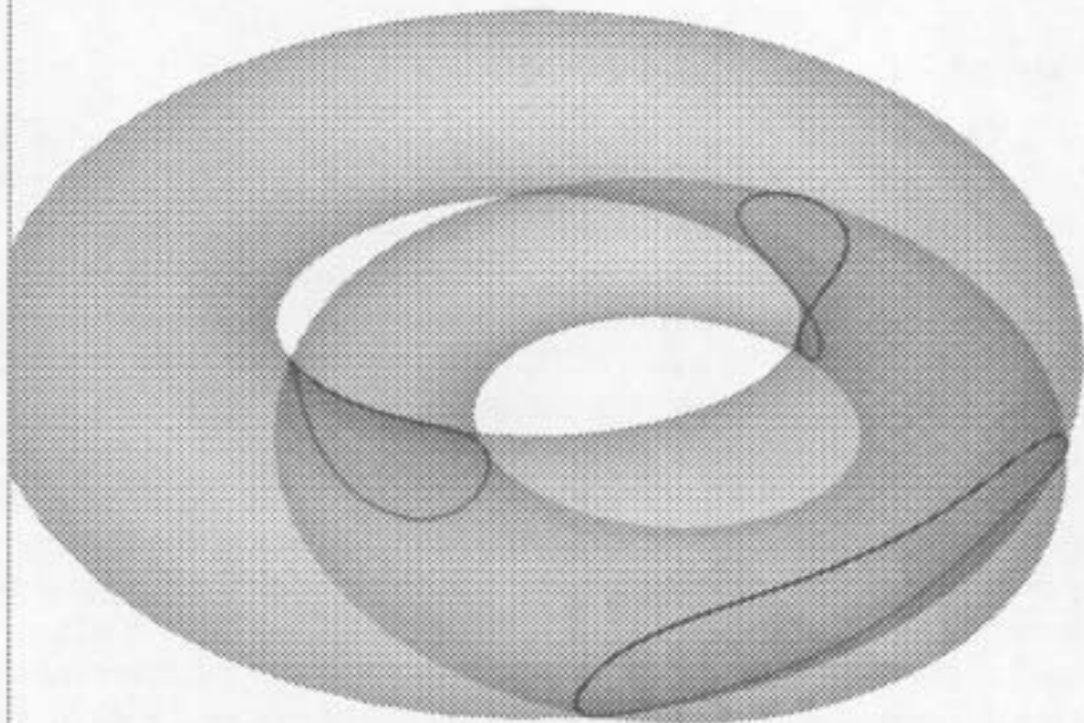

Figure 6. Intersection of two tori (max. error: 9.5097e−06; arc length: 6.79965; number of triangles: 4,492; time: 1.52 s).
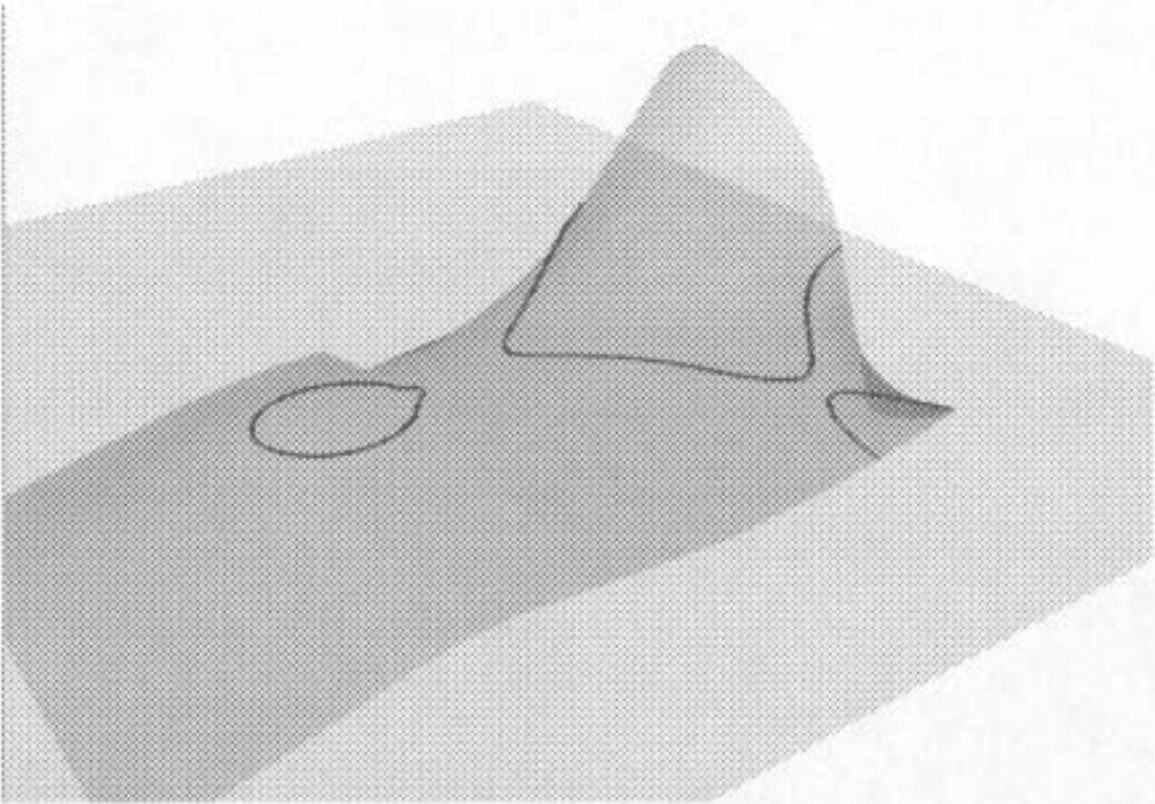
