Piecewise-circular curves for geometric modeling

by Jaroslaw R. Rossignac
Aristides A. G. Requicha

Modern solid modelers must be able to represent a wide class of objects, and must support Boolean operations on solids. These operations are very useful for defining solids, detecting interferences, and modeling fabrication processes. Computing the boundaries of solids defined through Boolean operations requires algorithms for surface/surface and curve/surface intersection. Many of the currently available modelers use closed-form parametric expressions for the curves of intersection of quadric surfaces, and compute intersections of these curves with other surfaces by finding the roots of low-degree polynomials. Because the curves that result from intersections involving tori or more complex surfaces generally cannot be expressed in closed form, modelers typically approximate these curves by cubic splines that interpolate points lying on the true intersections. Cubic splines exhibit second-degree continuity, but they are expensive to process in solid modeling computations. In this paper, we trade second-degree continuity for computational simplicity, and present a method for interpolating three-dimensional points and associated unit tangent vectors by smooth space curves composed of straight line segments and circular arcs. These curves are designated as PCCs (for piecewise-circular curves) and have continuous unit tangents. PCCs can be used in efficient algorithms for performing fundamental geometric computations, such as the evaluation of the minimal distance from a point to a curve or the intersection of a curve and a surface. Formulae and algorithms are presented for generating and processing PCCs in solid modelers. We also show that PCCs are useful for incorporating toroidal primitives, as well as sweeping, growing, shrinking, and blending operations in systems that model solids bounded by the natural quadric surfaces—planes, cylinders, cones, and spheres.

Introduction

Solid modeling plays a key role in computer-aided design and manufacturing of mechanical and electromechanical parts and assemblies. It also is becoming increasingly important in computer graphics, computer vision, robotics, and other disciplines that involve spatial phenomena. A modern solid modeler must be able to represent a wide class of geometric objects, and must also support Boolean operations on solids. Boolean operations—regularized set union, difference, and intersection [1]—are very useful for defining solids via CSG (Constructive Solid Geometry), for detecting spatial interferences, and for modeling physical processes such as machining and integrated circuit fabrication [2]. The study reported in this paper is aimed at increasing the geometric coverage of modelers that support Boolean operations.

To compute the bounding edges and vertices of a solid defined by Boolean operations one must (1) find the

potential edges, which are the curves of intersection of the object's surfaces, and (2) *classify* these curves to determine the segments that are inside, outside, or on the boundary of the object [3, 4]. These curve segments are bounded by vertices, which are points of intersection of the potential edges with the object's surfaces. Thus, support for Boolean operations requires the computation of surface/surface and curve/surface intersections. Typically, thousands of such calculations are needed to evaluate the boundary of a solid of moderate complexity.

Which objects should be representable in a modeler? Mechanical parts may be classified in two broad groups: 1) Sculptured or free-form objects, e.g., car bodies, characterized by doubly curved bounding surfaces, and 2) unsculptured or functional objects, e.g., machine-tool components. Part surveys show that a large proportion of parts are unsculptured, and that over 90% of these are bounded by planes, cylinders, spheres, cones, tori, and blends (i.e., fillets and rounds) between such surfaces. Planar, cylindrical, spherical, and conical surfaces are often called the *natural quadrics* [5], because they are produced easily by the usual machining operations. Tori have similar properties, and we refer to the natural quadrics plus the torus as *natural surfaces*.

Modelers for sculptured objects are emerging [6], but known techniques for computing the required curve and surface intersections are inefficient, numerically unreliable, or both. This study is focused on the more restricted domain of solids bounded by natural surfaces and blends. Current solid modelers use either one of the two following techniques to support Boolean operations on objects bounded by natural surfaces:

1. All surfaces are approximated by planar facets, and these are used for all intersection calculations. Polyhedral approximations with reasonable numbers of facets are excellent for displaying objects, but not sufficiently accurate for numerically controlled machining and other applications.
2. Curves of intersection of natural quadrics are expressed parametrically in closed form. The parametric equations $x = x(t)$, $y = y(t)$, $z = z(t)$ of a curve are substituted in the implicit equation $F(x, y, z) = 0$ of a surface, and the resulting equation solved for t. This technique may provide exact answers, within the accuracy of floating-point arithmetic, but is not applicable to toroidal or blending surfaces because closed-form expressions for the corresponding intersection curves are unavailable. The curves of intersection of tori with tori or with other surfaces are approximated with cubic splines that interpolate a sequence of points lying on the exact edge. (These points may be computed by several techniques.) The cubic approximations are used to compute curve/surface intersections. For cubic/torus intersection this amounts to solving a twelfth-degree algebraic equation, which must be done numerically. High-degree equations are undesirable because the efficiency and reliability of numerical root finders generally decrease when the degrees of the equations to be solved increase.

In this paper we present an alternative scheme for approximating intersections that involve toroidal or blending surfaces by using PCCs (piecewise-circular curves).

Curve approximation by smoothly joined rational or polynomial spans has been studied since the early times of computer-aided design [7, 8], and many approximation and interpolation schemes have been proposed. Bezier and B-spline (rational or integral) cubic parametric curves are the most widely used in modern systems for the interactive design of free-form curves, in both two and three dimensions [7].

These schemes exhibit several features important to free-form curve design:

1. Parametric formulation—a curve is composed of segments or spans defined by parametric vector-valued functions that are either polynomials or quotients of polynomials of low degree.
2. Second-degree continuity—the spans are joined with second-degree parametric continuity; this is not equivalent to continuity of geometric curvature [9] but produces smooth curves in most common cases.
3. Local control—a curve can be modified locally by adjusting parameters or moving control points, without producing side effects on the rest of the curve.

In our opinion, only the first of these features is important when the curves are used to approximate surface intersections in solid modeling. Therefore we trade second-degree continuity (which is unimportant in many geometric modeling applications) for computational simplicity of curve/surface intersection calculation, and we use approximations based on PCCs rather than cubic splines. PCCs are composed of line and arc segments and exhibit first-degree geometric continuity (G^1), i.e., they have continuous unit tangent directions [9]. The intersection of a PCC with any of the natural surfaces, including tori, amounts to solving a fourth-degree equation, and this can be done analytically. Using PCCs instead of piecewise-cubic curves to approximate the edges of the modeled solids does not change the computational complexity of the algorithms that are fundamental to solid modeling, because the number of spans is the same in both cases—our experience indicates that cubic spans are not better for approximating space curves than the bi-arc spans that form our PCCs. Furthermore, various speedups based on the convex-hull properties of piecewise-cubic curves may also be used with PCCs, because, as explained below, each arc of a PCC is

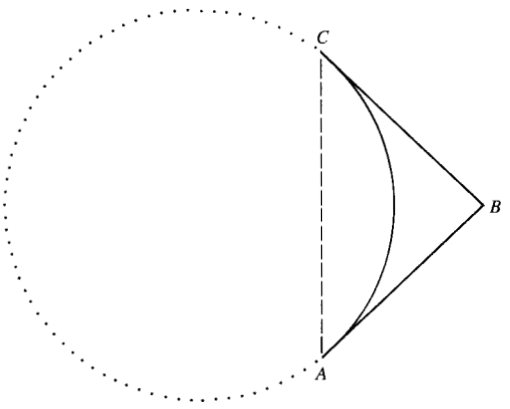

Figure 1

Control triangle: The circular arc is completely defined by its control triangle (A, B, C).

contained in a very simple convex hull (a triangle), and can be efficiently subdivided.

In addition to supporting tori in modelers based on natural quadrics, a major goal of the work described here was to use natural surfaces to support blending and offsetting operations [10]. Most of the blending surfaces found in mechanical parts are constant-radius blends, which are conceptually generated by rolling a sphere tangentially to the surfaces being blended. PCCs are well suited for supporting constant-radius blending for the following reasons.

Constant-radius blends can be produced by combining growing and shrinking operations (collectively called offsetting) with Boolean operations [11]. Offsetting a solid bounded by natural surfaces produces another solid bounded by natural surfaces and by so-called canal surfaces [12]. A canal surface is the envelope of a family of spherical surfaces generated by sweeping a sphere along a trajectory called a spine. Closed-form implicit equations for general canal surfaces are unknown, and exact techniques for dealing with such surfaces in a modeler also are unknown. Therefore canal surfaces must be approximated—preferably by piecewise-simple smooth surfaces. In our approach spines of canal surfaces are approximated by PCCs, which implies that the canal surfaces are approximated by smoothly joined piecewise-toroidal or -cylindrical surfaces. This approximation scheme is very attractive. Since canal surfaces are approximated by natural surfaces, new intersection algorithms are not required. Thus PCC approximation enables a natural surface modeler to deal with constant-radius blends, and with Boolean operations on blended objects.

To test the approach described in this paper, the authors implemented an experimental solid modeling system [10]. The use of PCC approximations for all intersection curves greatly simplified the implementation because only one type of edge needed to be considered for curve/surface intersection calculations.

The remainder of the paper is divided into five major sections: construction of PCCs, given point and tangent data; approximation of intersection curves between two natural surfaces; various utilities for supporting calculations that involve PCCs; other applications of PCCs; and experimental implementation and conclusions.

Construction and representation of PCCs

The interpolation of an ordered set of points and of associated unit tangents by a smooth curve can be broken down into a sequence of simpler independent subproblems, each of which consists of interpolating a pair of consecutive points and tangents with a smooth span. It is assumed throughout this paper that the term "smooth" is equivalent to "G^1-continuous." We first show how interpolants over a single span can be constructed, and then discuss storage costs for a whole PCC.

