Intersecting general parametric surfaces using bounding volumes

Ernst H. Huber
Institute of Computer Graphics
Vienna University of Technology
A-1040 Vienna, Karlsplatz 13/186/1, Austria
huber@eiunix.tuwien.ac.at

Abstract

This paper describes a robust algorithm for computing all parts of the intersection curve of two general parametric surfaces. The algorithm follows a divide-and-conquer approach. Surfaces (and parts of them) are enclosed by tight parallel epipeds and axis aligned bounding boxes. If two bounding volumes intersect, one surface is split into two patches. This step is repeated recursively for both patches and the other surface until a predefined termination condition is satisfied. The result is an approximation to the intersection curve in the parameter domains of both surfaces and in object space. Interval arithmetic is used for the computation of bounding volumes and to guarantee correctness of the results.

Keywords: *general surface intersection, interval arithmetic, bounding volumes, parallel epipeds*

1 Introduction

Surface-to-surface intersection (SSI) is a basic problem in computer aided geometric design (CAGD). If we want to perform calculations on a geometric object we first have to create a geometric model of it. Several approaches exist for geometric modelling, the SSI-problem arises if an object is described by its bounding surfaces (boundary representation) or if an object is represented by its volume (constructive solid geometry, CSG). In boundary representation an edge of an object can be seen as the intersection curve of two bounding surfaces. In CSG solids are combined by boolean operations (union, intersection, difference) resulting in intersection volumes with surfaces and intersection curves on the edges. The SSI-problem has also to be solved if a CSG model is transformed to boundary representation.

For describing surfaces implicit and parametric representations are common in CAGD. If two surfaces intersect, the result will be either a set of isolated points, a set of curves, a set of overlapping surfaces or any combination of these three cases [2].

Previous work: The SSI-problem has been studied detailed in a variety of publications. Apart from some special (surface dependent) methods which provide exact solutions, two general approaches are common. Both are approximation methods, each approach can be qualified according to the attributes efficiency, generality and robustness.

Marching methods start from a known common point of both surfaces and march along the intersection curve by successively computing new points and connecting them by line segments. The problem of this approach is its lack of robustness: even simple surfaces possibly intersect in complex curves with several branches, closed loops and singular points. Such cases are difficult to detect and are a major problem for SSI algorithms.

Subdivision methods divide both surfaces until each patch can be replaced by a simple approximation. For each pair of these approximations (one piece of each surface) the intersection problem can be solved easily. This approach has some weaknesses: it leads to a huge number of intersection-operations and, due to the approximation of the patches, some parts of the solution might be lost. The amount of intersection operations can be reduced if the surfaces are enclosed by bounding volumes and are subdivided recursively. Parts of a surface which do not intersect with the other surface can be detected early by testing whether the according bounding volumes do not intersect and need not be considered in the further calculation [6].

A robust subdivision algorithm: In this paper a robust divide-and-conquer algorithm for the computation of the intersection curve of two general parametric surfaces is presented. Surfaces are defined by three coordinate functions mapping a rectangular parameter domain to object space:

$$S(u,v) = \left\{ \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} \middle| \begin{array}{c} u \in U = [\underline{u}, \overline{u}] \\ v \in V = [\underline{v}, \overline{v}] \end{array} \right\}, \quad (1)$$

where U, V are intervals in $\mathbb{R}$. We only assume that the coordinate functions are continuous in the whole domain and that their partial derivatives exist and are continuous, too.

The weaknesses of previous subdivision methods are overcome by using tight bounding volumes instead of simple approximations for the patches. For each bounding volume it is guaranteed that it encloses the according patch. If there is an intersection between two bounding volumes, we split one surface into two patches (by bisecting the parameter domain parallel to the u- and v- axes) and test both patches against the other surface. This procedure is repeated recursively (the surfaces are divided alternately), stopping if a predefined termination condition is satisfied.

```
(* Simplest divide-and-conquer SSI algorithm *)
procedure intersect (Surface F_i, G_j)
  if boundingVolumesIntersect(F_i, G_j) then
    if termCondition(F_i, G_j) then
      writeResult
    else
      splitSurface(F_i, F_{2i+1}, F_{2i+2})
      intersect(G_j, F_{2i+1})
      intersect(G_j, F_{2i+2})
```

The result is either a collection of intersecting bounding volumes in object space or a collection of rectangular pieces in both parameter domains, enclosing all parts of the intersection curve.