Figure 7. Intersection of plane and B-spline surface (max. error: 1.5772e−07; arc length: 4.94811; number of triangles: 5,898; time: 0.71 s).
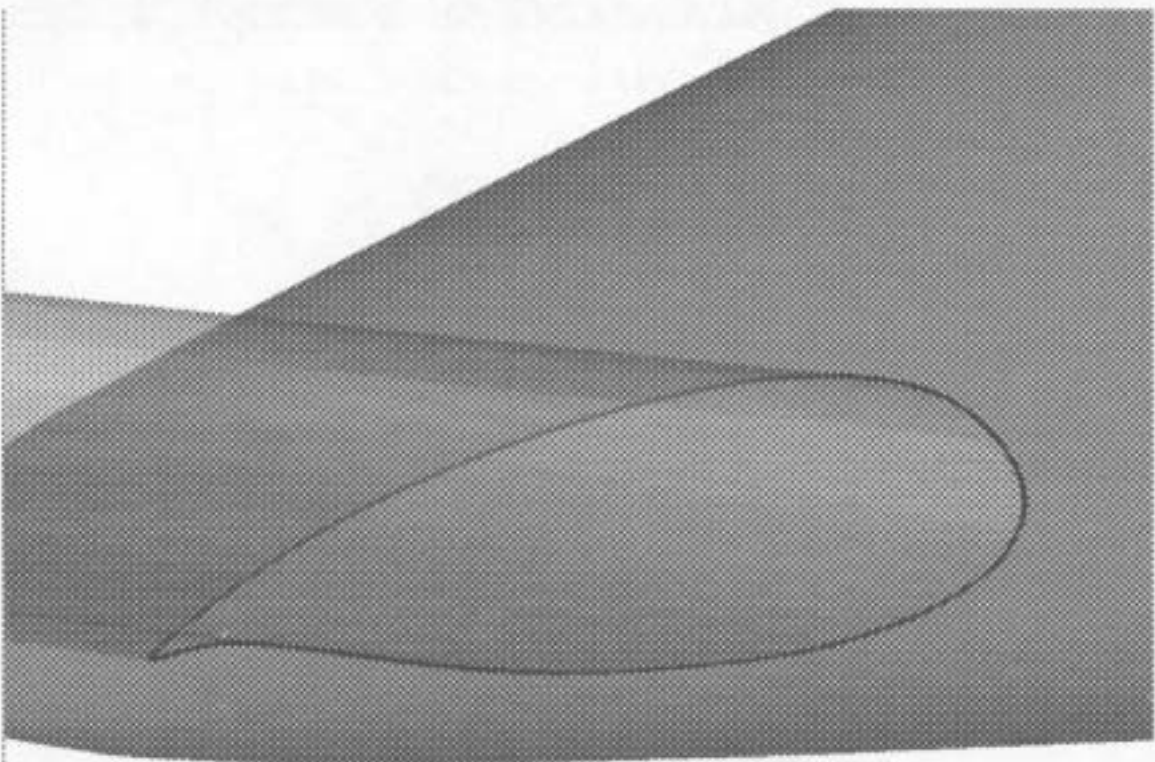