• *Local interpolation*

We seek a computationally efficient solution for the Hermite interpolation problem stated as follows. Given two points P_1 and P_2 and associated unit tangent vectors T_1 and T_2, generate a 3D curve segment that is G^1-continuous and tangent to T_1 at P_1 and to T_2 at P_2. As noted earlier, some geometric modeling systems solve this problem by using cubic curves. Our approach uses *twisted bi-arcs*, which are curve segments composed of two circular or linear segments. From now on, we make no distinction between circular and linear segments and we use the common term *arc* for both. (A linear segment may be viewed as an arc of a circle whose center lies at infinity, and implementational problems resulting from such generalizations are minor and best addressed directly in the lowest-level routines.) Each arc in a twisted bi-arc interpolates the boundary conditions at one of the given endpoints, and the two arcs are joined together smoothly. Curve classification [3] and other geometric calculations used in modeling systems are much simpler and less expensive for circular arcs than for cubics or for noncircular conics.

Two related approaches have been reported in the literature. Varady [13], in a parallel and independent effort, has developed *parabolic* bi-arcs, and Sabin [14] has used *planar* circular bi-arcs.

Control triangle for a circular arc

A circular arc with center O and radius r, spanning an angle of less than 180 degrees, can be described by an isosceles control triangle (A, B, C) as shown in **Figure 1**. A and C are

the endpoints of the arc, which is tangent to AB and BC at these endpoints. (In fact, the same isosceles control triangle may be used to represent two complementary arcs of the same circle.) The three points of a control triangle in space have nine coordinates, but these are linked by the constraint $\|A - B\| = \|B - C\|$. Therefore, a circular arc in space can be defined by eight parameters.

If we constrain one end of the arc in position ($A = P_1$) and in tangent direction (AB parallel to T_1), the point B must lie on the line that passes through P_1 and is tangent to T_1. The arc has still three degrees of freedom. We can specify the signed distance a defined by $B = P_1 + aT_1$ to fix one of the degrees of freedom. The two remaining degrees can be chosen as the angle between AB and CB and an angle that defines the rotation of CB around AB. In order to parameterize the arc in a manner consistent with the orientation of T_1, we adopt the following convention: If $a > 0$, the triangle (A, B, C) defines the circular arc inscribed in the triangle; if $a < 0$, the triangle defines the complement of that arc in the circle (see **Figure 2**).

The control triangle of a semicircle is degenerate. It consists of a pair of parallel semilines, which meet at a point B at infinity. (One can represent points at infinity with homogeneous coordinates, but it is simpler to treat semicircles as special cases. This is a low-level implementation issue, not addressed here.)

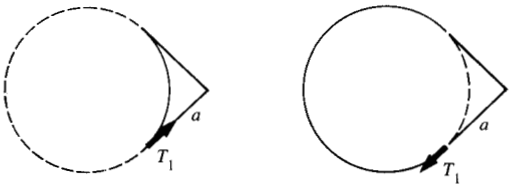

Figure 2

Compatible orientation: The portion of the circle defined by the control triangle (A, B, C) depends on the orientation of T_1 and on the sign of a.

Control polygon for a twisted bi-arc

To ensure G^1 continuity of the PCC, we force pairs of arcs to share a common tangent direction and orientation at their common end. Since an arc is tangent to its control triangle, we can smoothly join two arcs by connecting their control triangles in a smooth manner, as shown in **Figure 3**.

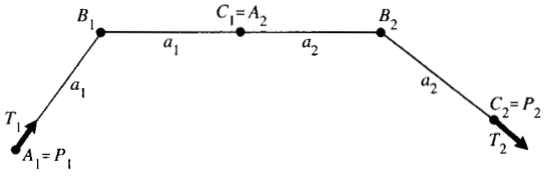

Figure 3

Interpolating control polygon: The endpoints P_1, P_2 and the tangents T_1, T_2 are interpolated by a control polygon (A_1, B_1, B_2, C_2).

The resulting figure is a (not necessarily planar) quadrilateral called the *control polygon* of the bi-arc. Notice that when the four vertices of the control polygon are not coplanar, the bi-arc is twisted; i.e., it is a smooth nonplanar curve.

The control polygon of the bi-arc made of the arcs described by the triangles (A_1, B_1, C_1) and (A_2, B_2, C_2) is (A_1, B_1, B_2, C_2), because $C_1 = A_2$ and A_2 lies on the segment (B_1, B_2).

Constraints

The eighteen coordinates of the six points A_1, B_1, C_1, A_2, B_2, and C_2 that define a control polygon of an interpolating bi-arc must satisfy the following constraints:

1. The polygon satisfies the position boundary conditions (six equations):

   $A_1 = P_1$ and $C_2 = P_2$.

2. The polygon can be decomposed into two isosceles triangles (two equations):

   $\|B_1 - A_1\| = \|C_1 - B_1\|$ and $\|B_2 - A_2\| = \|C_2 - B_2\|$.

3. The two control triangles are smoothly connected at their common point (six more equations, but also an extra variable k):

   $C_1 = A_2$ and $C_1 - B_1 = k(B_2 - C_1)$.

4. The polygon satisfies the tangential boundary conditions (six equations and two new variables a_1 and a_2):

   $B_1 = P_1 + a_1 T_1$ and $B_2 = P_2 - a_2 T_2$.

Thus we have 20 equations and 21 variables, and therefore one degree of freedom. If we fix a_1 and a_2, the whole system of equations can be simplified by constructing the control polygon as follows:

$A_1 = P_1$,

$C_2 = P_2$,

$B_1 = P_1 + a_1 T_1$,

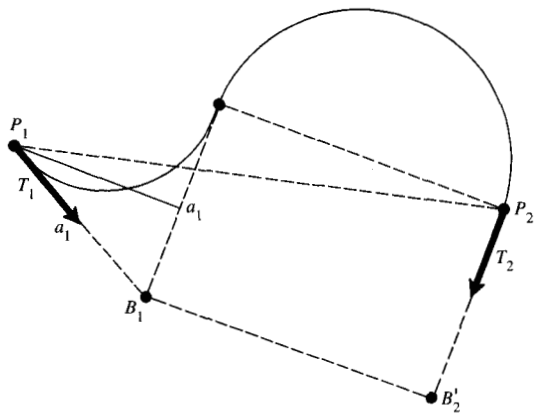

Figure 4

Bi-arc with one semicircle: In the special case where $a_1(T_1 \cdot T_2 - 1) = S \cdot T_2$, the point B_2 is at infinity and hence is not depicted in the figure. The second triangle defines a semicircle.

$$B_2 = P_2 - a_2 T_2,$$

$$C_1 = B_1 + \frac{a_1}{a_1 + a_2}(B_2 - B_1),$$

$$A_2 = C_1.$$

Two unknowns remain: a_1 and a_2. They are bound by a single equation,

$$\| B_2 - B_1 \| = a_1 + a_2,$$

which becomes

$$\| (P_2 - a_2 T_2) - (P_1 + a_1 T_1) \| = a_1 + a_2, \quad (1)$$

• *Computing the control polygon*

We use a_1 as the parameter that determines the remaining degree of freedom for the bi-arc, and express a_2 in terms of a_1.

We can solve Equation (1) for a_2 by first squaring both terms:

$$\| P_2 - P_1 - (a_2 T_2 + a_1 T_1) \|^2 = (a_1 + a_2)^2, \quad (2)$$

which, with $S = P_2 - P_1$, yields

$$\| S \|^2 - 2S \cdot (a_1 T_1 + a_2 T_2) + a_1^2 \| T_1 \|^2 + a_2^2 \| T_2 \|^2 + 2a_1 a_2 T_1 \cdot T_2 = a_1^2 + a_2^2 + 2a_1 a_2,$$

where $\cdot$ denotes the scalar (inner) product of two vectors. Since $\| T_1 \| = \| T_2 \| = 1$, we obtain

$$a_1 a_2 (T_1 \cdot T_2 - 1) + \frac{\| S \|^2}{2} = a_1 (S \cdot T_1) + a_2 (S \cdot T_2)$$

or

$$a_2 (a_1 (T_1 \cdot T_2 - 1) - S \cdot T_2) = a_1 S \cdot T_1 - \frac{\| S \|^2}{2}. \quad (3)$$

General solution

In the general case, where $a_1(T_1 \cdot T_2 - 1) \neq S \cdot T_2$, using Equation (3) yields

$$a_2 = \frac{a_1 (S \cdot T_1) - \frac{1}{2} \| S \|^2}{a_1 (T_1 \cdot T_2 - 1) - S \cdot T_2}.$$

Degenerate cases

Let us consider the degenerate case of Equation (3), where $a_1(T_1 \cdot T_2 - 1) = S \cdot T_2$. Replacing 1 with $T_2 \cdot T_2$ and S with $P_2 - P_1$ in the above condition yields

$$(P_2 + a_1 T_2 - (P_1 + a_1 T_1)) \cdot T_2 = 0.$$

Control polygons may easily be constructed in this degenerate case if one notes that the line segment joining the point $B_1 = P_1 + a_1 T_1$ with the point $B_2' = P_2 + a_1 T_2$ must be orthogonal to T_2, and therefore is tangent at B_2' to the sphere of center P_2 and radius a_1.

First let us examine the subcase where the right side of Equation (3) is not null, i.e.,

$$a_1 S \cdot T_1 \neq \frac{1}{2} \| S \|^2.$$

Clearly, there is no finite a_2 that satisfies the general equation. However, if we let a_2 go to infinity we obtain a control polygon with one vertex at infinity, i.e., a degenerate control polygon that corresponds to a semicircle, as shown in **Figure 4**.