2 Interval arithmetic and automatic differentiation

In this work we need enclosing intervals for function values and derivatives of parameter functions. We consider real valued continuous functions depending on one or two independent variables.

Interval arithmetic: If we evaluate the formula of a function, $f(t)$ with parameter domain $t \in T = [\underline{t}, \overline{t}]$, $\underline{t} \leq \overline{t}$, $\underline{t}, \overline{t} \in \mathbb{R}$, for one $t \in T$ with real arithmetic, we obtain a real number as function value for a single point in parameter domain.

Evaluating the formula of $f(t)$ for all $t \in T$ provides an interval $V = [\underline{v}, \overline{v}]$ enclosing all function values, so that $f(t) \in V \; \forall \; t \in T$. An arithmetic operation for two intervals $A = [\underline{a}, \overline{a}], B = [\underline{b}, \overline{b}]$ is defined by

$$A \circ B \quad = \quad \{a \circ b \mid a \in A, b \in B\}$$

Interval arithmetic [1], [8], [9] generates upper and lower bounds for the set of function values that belong to all parameter values contained in given intervals. Therefore upper and lower bounds of the input intervals A and B are used, for the basic arithmetic operations we have the definitions:

$$\begin{aligned}
A + B &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\
A - B &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \\
A * B &= [\min(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b}), \\
&\quad\ \max(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b})] \\
A \;/\; B &= [\underline{a}, \overline{a}] * [1/\overline{b}, 1/\underline{b}] \text{ if } 0 \notin B
\end{aligned}$$

(division by an interval containing zero has to be avoided since the result is no longer a finite interval). Similar definitions can be given for basic functions like $\ln, \sin, \cos, \ldots$, but computation may be more complicated if the function is not monotonic.

To guarantee the inclusion of an exact value by an interval, interval arithmetic packages have to round the bounds of intervals to the next upper and lower machine numbers. For basic interval arithmetic we use the BIAS-library [7].

Automatic differentiation: Automatic differentiation allows us to compute the value of the derivative $f_t(t)$ while computing the value of the function $f(t)$ for a $t \in T$. We define an arithmetic for pairs (f, f_t), starting the evaluation with the substitution of all

$$\begin{array}{ccccl}
t & \rightarrow & (t, 1) & \ldots & \text{independent variable,} \\
u & \rightarrow & (u, 0) & \ldots & \text{any other variable,} \\
c & \rightarrow & (c, 0) & \ldots & \text{a constant,}
\end{array}$$

in f and f_t and with definitions for arithmetic operations similar to interval arithmetic:

$$\begin{aligned}
(a, a_t) + (b, b_t) &= (a + b, a_t + b_t) \\
(a, a_t) - (b, b_t) &= (a - b, a_t - b_t) \\
(a, a_t) * (b, b_t) &= (a * b, a_t * b + b_t * a) \\
(a, a_t) \;/\; (b, b_t) &= (a/b, a_t/b - a * b_t/b^2)
\end{aligned}$$

For functions we define

$$f(a, a_t) \quad = \quad (f(a), f_t(a) * a_t).$$

This concept can be extended for functions with two independent variables and for their partial derivatives.

A combination of both techniques can be achieved simply by replacing real arithmetic by interval arithmetic during the evaluation process of automatic differentiation.

3 Bounding volumes

Our algorithm relies on two major operations: the computation of bounding volumes and the test whether two bounding volumes intersect.

3.1 Computing bounding volumes

Let us first consider the computation of bounding volumes. Values for coordinate functions for a given pair of parameter values (u, v) are computed by an evaluator module which also computes values for the partial derivatives.

Using interval arithmetic and evaluating the formulas of the coordinate functions substituting u by U and v by V returns an axis aligned bounding box (AABB) of a surface and intervals for the derivatives. AABBs generally overestimate the enclosed surface, thus leading to unnecessary subdivisions and intersection tests [4].