Figure 8. Wing–fuselage intersection (max. error: 7.2632e−07; arc length: 0.766139; number of triangles: 5,000; time: 0.58 s).
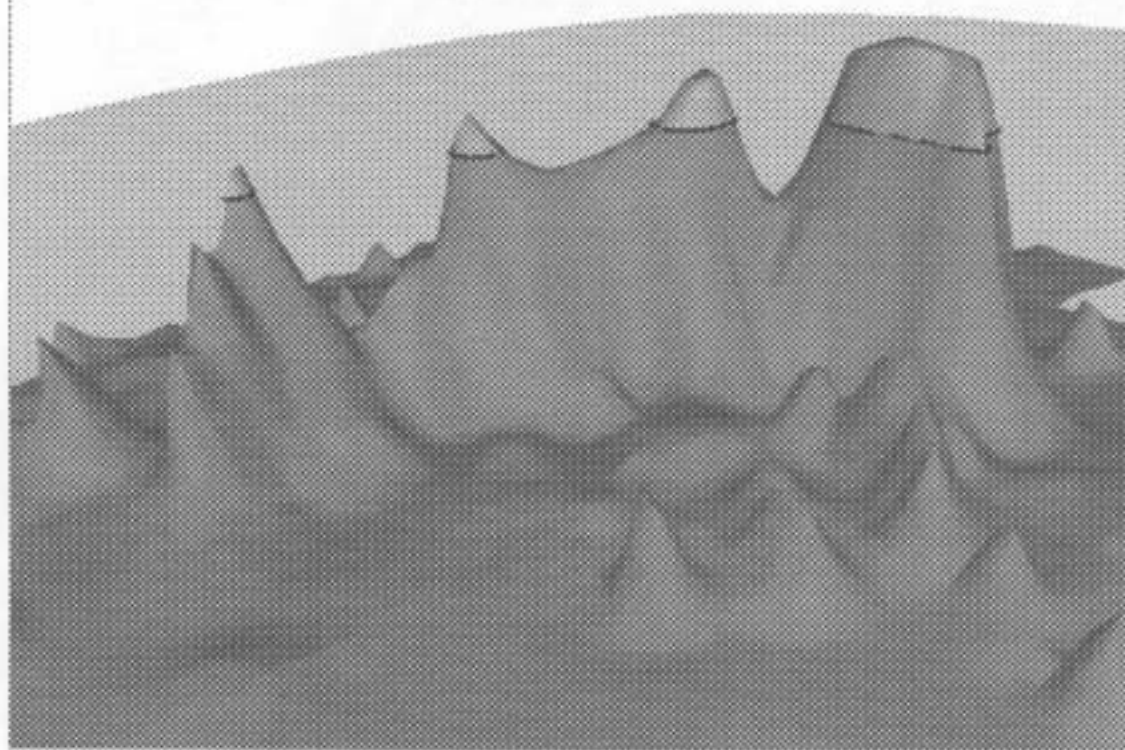
Figure 9. Coast lines obtained as intersection curves (max. error: 2.4651e−06; arc length: 3.79894; number of triangles: 15,192; time: 5.21 s).
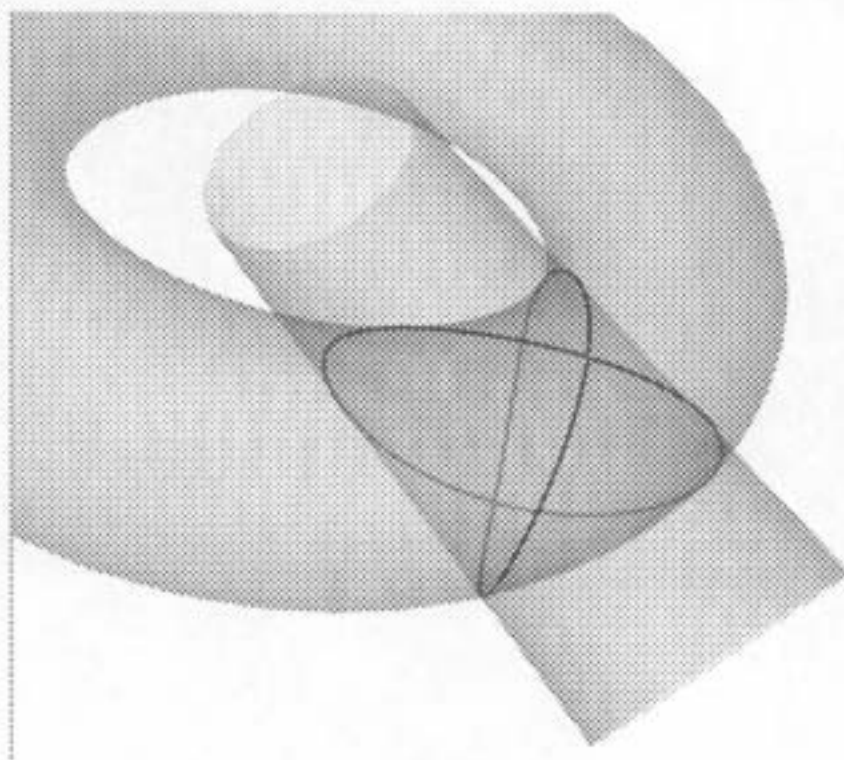