Consider now the subcase where the right-hand side of Equation (3) is null:

$$a_1 S \cdot T_1 = \frac{1}{2} \| S \|^2.$$

The general equation is satisfied for any value of a_2. This situation is characterized by the two simultaneous equations

$$2a_1 (S \cdot T_1) = \| S \|^2$$

and

$$a_1 (T_1 \cdot T_2 - 1) = S \cdot T_2.$$

The first equation constrains B_1 to lie in the plane normal to S and passing through the midpoint of P_1 and P_2. (The projection of $a_1 T_1$ on the direction of S is equal to half the length of S.)

If $T_1 \cdot T_2 = 1$, then $T_1 = T_2$ and $S \cdot T_2 = 0$, which implies that $S \cdot T_1 = 0$ and that $\| S \| = 0$. This corresponds to a degenerate case, where the bi-arc reduces to a single point ($P_1 = P_2$ and $T_1 = T_2$).

Suppose now that $T_1 \cdot T_2 \neq 1$. We can eliminate a_1 from the system of two equations and obtain

$$2(S \cdot T_2)(S \cdot T_1) = \| S \|^2 (T_1 \cdot T_2 - 1).$$

Dividing this equation by $\| S \|^2$, and choosing a coordinate system such that S is aligned with the X-axis, we obtain

$$2x_1x_2 = x_1x_2 + y_1y_2 + z_1z_2 - 1,$$

which yields

$$x_1x_2 - y_1y_2 - z_1z_2 = -1,$$

where $T_1 = (x_1, y_1, z_1)$ and $T_2 = (x_2, y_2, z_2)$. Define the real quantity g as

$$g = (x_1 + x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2.$$

Using $x_1^2 + y_1^2 + z_1^2 = 1$ and $x_2^2 + y_2^2 + z_2^2 = 1$, we obtain

$$g = x_1^2 + 2x_1x_2 + x_2^2 + y_1^2 - 2y_1y_2 + y_2^2 + z_1^2 - 2z_1z_2 + z_2^2$$

$$= x_1^2 + y_1^2 + z_1^2 + x_2^2 + y_2^2 + z_2^2 + 2(x_1x_2 - y_1y_2 - z_1z_2)$$

$$= 1 + 1 - 2 = 0.$$

Because g is the sum of three positive terms, each term must be null:

$$x_1 = -x_2,$$

$$y_1 = y_2,$$

$$z_1 = z_2.$$

It follows that $S \cdot T_1 = -S \cdot T_2$, and that the vector $T_1 - T_2$ is parallel to S. Since $S \cdot (2a_1T_1) = \| S \|^2$, we obtain

$$a_1(T_1 - T_2) = S,$$

which yields

$$B_1 = P_1 + a_1T_1 = P_1 + S + a_1T_2 = P_2 + a_1T_2.$$

This implies that $\| B_1 - P_2 \| = a_1$, and that B_1 lies on the line passing through P_2 and parallel to T_2. It follows that the first control triangle is (P_1, B_1, P_2). The second control triangle is (P_2, B_2, P_2), and it represents a circle of zero radius, which produces a cusp at P_2 regardless of the value of a_2. This situation is depicted in **Figure 5**.

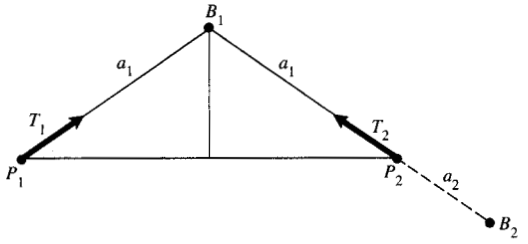

Figure 5

Bi-arc with a cusp: In the special case where $a_1(T_1 - T_2) = S$, the second triangle degenerates into a cusp and a_2 is unconstrained.

• *Selecting a control polygon*

We showed in the previous section how to compute a_2 and derive the control polygon of an interpolating bi-arc, given the interpolated points and tangents and the value of a_1.

The shape of the bi-arc depends on the value of a_1. When the exact curve is known (but is complicated and therefore needs to be approximated), a_1 can be chosen to minimize some error measure for the approximation. When the exact curve is not known, a choice for a_1 may be dictated by the desire to minimize the curvature, the total arc length, the twist, or any other characteristic of the resulting bi-arc.

Using numerical methods when no closed solution was available, we investigated several of these optimization criteria and concluded that in certain cases they lead to surprising and undesirable side effects (awkward-looking curves), and do not offer any significant advantage over the scheme described below, which does not require expensive computations.

• *Equisided control polygon*

Excellent results are obtained by choosing $a_1 = a_2$, which produces very regular curves and leads to an efficient evaluation of a_1. For example, we found experimentally that a bi-arc with $a_1 = a_2$ is very close to a Bezier cubic curve that interpolates the same boundary conditions as the bi-arc and passes through the junction of the two arcs.

With $a_1 = a_2 = a$, $S = P_2 - P_1$, and $T = T_1 + T_2$, Equation (2) becomes $\| S - aT \|^2 = 4a^2$, which can be written as $\| S \|^2 + a^2 \| T \|^2 - 2a(S \cdot T) = 4a^2$, or

$$a^2(\| T \|^2 - 4) - 2a(S \cdot T) + \| S \|^2 = 0. \quad (4)$$

This is a second-degree equation in a and has for discriminant

$$d = (S \cdot T)^2 + \| S \|^2(4 - \| T \|^2). \quad (5)$$

Because $\| T \| \leq 2$, d is the sum of two nonnegative terms and therefore must be positive or null.

General solution

If $\| T \|^2 \neq 4$, we have two real solutions. Of the two roots, we chose

$$a = \frac{\sqrt{d} - S \cdot T}{4 - \| T \|^2}.$$

Since $d \geq (S \cdot T)^2$, a is always positive, and therefore this solution produces arcs of less than 180 degrees. The second root is negative and therefore corresponds to arcs of more than 180 degrees.

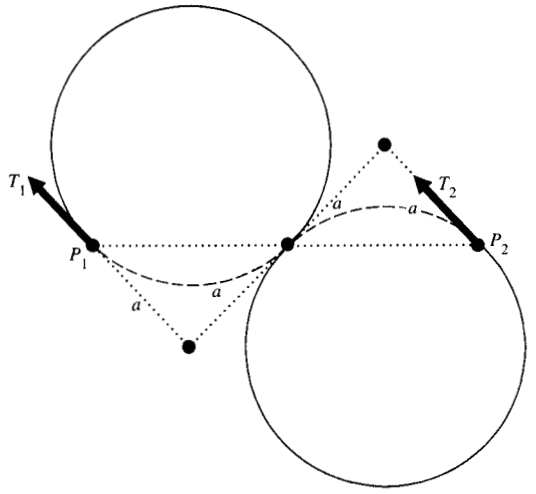

Figure 6

Negative control polygon: If the coefficient a is negative, we use circular arcs of more than 180 degrees defined by the control polygon.

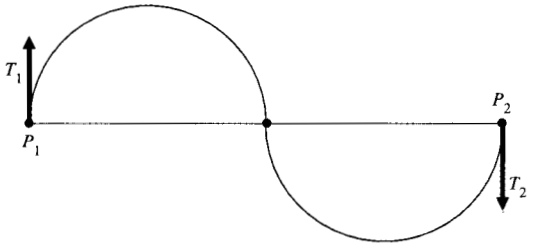

Figure 7

Bi-arc with two semicircles: When $T_1 = T_2$ and $S \cdot T_1 = 0$, the bi-arc is composed of two semicircles.

Degenerate case

The special case where $\| T \|^2 = 4$ is equivalent to $T_1 = T_2$.

If $S \cdot T_1 \neq 0$, then

$$a = \frac{\| S \|^2}{4(S \cdot T_1)}$$

and if $S \cdot T_1 < 0$, then a is negative, and arcs of more than 180 degrees must be used (see **Figure 6**).

If $S \cdot T_1 = 0$, a becomes infinite and we use semicircles, as shown in **Figure 7**.

In the rest of this paper we use only equisided control polygons. Our experience indicates that they are adequate both for the approximation of surface/surface intersections and for the interactive design of free-form space curves (see the section on applications of PCCs).

However, for other applications, such as the approximation of known but more complex curves by PCCs, a different choice of a_1 may be appropriate. For example, the approximation of ellipses by PCCs has many applications in CAD/CAM and will be addressed in a separate paper.

Storage costs

Representations of PCCs in terms of their control polygons are well suited for graphics, but trigonometric representations are preferable for the computation of curve/surface intersections and of point/curve minimum distances. Several representations for PCCs are presented in [15], where we analyze their storage cost and propose formulae for efficiently converting between one representation and another.

Storage cost is a major concern in industrial use of solid modelers, because models that involve a large number of faces and edges are frequent [16]. Our experience shows that an average edge may be adequately approximated with a PCC that involves roughly 20 bi-arcs. Each bi-arc requires between 5 and 28 real numbers, depending on the specific representation scheme adopted. (We used a scheme that requires 8 real numbers [15] in the experimental implementation discussed below.) Representations based on exact closed-form equations for edges, when available, are preferable to the corresponding PCC approximations, because such exact representations require at least an order of magnitude less storage.

PCC approximation of intersection edges

Most solid modeling systems compute surface/surface intersections to obtain the edges of solids defined by Boolean operations [4] on primitive solids.