In order to take the surface shape and orientation into account the additional usage of parallel epipeds is suggested. From the intervals of the partial derivatives we compute a hull enclosing the surface [3]. Each parallel epiped which encloses this hull encloses also the surface.

We start with a two-dimensional curve

$$C(t) = \left\{ \left(\begin{array}{c} x(t) \\ y(t) \end{array} \right) \middle| \; t \in T = [\underline{t}, \overline{t}] \right\}, \qquad (2)$$

where T is an interval in $\mathbb{R}$, and apply the mean value theorem of differential calculus, e.g. for $x(t)$:

For all $t, t_0 \in T$ exists a $\tau \in [t, t_0] \subseteq T$, so that

$$x(t) = x(t_0) + (t - t_0)x_t(\tau), \qquad (3)$$

$x_t(t)$ the derivative from $x(t)$ with respect to t (analogous for $y(t)$).

Evaluating the formulas for x_t and y_t with T as independent variable with interval arithmetic provides inclusion intervals ($X_t = [\underline{x}_t, \overline{x}_t], Y_t = [\underline{y}_t, \overline{y}_t]$) for the derivatives. Substituting $\overline{t}$ by t, $x_t(\tau)$ by X_t and $y_t(\tau)$ by Y_t yields

$$C(\overline{t}) \in R(\overline{t}) = \left(\begin{array}{c} x(t_0) \\ y(t_0) \end{array} \right) + (\overline{t} - t_0) \left(\begin{array}{c} X_t \\ Y_t \end{array} \right) \qquad (4)$$

Evaluating equation 4 with interval arithmetic we get a rectangle $R(\overline{t})$ enclosing the end point $C(\overline{t})$ of the curve $C(t)$. Doing this for all $t \in [t_0, \overline{t}]$ we receive an angular inclusion for $C(t), t \in [t_0, \overline{t}]$ starting at $C(t_0)$. This construction may be extended in the other direction for $t \in [\underline{t}, t_0]$. $C(t)$ is now included by the two outermost rectangles $R(\overline{t})$, $R(\underline{t})$ and the angular regions connecting them (figure 1).

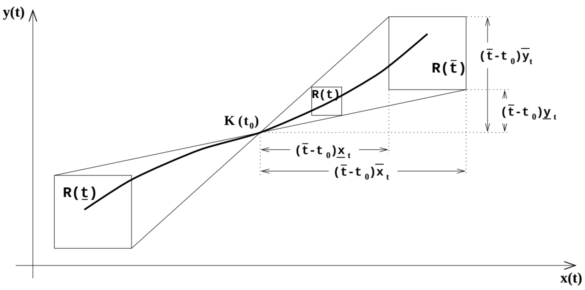

Figure 1: Constructing a bounding area

The considerations mentioned above can be rephrased in the following corollary:

Corollary 1 *Each convex area, e.g. each parallelogram enclosing the two outermost rectangles $R(\underline{t})$, $R(\overline{t})$ is a bounding area for the curve $C(t)$ from equation 2.*

Extending this proposition for space curves we receive boxes instead of rectangles and pyramidal volumes instead of angular regions (figure 2 left shows two of them, one for a u-line and one for a v-line). For a surface $S(u, v)$ we use the mean value theorem for two independent variables

$$x(u, v) = x(u_0, v_0) + (u - u_0)x_u(\tau_u) + (v - v_0)x_v(\tau_v) \qquad (5)$$

and apply it according to equation 3 and 4 (analogous for $y(u, v)$ and $z(u, v)$). We obtain four boxes at the end points of the u, v-lines (which divide the surface into four parts) and four boxes for the corner points of the surface (figure 2 right shows two boxes for u- and v-lines, one corner box and the according angular regions).