Figure 10. Cylinder−torus intersection (max. error: 4.20526e−07; arc length: 4.14077; number of triangles: 1,838; time: 0.43 s).

Often it is possible to reduce significantly the number of SSI points while still satisfying the required tolerance. Methods regarding data point reduction and decimation in the context of piecewise linear spline curves are discussed in Hamann and Chen (1994) and Ihm and Naylor (1991).

8. CONCLUSIONS

The SSI algorithm we have discussed is only one of many possible approaches to solving this difficult problem. The advantages of our algorithm are its speed and robustness and the ability to operate with a minimum of user input.

A few remarks regarding current shortcomings of the implementation, potential solutions, and future work seem appropriate. Self-intersecting surfaces are currently not treated in the implementation of the algorithm. However, one can apply the same basic algorithm to self-intersecting surfaces. When using cubic splines for the interpolation of the refined intersection points (lying exactly on the surfaces), an individual cubic spline segment might, in its interior, deviate from the surfaces too much. If this is the case, we propose to bisect cubic segments until a certain error condition is satisfied.

Surfaces touching each other tangentially, i.e., having the same tangent plane at a point of intersection, cause this algorithm to fail. This is explained by the fact that the SSI algorithm uses surface triangulations to determine the intersection points and the topology of the intersection curves. It will be a subject of future investigation to extend the algorithm such that the triangulations used for triangle–triangle intersection lead to results that accurately reflect the topology of the SSI curves. With the current implementation one has to detect and resolve such situations interactively. Figure 10 is an example of such a case. In this case, the algorithm produced four pairs of intersection curves (all with end points at the two singular points where all the curves meet).