An edge is a segment of the curve of intersection of two surfaces. Applications that use intersection curves typically are built upon a few primitive tools which include the following:

Producing a sequence of points evenly spaced along a curve.
Computing the intersections of a curve with a surface.
Computing the minimum distance between a point and a curve.

The development of such tools is greatly simplified if a simple closed-form *parametric* equation is available for each curve. We noted in the introduction that closed-form equations exist for intersections of quadrics, but not for

general intersections involving toroidal surfaces. This section explains in detail how PCC approximations may be used to address intersection problems when closed-form solutions are unavailable.

• *Approximation of surface/surface intersections*
Two PCC-based approaches are available to designers of systems for modeling solids bounded by natural surfaces:

1. Approximate all intersection edges with PCCs. This approach simplifies the implementation, because it requires a curve/surface intersection procedure for only one type of edge.
2. Use exact expressions for quadric intersections and use PCCs only when tori are involved.

The first approach was used by us in an experimental modeler, to test the technique discussed below. The second approach has been implemented recently in the UNISOLIDS modeler by McDonnell Douglas.* Experimental evidence indicates that, as expected, for natural quadric surfaces the homogeneous (first) approach is more expensive in time and storage than the second.

Generation of intersection curves
This section discusses the computation of PCC approximations for intersection edges between any two natural faces (i.e., faces of natural primitives: blocks, spheres, cylinders, cones, and tori, shown in **Figure 8**). Our method can be extended to other surface patches and can be modified easily to generate smooth piecewise-cubic curves instead of PCCs. It interpolates the exact intersection curve at discrete points, unlike an alternative technique known as *recursive subdivision*, which converts the faces into a grid of low-degree patches, often planar facets, whose intersections do not interpolate the exact curve [18, 19]. For intersections of tori, our approach is simpler and more reliable than *tracing methods*, which march along an intersection edge from a starting point [20], and compute the next point iteratively. Tracing often fails in the vicinity of singularities. Our method is a variation of the parametric grid method [21] and is composed of three steps:

1. Generate intersection points and tangents.
2. Sort the points along the edges.
3. Interpolate the points with PCCs.

To meet accuracy requirements, one may have to iterate this sequence by recursively refining the approximation in selected intervals.

* D. L. Vossler, McAuto, McDonnell Douglas Automation Company, Cyprus, CA, December 1986, private communication.

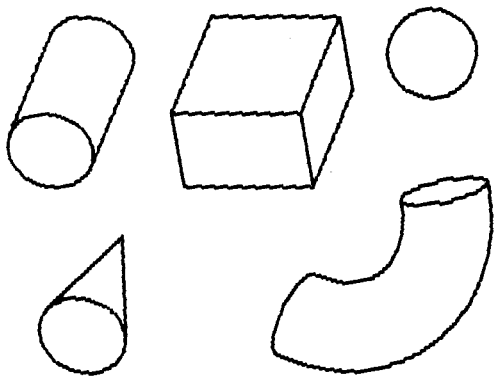

Figure 8

Natural primitives: Block, sphere, cylinder, cone, and torus segment.

Before we describe the steps in detail, let us summarize their effect and indicate how they differ from techniques reported elsewhere.

The first step requires the computation of curve/surface intersections. For the faces of natural primitives, such computations are reduced to the intersection of lines or circles with natural surfaces. An efficient implementation of such computations is described in [15]. For each intersection point, the tangent to the intersection curve is derived from the cross product of the normals to both surfaces at the intersection point.

The second step is the source of inefficiencies in the algorithms discussed in [20] and [21]. We map intersection points to an array of cells in the two-dimensional parameter space of one primitive face. (Hereafter, for brevity we refer to primitive faces as patches.) Our method locates consecutive points practically without searching. Because heuristic methods are inadequate, rare matching ambiguities are addressed by recursively subdividing certain cells into four subcells.

In the last step, we interpolate pairs of consecutive intersection points and tangents with twisted bi-arcs, thus generating a PCC that interpolates the actual edge at all intersection points. Computation of twisted bi-arcs and representation of the PCC were discussed earlier.

Step 1: Constant-parameter curves
Given a solid S, defined as a Boolean combination of primitives P_i, any face of S is contained in the union of the faces of all P_i. Faces of natural primitives can be

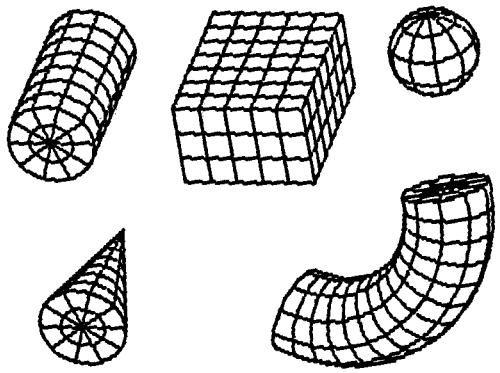

Figure 9

Constant-parameter curves on faces of natural primitives: Lines and circles.

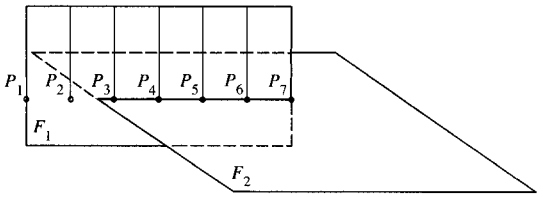

Figure 10

Intersection point lying in one face only: The point P_2 is out of F_2, but is needed to specify the bi-arc (P_2, P_3), which partly lies on F_2.

parameterized in such a way that constant-parameter curves correspond to lines of curvature and are circular arcs or line segments (see **Figure 9**). A patch can be represented in its biparametric form by a vector-valued function $F(u, v)$ that maps a rectangle in parameter space ($u \in [u_1, u_2]$ and $v \in [v_1, v_2]$) into a natural face. Constant-parameter curves (also called *generators*) are *u-curves* of the form $C_u(t) = F(u, t)$, and *v-curves* of the form $C_v(t) = F(t, v)$.

Let F_1 and F_2 be two patches contained respectively in the natural surfaces S_1 and S_2. We assume that the intersection of S_1 with S_2 is a set of dimension one or is empty. Singularities and degeneracies are discussed below in a separate subsection.

We generate points that lie on F_1 and on S_2 by using a grid of generators (u-curves and v-curves) of F_1. We compute the intersection points of each generator with S_2. Unless we are in a special case where the whole generator lies on S_2, we obtain at most four intersection points [15]. For example, given a constant-u curve C_u of F_1, we compute its intersection points $C_u(v_i)$ with S_2. These intersection points are represented in the parametric space of F_1 by the parameter values (u, v_i), and lie on F_1 and on S_2 but can be out of F_2. Points out of F_2 could be easily rejected, since, given a point P on S_2, it is easy (for natural surfaces) to obtain its (u, v) values in the parameter space of F_2 and compare these with u- and v-limits that correspond to the boundaries of F_2. However, we keep such points, because they define bi-arcs (**Figure 10**) that may lie partly on F_2 and partly out of it. (A bi-arc approximates a segment E of the intersection edge between S_1 and S_2. By saying that the bi-arc is partly on F_2 we mean that E is partly on F_2, since in general the bi-arc is not entirely on any of the intersecting surfaces.)

It is necessary to generate similar intersection points for v-curves of F_1; otherwise we could miss large intersection edges that are almost parallel to u-curves of F_1 (**Figure 11**).

The tangents to intersection points are used as additional constraints for the approximation. The exact tangent to the intersection curve at an intersection point P can be computed as the cross product of the normals at P to S_1 and to S_2. Normals to natural surfaces are simple to obtain. When the two normals are collinear, the two surfaces are tangent, which implies either a self-crossing of the intersection edge or a point or curve of tangency. These cases are discussed in the subsection on singularities.

It is desirable to produce intersection points which are evenly spaced (in a rough sense) and to provide an upper bound on the size of details missed by the approximation. The generators form a grid, which divides the patch F_1 into *cells*. The cell that corresponds to the parameter interval $[u_1, u_2] \times [v_1, v_2]$ is defined by the function $F(u, v)$ for $u \in [u_1, u_2]$ and $v \in [v_1, v_2]$. The "size" of the cell is controlled by the two increments $\delta u = u_2 - u_1$ and $\delta v = v_2 - v_1$, and the boundary of such a cell is composed of four segments: the segments of $C_{u_1}(v)$ and of $C_{u_2}(v)$ for $v \in [v_1, v_2]$, and the segments of $C_{v_1}(u)$ and of $C_{v_2}(u)$ for $u \in [u_1, u_2]$. For natural patches, one can easily compute values of δu and of δv that correspond to the desired precision.

The parameterization we have chosen has the drawback of producing some cells that degenerate into triangles (for example at the center of a disk). Intersection curves that pass close to points where many such triangles meet generate a dense and nonuniform distribution of intersection points. All these points are needed for our matching procedure, but many may be redundant in the final PCC, because a PCC obtained with a subset of these intersection points may

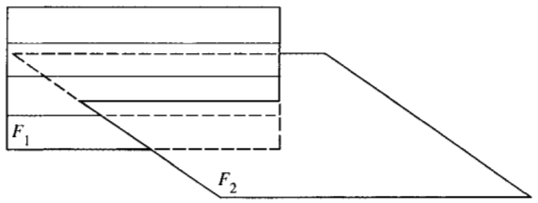

Figure 11

Intersection almost parallel to u-curves: Using only u-curves of F_1, we miss the intersection edge.