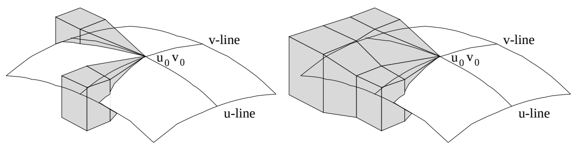

Figure 2: Constructing a bounding volume

The extended corollary now is

Corollary 2 *Each convex volume, e.g. each parallel epiped enclosing the four corner boxes $B(\underline{u}, \underline{v})$, $B(\underline{u}, \overline{v})$, $B(\overline{u}, \underline{v})$, $B(\overline{u}, \overline{v})$ is a bounding volume for the surface $S(u, v)$ from equation 1.*

For a bounding parallel epiped $E = A + c_1\vec{v}_1 + c_2\vec{v}_2 + c_3\vec{v}_3$, $c_i \in [0, 1]$ we have to find the orientation and position of the $2 * 3$ planes containing the faces of the epiped and calculate from this representation vertex A and the $\vec{v}_i$.

The orientation of the top and bottom plane can be defined by average vectors in u and v direction computed from the four corner points of the surface:

$$\begin{aligned} \vec{u}_D &= \frac{S(\overline{u}, \underline{v}) - S(\underline{u}, \underline{v}) + S(\overline{u}, \overline{v}) - S(\underline{u}, \overline{v})}{2(\overline{u} - \underline{u})} \\ \vec{v}_D &= \frac{S(\underline{u}, \overline{v}) - S(\underline{u}, \underline{v}) + S(\overline{u}, \overline{v}) - S(\overline{u}, \underline{v})}{2(\overline{v} - \underline{v})} \end{aligned} \qquad (6)$$

Assuming that epipeds are used mainly for flat surfaces, we may choose the third direction orthogonal to $\vec{u}_D$ and $\vec{v}_D$. As a result we get three normal vectors describing the orientation of the planes:

$$\begin{aligned} \vec{n}_1 &= \vec{u}_D \times \vec{v}_D \\ \vec{n}_2 &= \vec{n}_1 \times \vec{v}_D \\ \vec{n}_3 &= \vec{n}_1 \times \vec{u}_D \end{aligned}$$

The positions of these planes are determined by moving each plane along its normal vector until all four corner boxes of the above specified hull lie inside the parallel epiped. The axis aligned bounding box of a surface is considered additionally to restrict the parallel epiped.

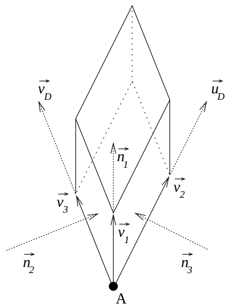

Figure 3: Vectors defining an epiped

3.2 Intersection Test

The second major operation is the test whether two bounding volumes intersect. This operation is trivial for AABBs, but expensive for parallel epipeds. As parallel epipeds are convex polyhedra we are able to apply the separating axis theorem [5]:

Theorem 1 *Two convex polyhedra do not intersect if there exists a separating plane which is either parallel to a face of one polyhedron or which is parallel to at least one edge of each polyhedron.*

An axis normal to a separating plane is called a *separating axis*. Projecting both polyhedra on a separating axis (by an orthogonal projection) we obtain an interval for each polyhedron. Exists at least one separating axis where the projected polyhedra (the intervals) do not intersect, then the polyhedra do not intersect (otherwise the polyhedra intersect).

For epipeds we have to consider at most 15 separating axes: 3 (already known) normals to the faces of each epiped and $3 * 3$ normals to planes defined by pairwise combining the edge-directions of both epipeds. From each epiped we only project the starting point A and the defining vectors $\vec{v}_i$ to a separating axis and build the projection from this points.

To take advantage of both bounding volume types, we suggest the combined use of them. This hybrid algorithm first tests whether two AABBs intersect. If this evaluates to true, then in a second stage the parallel epipeds are calculated and tested. The recursion is only entered if the parallel epipeds intersect, too.

4 Termination condition

For terminating the recursion several conditions are used: in parameter space simply the size of the parameter domain of a patch relative to the size of the parameter domain of the original surface is taken. Alternately in object space the volume of the intersection of two bounding volumes can be analysed. Additional restrictions can be made considering the linearity or flatness of the resulting curve or patch.