REFERENCES

Anderson, J., Khamayseh, A. and Jean, B. (1997). *Adaptive Resolution Refinement*, Technical Report, Los Alamos National Laboratory.

Barnhill, R. E. (1992). *Geometry Processing for Design and Manufacturing.* SIAM, Philadelphia.

Barnhill, R. E. and Kersey, S. N. (1990). A marching method for surface/surface intersection. In: *Curves and Surfaces in CAGD '89*, R. E. Barnhill *et al.* (Eds), pp. 257–280. Elsevier, New York.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Comm. ACM* **18** (9), 509–517.

Brunnett, G. (1995). Geometric design with trimmed surfaces. In: *Geometric Modeling, Dagstuhl 1993*, H. Hagen *et al.* (Eds), Computing Supplement 10, pp. 101–115, Springer-Verlag, New York.

Farin, G. (1997). *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego.

George, P. L. (1991). *Automatic Mesh Generation.* Wiley, New York.

Hamann, B. and Chen, J. L. (1994). Data point selection for piecewise linear curve approximation. *Comput. Aided Geom. Design* **11** (3), 289–301.

Hamann, B. and Jean, B. A. (1996). Interactive surface correction based on a local approximation scheme. *Comput. Aided Geom. Design* **13** (4), 351–368.

Hamann, B. and Tsai, P.-Y. (1996). A tesselation algorithm for the representation of trimmed NURBS surfaces with arbitrary trimming curves. *Comput. Aided Geom. Design* **28** (6/7), 461–472.

Hosaka, M. (1992). *Modeling of Curves and Surfaces in CAD/CAM.* Springer, New York.

Hoschek, J. and Lasser, D. (1993). *Fundamentals of Computer Aided Geometric Design.* A. K. Peters, Ltd., Wellesley.

Hoschek, J. and Schneider, F. J. (1990). Spline conversion for trimmed rational Bézier and B-spline surfaces. *Comput. Aided Design* **22**, 580–590.

Houghton, E. G. and Emnett, R. F. (1985). Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Comput. Aided Geom. Design* **2**, 173–183.

Ihm, I. and Naylor, B. (1991). Piecewise linear approximations of digitized space curves with applications. In: *Scientific Visualization of Physical Phenomena*, N. M. Patrikalakis (Ed.), pp. 545–569, Springer-Verlag, New York.

Knupp, P. M. and Steinberg, S. (1993). *Fundamentals of Grid Generation.* CRC Press, Boca Raton.

Mortenson, M. E. (1985). *Geometric Modeling.* Wiley, New York.

Patrikalakis, N. M. (1993). Surface-to-surface intersections. *IEEE Computer Graphics and Applications* **13** (1), 89–95.

Piegl, L. A. and Richard, A. M. (1995). Tesselating trimmed NURBS surfaces. *Comput. Aided Design* **27** (1), 16–26.

Piegl, L. A. and Tiller, W. (1996). *The NURBS Book.* Springer, New York.

Rockwood, A. P., Heaton, K. and Davis, T. (1989). Real time rendering of trimmed surfaces. *ACM Trans. Graphics* **23**, 107–116.

Samet, H. (1990). *The Design and Analysis of Spatial Data Structures.* Addison Wesley, New York.

Thompson, J. F., Warsi, Z. U. A. and Mastin, C. W. (1985). *Numerical Grid Generation.* North-Holland, New York.

Vries-Baayens, A. E. and Seebregts, C. H. (1992). Exact conversion of a trimmed nonrational Bézier surface into composite or basic nonrational Bézier surfaces. In: *Topics in Surface Modeling.* H. Hagen (Ed.), pp. 115–143, SIAM, Philadelphia.

Weiler, K. J. (1986). *Topological Structures for Geometric Modeling.* PhD thesis. Rensselaer Polytechnic Institute, Troy.