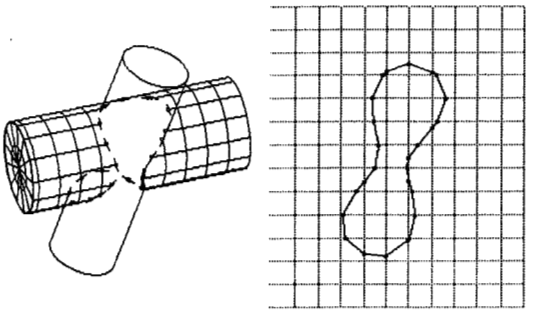

Figure 12

Parametric grid: Intersection points (left) are mapped into cells of the parametric grid (right).

approximate the real curve with enough precision. The precision tests described below may be used to determine whether a given intersection point can be eliminated. The elimination itself is trivial.

Step 2: Matching intersection points

The parameters v_i of the intersections of generators C_u of F_1 with S_2 are stored in an array $V[u]$ indexed by discrete values of u. Each element of the array contains at most four v values, because C_u can intersect S_2 in at most four discrete points. If C_u lies entirely in S_2, it is ignored and the intersection is detected by v-curves. The parameters u_i of the intersections of generators C_v of F_1 with S_2 are stored in a similar array $U[v]$ indexed by discrete values of v. Given a cell defined as $F_1(u, v)$ with $u \in [u_1, u_2]$ and $v \in [v_1, v_2]$, the intersections of its boundary with S_2 contain at most sixteen points, which can be found in four lists. Each list has at most four elements. The intersection points are of the following form:

1. $F_1(u_1, v)$ with v stored in $V[u_1]$ and $v \in [v_1, v_2]$.
2. $F_1(u_2, v)$ with v stored in $V[u_2]$ and $v \in [v_1, v_2]$.
3. $F_1(u, v_1)$ with u stored in $U[v_1]$ and $u \in [u_1, u_2]$.
4. $F_1(u, v_2)$ with u stored in $U[v_2]$ and $u \in [u_1, u_2]$.

Thus, intersection points that lie on the boundary of a given cell can be accessed without searching. If care is taken to count twice certain intersection points that correspond to corners of cells, and tangency points where a generator "touches" S_2, the boundary of each cell contains an even number of intersection points, because S_2 is a closed or infinite surface (see **Figure 12**), and therefore the curve of intersection between S_1 and S_2 cannot terminate (i.e., have an endpoint) within a cell.

If a cell has only two intersection points on its boundary, the matching is trivial. If we have more than two points for a single cell, we use adaptive subdivision to match these intersection points into pairs. (We tried heuristic approaches

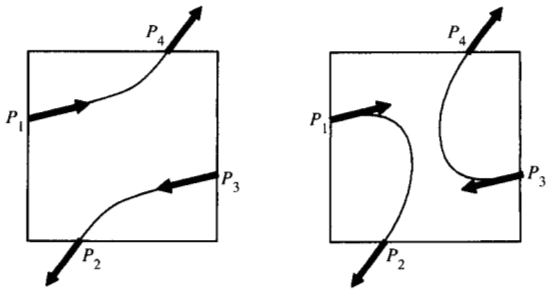

Figure 13

Errors of heuristic matching: Given four intersection points P_1 to P_4, and the associated tangents, most heuristic approaches would match P_1 with P_4 (left), and not with P_2 (right), which might be the correct solution.

based on the direction of the tangents associated with each point, but one cannot guarantee that such approaches will produce the appropriate results, especially near points where the two surfaces are almost tangent. An example is shown in **Figure 13**.)

When the boundary of a cell contains more than two intersection points, or when the bi-arc approximation does not satisfy the precision criterion discussed below, the cell is recursively subdivided into four cells until all precision criteria are met and all matching ambiguities resolved, or until the "size" of the cell is less than a specified limit. A cell is subdivided by using generators that correspond to midvalues of the parameters defining the cell. If the subdivision process reaches a specified minimum size for the cell without resolving the matching ambiguities (more than

two intersection points remain on the boundary of the cell), we could declare that the edge is self-intersecting, but it is easier to match points so that the induced edge segments do not intersect. The sampling points do not provide enough information from which to infer the correct topology of the intersection. We cannot distinguish between possible topologies on the basis of approximate geometric data, and we choose a simple topology: We match points so that they can be connected in the parametric space by noncrossing line segments. (See the subsection on singularities for further discussion.)

Our cell-based approach to the matching problem is simpler and seems more robust than a similar method reported in [13, 22] and used in the BUILD modeler. Is it less efficient? The approach, partly described in [13], uses only u-curves and an isosceles search-triangle to sort the intersection points in parametric space. When an intersection edge is almost parallel to the u-curves, Varady's method reduces the size of the basic step and requires expensive geometric computations to generate intermediate intersection points and to test whether they lie within the search-triangle. In our approach almost all cells have zero or two intersection points on their boundaries, and for these the matching is performed without any geometric calculation.

Step 3: Interpolation

After the matching process, each pair of points (with associated tangents) is interpolated by a *twisted bi-arc*. (Tangential intersection curves may require inferring the tangents, as discussed below.) The result is a geometrically smooth space curve composed of linear and circular segments which interpolates all intersection points and their tangents.

To obtain a continuous parameterization of the whole curve, we must maintain consistency between the orientation of two consecutive bi-arcs. A consistent orientation of the bi-arcs may be propagated along the PCC curve by matching the ending point of one bi-arc with the starting point of the next bi-arc along the curve. Self-intersecting edges are approximated by PCCs that do not reflect their singularities and therefore can be consistently oriented.

Singularities

In the previous discussion we assumed that patches F_1 and F_2 either were disjoint or intersected transversely, producing well-behaved curves. But several singularities may arise:

1. Two surfaces may be coincident.
2. Two surfaces may be tangent at isolated points.
3. Two surfaces may be tangent along a curve.
4. Intersection curves may self-intersect.

Cases 1, 2, and 3 may be detected directly by testing the relative position of one surface with respect to the other, or may be derived from the user's input. An approach for specifying and storing tangency relations between surfaces is described in [23].

We do not attempt to detect singularities of Case 4 and predict situations where the intersection edge crosses itself; instead, we produce an approximation that may not reflect the true topology of the intersection curve, but approximates it within controlled accuracy and corresponds to a valid topology. In our experience such decisions have no consequences as long as they are made consistently. However, maintaining consistency in geometric algorithms that involve floating-point computation is a delicate and poorly understood problem—even when *no* approximations are used—and we make no attempt to solve it here. (Although the correct processing of singularities is mathematically tractable, its numeric implementation leads to severe stability problems which are far from resolved. We consider them outside the scope of this paper.)

Precision

We do not have an explicit representation of the real edge of intersection of tori with other surfaces, and therefore it is difficult to compute a bound on the maximum distance between points on the PCC approximation and the real edge. Instead, we define the error of the approximation as the maximum of the distances from all points of the PCC to the two surfaces that intersect at the approximated edge. This error measure is more practical than the distance between curves because it directly relates to the "tolerance" used in set-membership classification algorithms [4]. For example, if the intersection of two surfaces S_1 and S_2 is approximated within a specified tolerance by a PCC, all points on the PCC must be on S_1 and on S_2 within the same tolerance. A point is considered on a given surface if it lies within a given tolerance from the surface.

Point/surface distance is simple to compute for natural surfaces, but computing its maximum over a circular or even a linear arc is very expensive. For example, the maximum distance between a circular arc and a torus may be derived from the maximum distance between the whole circle and the circular spine of the torus. Computing such a distance requires finding roots of an eight-degree polynomial. Exploratory investigations indicated that evaluation of error bounds usually is more expensive than any refinement it might avoid. Thus, in our implementation we replace the exact maximum distance between an arc and a surface with an estimate computed at specific points of the interpolating curve, or of its convex hull. The optimal choice of such points and the derivation of an error bound remain open issues. On the other hand, if the additional computational cost is acceptable, exact error bounds may be calculated to measure the accuracy of the PCC approximation and to drive subdivision steps that may be necessary to achieve a desired accuracy.

• *Approximating edges of solids*

Classification of PCCs

Each edge of a given solid is a subset of the intersection between two patches. We showed above how to compute an approximation of the intersection of one patch with the surface that contains the other patch. We must now trim this approximation to obtain the subset that corresponds to the actual edge, i.e., to the portion of the real curve that lies on the boundary of the solid. The simplest approach is to classify the approximation with respect to the solid and retain only the part that is on.

If one accepts the risk of missing intersection details that are smaller than one cell, it is economical to start by classifying the intersection points that were used to define the interpolating PCC. If two consecutive intersection points have the same classification (for example, they are both on the solid), assume that the classification is constant along the bi-arc bounded by these two points. On the other hand, if the classification changes between one intersection point and its neighbor, find the set of primitive solids that may have caused such a change of classification, generate the bi-arc between the two intersection points, and compute the intersections of both arcs with the surfaces of the selected primitives. These intersections divide the arcs into segments that have a constant classification. Classifying the midpoint of each segment yields the classification of the segment.

Classification procedures are useful for trimming edges of solids, and for many other applications in geometric modeling.

Neighborhoods

Correct classification of curves that result from Boolean operations on solids having overlapping faces or edges requires information on edge neighborhoods. These can be constructed by a technique used in PADL-2 for quadric intersections. Specifically, compute the plane normal to the edge at a selected point P of the edge, and find the intersection curves of that plane with all the solid's surfaces S_i that contain the edge. Sort the half-curves starting at P around P so that material is contained between the first and second half-curve, between the third and fourth, and so on. Sorting is accomplished by computing tangents to the curves or by using curvature and higher-order derivatives when curves have the same unit tangent.