5 Experiences

Implementation: We have incorporated the methods of interval arithmetic and automatic differentiation into an evaluator module. This module works as interpretative system for functions defining surfaces. The whole system was implemented in C++.

Results: Using interval arithmetic instead of floating point arithmetic for all critical operations guarantees that the intersection curve lies inside the resulting bounding volumes (in object space) or rectangles (in both parameter domains). The usage of bounding volumes guarantees that all kinds of intersections are detected.

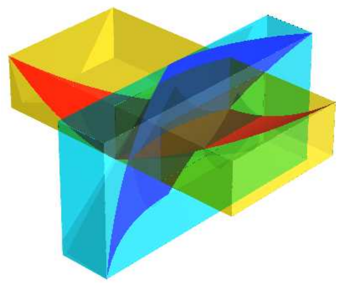

Figure 4: Surfaces enclosed by AABBs

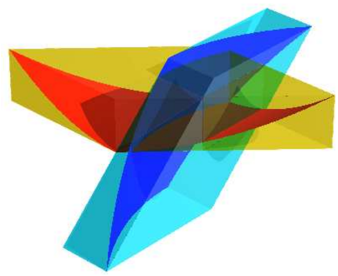

Figure 5: Surfaces enclosed by tight PEs

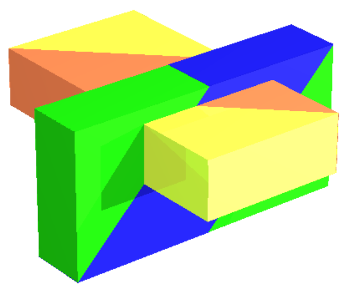

Figure 6: AABBs versus PEs

The figures above show two intersecting surfaces enclosed by (transparent) bounding volumes (AABBs in figure 4 and tight parallel epipeds (PEs) in figure 5). Both bounding volumes are compared in figure 6.
Three versions of this algorithm were implemented, one algorithm using AABBs only, one pure parallel epiped algorithm and the above stated hybrid algorithm. We demonstrate these algorithms by means of intersecting surfaces

$$S_1(u_1, v_1) = \begin{pmatrix} \cos u_1 * \sin v_1 * (1 - \cos v_1)/2 \\ \sin u_1 * \sin v_1 * (1 - \cos v_1)/2 \\ \cos v_1 \end{pmatrix},$$

("drop") with parameter domain $u_1 \in [-\pi, \pi]$, $v_1 \in [0, \pi]$, and

$$S_2(u_2, v_2) = \begin{pmatrix} v_2 \\ 0.4 * \sin(2\pi u_2) \\ 0.4 * \cos(2\pi u_2) \end{pmatrix},$$

("cylinder") with parameter domain $u_2 \in [0, 1]$, $v_2 \in [-1, 1]$ (figure 7).

Resulting intersection curves are shown in figure 8 to 11. Figure 8 shows the intersection curve in object space on both (transparent) surfaces. Figure 9 to 11 show the bisection of the parameter domains for both surfaces (S_1 left, S_2 right) returned by the AABB, parallel epiped and hybrid algorithm.

For termination the simple condition $A_{D_i} \leq \frac{A_{D_0}}{10000}$ was used, where A_{D_i} is the area of the parameter domain of patch i, and where A_{D_0} is the area of the parameter domain of the original surface.

For qualifying efficiency, the number of iterations (surface splits) to solve a problem with a predefined precision and relative consumed time were considered.

	splits S_1	splits S_2	rel. time
AABB	7317	8264	1.00
Epiped	3475	3734	0.93
Hybrid	2947	3264	0.72

Table 1: Algorithm comparison

In this example the hybrid algorithm performs best, while the AABB algorithm shows worst behaviour for both qualification criteria (table 1).

For exhaustive testing we setup a test-case database with over 50 surface-types and more than 1000 test-cases. Generally it can be said that tight bounding volumes lead to less iterations, but are more expensive to compute than simple ones, so there is a trade-off between simplicity and tightness. The combination of both bounding volumes leads in many cases to a better time behaviour than the usage of just one of them.