For PCCs one should construct neighborhoods at the intersection points used to define the PCC because (within the tolerance implied by floating-point accuracy) these points lie on all the surfaces S_i, and the tangent direction computed as the cross product of the normals to any two of these surfaces is tangent to all S_i. Selecting other points of the PCC may lead to incorrect neighborhoods, since these points generally do not lie in any of the intersecting surfaces.

Computational complexity

The average complexity of boundary evaluation is unknown. Worst-case bounds are easy to derive, but experimental data indicate that these bounds are too pessimistic to be useful.

The following is a comparison of the time and storage costs associated with the generation and classification of an edge by PCC and by closed-form methods.

Average values used in this comparison are rough estimates found experimentally. Generating an "average" PCC consisting of 20 bi-arcs to approximate the intersection of two surfaces involves in the order of 40 curve/surface intersection calculations. Each of these calculations requires finding the roots of a polynomial of degree ≤ 4. Additional cost comes from the precision tests and the occasional subdivision discussed earlier. Thus, PCC generation is much more expensive than closed-form solutions, when these are available.

Curve/solid classification by the brute-force method involves intersecting the curve with each surface bounding the solid (or with all surfaces bounding primitive solids, if the object is defined in Constructive Solid Geometry). If a closed-form equation for an edge is available, curve/surface intersection typically amounts to computing the roots of a single polynomial of degree ≤ 8. Intersecting a PCC composed of 20 bi-arcs with a natural surface amounts to 40 arc/surface intersection calculations that involve roots of polynomials of degree ≤ 4. However, if, as in the speedups mentioned above, we perform curve/surface intersection calculations only for bi-arcs that join points lying on different sides of the surface, the PCC/surface intersection calculation might be reduced to 20 point/halfspace classifications (quadric equations for quadrics) and a few occasional arc/surface intersections.

We conclude that the main difference between PCC approximations and closed-form solutions (when available) for edges of solids lies in the cost of generating surface/surface intersections.

The number of curves (tentative edges) generated and classified in boundary-evaluation algorithms is proportional to the square of the number of primitives. Thus, the penalty for using PCC approximations when closed-form solutions are available is large, and we do not recommend this approach. On the other hand, when tori are involved, and for offsetting operations, no closed-form solution is available, and, for reasons outlined in the introduction, PCCs seem an excellent choice.

Utilities for PCCs

In this section we present several tools for PCCs that are used by the applications discussed in the next section.

• *Recursive subdivision*

To display or classify an arc, it is useful to subdivide it into two parts. Representation in terms of control triangles is

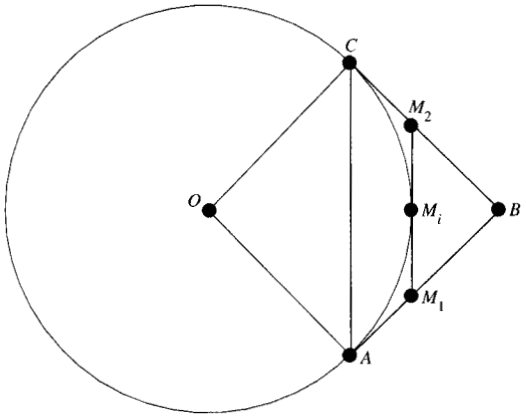

Figure 14

Subdivision: The arc defined by (A, B, C) is subdivided into two arcs defined by (A, M_1, M_i) and (M_i, M_2, C).

well-suited for display and subdivision. Given a control triangle (A, B, C), we produce the two control triangles (A, M_1, M_i) and (M_i, M_2, C) (see **Figure 14**) of the two half-arcs by the following sequence of

$$U = A - B, \qquad V = C - B, \qquad Y = C - A,$$

$$a = \|U\|, \qquad h = \frac{\|Y\|}{2}, \qquad m = \frac{a}{a + h},$$

$$M_1 = mU + B, \qquad M_2 = mV + B, \qquad M_i = \frac{M_1 + M_2}{2}.$$

The value of m is derived from the proportionality between the two triangles (A, B, C) and (M_1, B, M_2) in Figure 14. For simplicity we assume that a is not infinite and that both a and h are not null. The treatment of degenerate arcs is very simple and is not discussed here.

• *Point/curve minimum distance and normal projections*

The minimum distance between a point and a smooth curve may only occur at the normal projections of the point on the curve. The "normal projections" of a point $P = (x, y, z)$ on a curve C are the points H_i of C for which the vector PH_i is orthogonal to the tangent to C at H_i. For PCCs, one needs only to project on lines and circles. Without loss of generality, we consider local coordinate systems such that a line lies on the Z-axis of its local coordinate system, and a circle lies in the $z = 0$ plane and is centered at the origin of its local coordinate system. Let C be a whole line or a complete circle and let (x, y, z) be the coordinates of P expressed in the local coordinate system of C.

When C is a line, the normal projection of P is $H = (0, 0, z)$. When C is a circle of radius r, if P is not on the Z-axis there are two normal projection points H_1 and H_2, having for coordinates $(qx, qy, 0)$ and $(-qx, -qy, 0)$, where $q = r/\sqrt{x^2 + y^2}$. If P is on the Z-axis, all the points of the arc are normal projections of P, and any one of them can be used for computing point/curve distance or point/sweep set-membership classification by the algorithms described below.

To obtain the normal projections of P on a segment of a curve C, compute normal projections on the whole curve and discard points that fall outside the segment. The set of normal projections of P on a PCC is the union of normal projections of P on all the arcs that form the PCC.

• *Inferring tangents*

When tangent data are not available, the following scheme may be used for inferring reasonable tangents from the position of neighboring points.

Our method is based on three circles defined by five consecutive points. Three noncollinear points P_{l-1}, P_l, P_{l+1} define one circle C_l. The center O_l and radius r_l of the circle, as well as the normal N_l to the plane that contains the circle, are computed as follows [7]:

$$V = P_{l-1} - P_l, \qquad W = P_{l+1} - P_l, \qquad N_l = \|V \times W\|,$$

$$R = \frac{\|V\|^2}{2}(W \times N_l) + \frac{\|W\|^2}{2}(N_l \times V), \; O_l = P_l + R,$$

$$r_l = \|R\|.$$

The circle C_j lies in the plane normal to N_j and passes by the points P_{j-1}, P_j, P_{j+1}. The tangent direction $T_{j,i}$ to C_j at a point P_i is the unit vector parallel to $N_j \times (O_j - P_i)$. $T_{l,l}$ could be used for estimating the tangent T_l associated with the sampling point P_l. However, this method is too local and can produce somewhat "unnatural" waves. Much better results are obtained by using a weighted average of tangents to three circles constructed over consecutive points (**Figure 15**):

$$T_l = \frac{w_{l-1}T_{l-1,l} + w_lT_{l,l} + w_{l+1}T_{l+1,l}}{w_{l-1} + w_l + w_{l+1}},$$

for $l = 3 \cdots n - 2$. For noncyclic curves, tangents at the first and last two points are inferred from only one or two circles.

When the weights w_j are all equal, a smooth curve with small curvature variations is obtained. In order to correctly approximate straight lines defined by three or more collinear points, we make the weights proportional to the corresponding radii ($w_l = r_l$). This produces infinite weights for collinear points (**Figure 16**). Another weight formula for combining three tangents is proposed in [24], where the coefficients w_l are slightly more complicated but do not seem to produce significantly better results than ours.

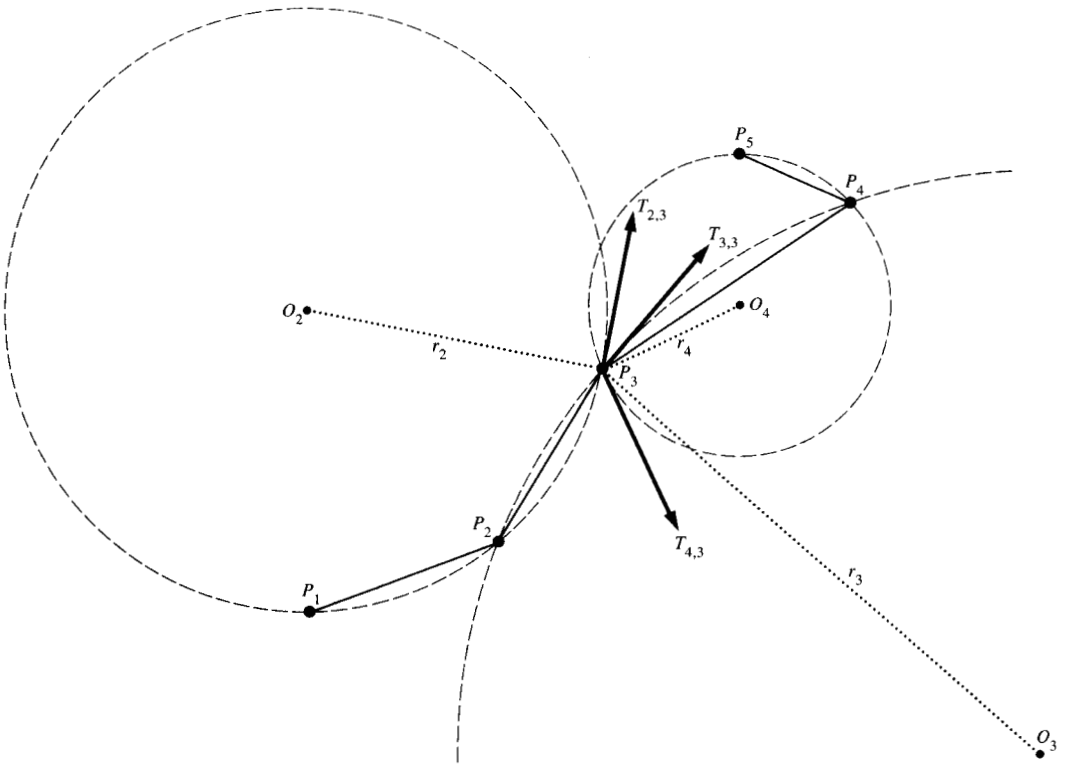

Figure 15

Tangent approximation: The tangent at P_3 is approximated by the weighted average of the unit tangents at P_3 to three circles: the circle passing by (P_1, P_2, P_3), the circle passing by (P_2, P_3, P_4), and the circle passing by (P_3, P_4, P_5).

Applications of PCCs

PCCs have many potential applications in geometric modeling, in addition to the approximation of intersection edges discussed earlier. A few of these applications are described in this section.

- *Wire-frame display*

PCCs offer an economical representation of smooth approximations for space curves. Practical experience indicates that PCCs require at least one order of magnitude fewer segments than linear approximations in order to produce wire-frame displays of comparable accuracy and visual smoothness. PCC representations can be stored in display devices to allow real-time transformations, and can be displayed by a fast algorithm that recursively subdivides circular arcs until their projection on the screen falls within one pixel or is almost linear. Subdivision of PCC arcs was described in the previous section.

- *Interactive design*

PCCs for defining and processing 2D contours

PCCs may be used to define 2D contours composed of linear and circular segments. A contour may be specified and modified graphically in terms of control points which define line segments, fillets for corners, or points to be interpolated with a smooth PCC.

Such contours contain only circular or linear segments, and therefore are closed under normal offsetting [10]. That is, exact offset contours are also composed of only linear and circular segments. Offsets of contours that bound 2D regions are used to drive numerically controlled cutting tools in

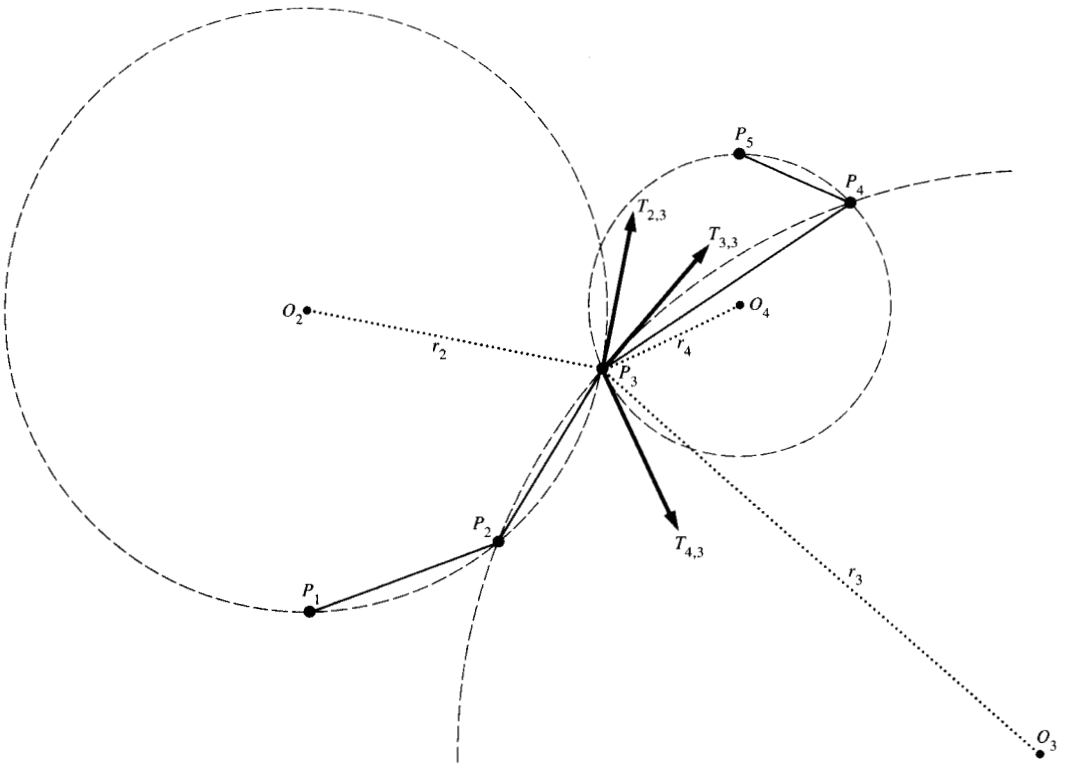

Figure 15

Tangent approximation: The tangent at P_3 is approximated by the weighted average of the unit tangents at P_3 to three circles: the circle passing by (P_1, P_2, P_3), the circle passing by (P_2, P_3, P_4), and the circle passing by (P_3, P_4, P_5).

Applications of PCCs

PCCs have many potential applications in geometric modeling, in addition to the approximation of intersection edges discussed earlier. A few of these applications are described in this section.

- *Wire-frame display*

PCCs offer an economical representation of smooth approximations for space curves. Practical experience indicates that PCCs require at least one order of magnitude fewer segments than linear approximations in order to produce wire-frame displays of comparable accuracy and visual smoothness. PCC representations can be stored in display devices to allow real-time transformations, and can be displayed by a fast algorithm that recursively subdivides circular arcs until their projection on the screen falls within one pixel or is almost linear. Subdivision of PCC arcs was described in the previous section.

- *Interactive design*

PCCs for defining and processing 2D contours

PCCs may be used to define 2D contours composed of linear and circular segments. A contour may be specified and modified graphically in terms of control points which define line segments, fillets for corners, or points to be interpolated with a smooth PCC.

Such contours contain only circular or linear segments, and therefore are closed under normal offsetting [10]. That is, exact offset contours are also composed of only linear and circular segments. Offsets of contours that bound 2D regions are used to drive numerically controlled cutting tools in

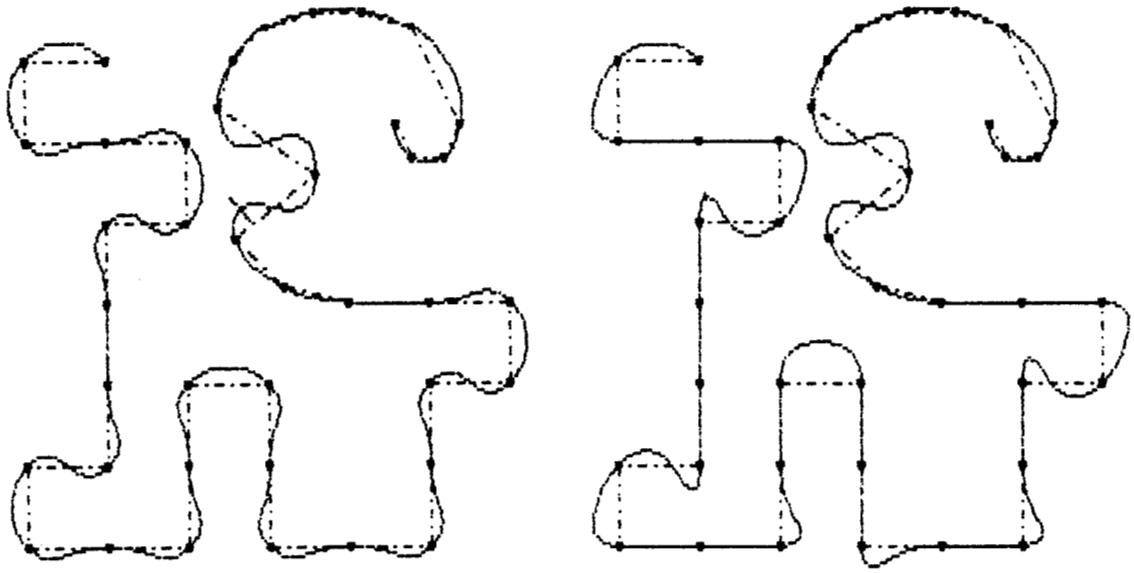

Figure 16

Tangent and weights: A smooth curve (left) is produced when tangents are approximated using equal weights. To interpolate three or more collinear points with a straight line we introduce weights that are proportional to the radii of the corresponding circles (right).

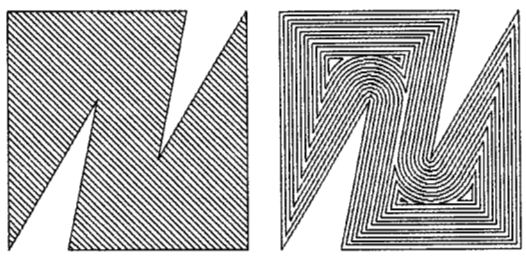

Figure 17

Offsetting 2D contours: The contour (left) can be recursively contracted (right).

computer-aided manufacturing systems, and recursive offsetting (**Figure 17**) yields pocketing algorithms.

Three-dimensional curves

As shown below, PCCs may be used to design smooth space curves that are particularly interesting for modeling sweeps of planar cross sections. The UNISOLIDS system (see footnote p. 303) provides facilities for defining 3D PCCs by specifying 3D points or points and tangent directions. Specification of tangent directions is not always convenient, and when tangents are not important one can use the scheme described in the previous section.