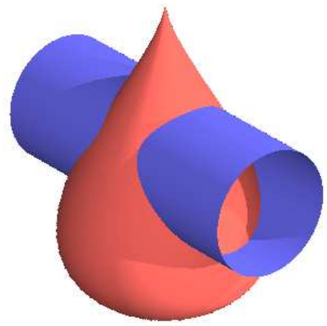

Figure 7: Intersecting surfaces

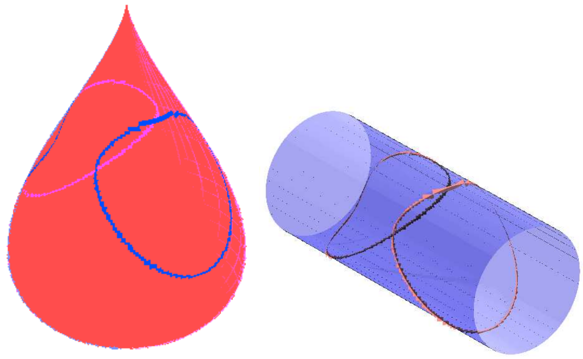

Figure 8: 3D intersection curve

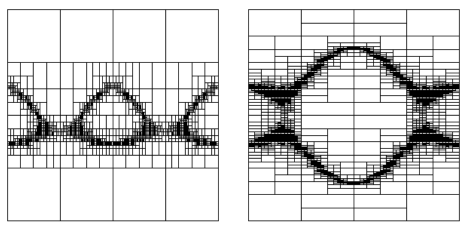

Figure 9: 2D intersection curve, AABB alg.

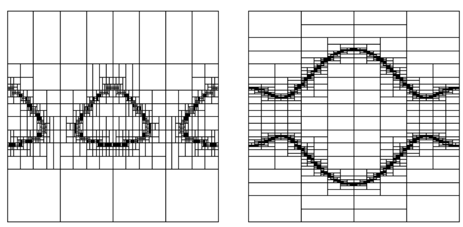

Figure 10: 2D intersection curve, PE alg.

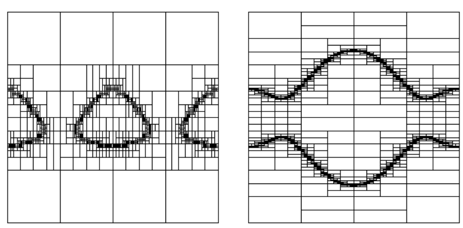

Figure 11: 2D intersection curve, hybrid alg.

Future work: It is thought that the subdivision process can be accelerated by an early detection of parts of a patch which definitely do not intersect with the second patch. Parts of the parameter domain where one patch cannot reach the enclosure of the other patch can be determined from the intervals of the partial derivatives. By truncating such parts it should be possible to reduce the parameter domains substantially. Further investigations will be made for termination conditions and in sorting the resulting hulls.

References

[1] G. Alefeld and J. Herzberger. *Introduction to interval computations.* Academic Press, Inc., 1983. ISBN 0-12-049820-0.

[2] R. Barnhill, G. Farin, M. Jordan, and B. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4:3–16, 1987.

[3] W. Barth, R. Lieger, and M. Schindler. Ray tracing general parametric surfaces using interval arithmetic. *The Visual Computer*, 10(7):363–371, 1994.

[4] M. Gleicher and M. Kass. An interval refinement technique for surface intersection. In *Graphics Interface '92*, pages 242–249, 1992.

[5] S. Gottschalk. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[6] E.G. Houghton, R.F. Emnett, J.D. Factor, and Ch.L. Sabharwal. Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Computer Aided Geometric Design*, 2:173–183, 1985.

[7] O. Knüppel. Bias - basic interval arithmetic subroutines. Bericht des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TU Hamburg-Harburg, July 1993.

[8] S.P. Mudur and P.A. Koparkar. Interval methods for processing geometric objects. *IEEE Computer Graphics & Applications*, 4(2):7–17, February 1984.

[9] K.G. Suffern and E.D. Fackerell. Interval methods in computer graphics. *Computer & Graphics*, 15(3):331–340, 1991.