• *Sweeps and offsets*

Offsets (solids obtained through growing and shrinking transformations), and sweeps (solids defined by sweeping a 2D area along a space trajectory) are important for simplifying the design process and for extending the domain of solid modelers. Using PCC-based approximation techniques, one can incorporate offsets and sweeps in solid modelers that support natural surfaces without extending their geometric domain, i.e., without adding any new surface type. One essentially needs to perform point and curve set-membership classification. This, in addition to curve/surface intersection discussed in [15], requires facilities for computing point/surface and point/curve normal projections, which were discussed in the previous section.

Sweeps

Sweeps of 2D PCC contours along 3D PCC spines can be incorporated into solid modelers that support only natural

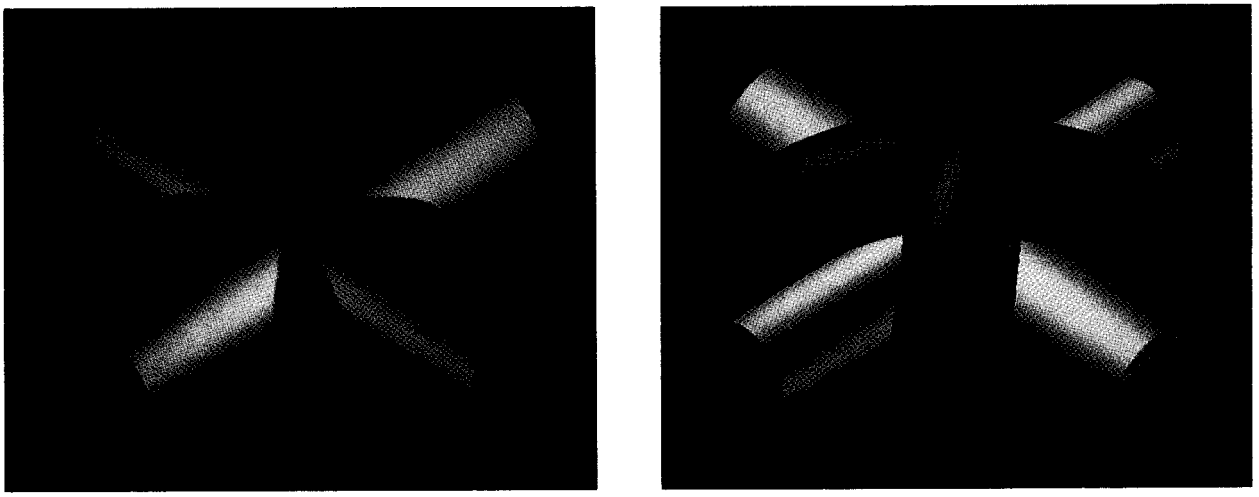

Figure 18

Constant-radius blends: Face of a blend generated by rolling a sphere along rounded edge (left) is bounded by a subset of canal surfaces (right).

surfaces, because sweeping along a PCC spine is equivalent to a sequence of rotational sweeps (for circular spine segments) and translational sweeps (for linear spine segments). Translating lines and circles produces planes and cylinders, rotating lines produces cylinders or cones, and rotating circles produces spheres or tori. Therefore, a 2D PCC contour swept along a 3D PCC spine produces a solid whose faces lie in natural surfaces.

Set-membership classification algorithms for solids generated by PCC sweeps can be organized as follows:

1. To classify a point P, one computes the normal projections P_i of P on the spine, and then classifies P against the 2D cross sections that contain each P_i. The 2D classification can be performed by the usual methods for classifying points against regions represented by their boundaries.
2. To classify a curve, one first computes all the intersections of the curve with the set of patches generated by sweeping individual arcs of the contour along individual arcs of the spine. Then one classifies an intermediate point of each curve segment defined by two consecutive intersecting points.

Support of offsetting and blending operations

Offset (i.e., expanded or shrunk) solids [12] as well as solids with constant-radius blends [11] have faces that lie on so-called canal surfaces of constant radius (see **Figure 18**). A canal surface is the envelope of a family of spherical surfaces with centers on a space curve called the spine.

Canal surfaces are complex but can be approximated by smoothly joined toroidal or cylindrical faces if their spines are approximated with PCCs. In addition, PCC approximation for intersection edges simplifies point-membership classification with respect to expanded or shrunk solids, because such classification requires the computation of the minimum distance between a point and the edges of the original (non-offset) solid, and distances can be deduced from the normal projections of points on edges. Blending and filleting operations can be modeled by combining growing and shrinking operations on solids [11].

Thus, to support sweeps, offsets, and constant-radius blends in a solid modeler where intersection edges are approximated with PCCs, one needs procedures for computing the normal projections of points on PCCs, in addition to the usual classification procedures provided by most modelers [4].

Experimental implementation and conclusions

An experimental solid modeler that incorporates offsetting operations in constructive solid geometry was implemented by the authors [10]. Canal surfaces and intersection edges of tori, and indeed all intersection edges, were approximated with PCCs in the modeler. The implementation was greatly simplified because we had to deal with only one edge type.

Our experience with PCCs showed that a small number of segments is sufficient to approximate complicated edges with PCCs that cannot be visually distinguished from the exact edges. Timing results were good when compared to those of other schemes for approximating intersection edges of nonquadric surfaces, even though no attempt was made to produce efficient code. For example, wire-frame displays of a simple object (**Figure 19**) were produced in a few seconds on a VAX/780.

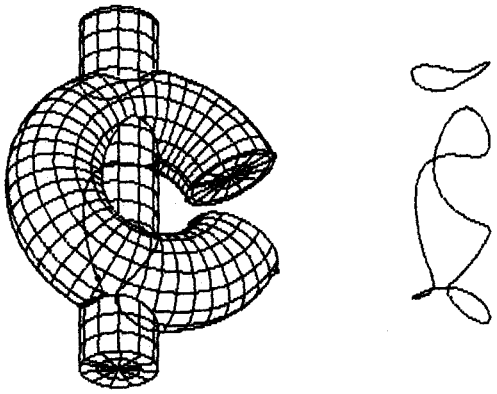

Figure 19

Torus/cylinder intersection: The intersection curve of a toroidal and a cylindrical face (left) was generated (right) in less than two seconds.

The generation and classification of a PCC approximation for the intersection of two natural quadrics is much more expensive than the generation of a closed-form curve representation. Therefore, PCCs are best used in conjunction with analytic solutions when the latter are available. This approach has been successfully implemented in the UNISOLIDS modeler, which uses PCCs to support intersection of tori, as well as free-form curve design for sweeping planar cross sections (see footnote p. 303).

In summary, this paper describes a new technique for interpolating points and tangent vectors by piecewise-circular curves, and shows that PCCs have many applications in geometric modeling. They are well-suited for approximating intersection edges in modelers that support the natural surfaces (planes, spheres, cylinders, cones, and tori), because PCCs are easy to compute and display, and lead to simple algorithms. For example, the intersection of a curve approximated by a PCC with a natural surface involves solving fourth-degree equations, and this can be done analytically. In addition, PCCs provide convenient means for incorporating sweeping, blending, growing, and shrinking operations in solid modelers that support only natural surfaces.

Acknowledgments

The initial phase of this work was carried out at the University of Rochester, New York, and was supported by the National Science Foundation under grants ECS-81-04646 and ECS-84-03882, and by the Industrial Associates of the Production Automation Project. The views expressed here are solely those of the authors, who thank the members of the Design Automation group at the IBM Thomas J. Watson Research Center for their constructive comments regarding this manuscript.

References

1. A. A. G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Comput. Surv.* **12,** No. 4, 437–464 (December 1980).
2. George M. Koppelman and Michael A. Wesley, "OYSTER: A Study of Integrated Circuits as Three-Dimensional Structures," *IBM J. Res. Develop.* **27,** No. 2, 149–163 (March 1983).
3. R. B. Tilove, "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," *IEEE Trans. Computers* **C-29,** No. 10, 874–883 (October 1980).
4. A. A. G. Requicha and H. B. Voelcker, "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE* **73,** No. 1, 30–44 (January 1985).
5. R. C. Hillyard, "The BUILD Group of Solid Modellers," *IEEE Comput. Graph. & Appl.* **2,** No. 2, 43–52 (March 1982).
6. W. Tiller, "Rational B-Splines for Curve and Surface Representation," *IEEE Comput. Graph. & Appl.* **3,** No. 6, 61–69 (September 1983).
7. I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood, Chichester, U.K., 1979.
8. M. E. Mortenson, *Geometric Modelling*, John Wiley & Sons, Inc., New York, 1985.
9. B. A. Barsky and T. D. DeRose, "Geometric Continuity of Parametric Curves," *Report No. UCB/CSD 84/205*, Computer Science Department, University of California, Berkeley, October 1984.
10. J. R. Rossignac, "Blending and Offsetting Solid Models," *Technical Memorandum No. 54* (Ph.D. dissertation), Production Automation Project, University of Rochester, NY, June 1985.
11. J. R. Rossignac and A. A. G. Requicha, "Constant Radius Blending in Solid Modelling," *Comput. in Mech. Eng.* **3,** No. 1, 65–73 (July 1984).
12. J. R. Rossignac and A. A. G. Requicha, "Offsetting Operations in Solid Modelling," *Comput. Aided Geom. Design* **3,** No. 2, 129–148 (August 1986).
13. T. Varady, "Surface-Surface Intersections for Double-Quadric Parametric Patches in a Solid Modeller," Proceedings of the UK/Hungarian Seminar on "Computational Geometry for CAD/CAM," Cambridge University, U.K., September 1983, pp. 1–23.
14. M. Sabin, "The Use of Piecewise Forms for the Numerical Representation of Shape," *Report No. 60*, Computer and Automation Institute, Hungarian Academy of Science, Budapest, 1977.
15. J. R. Rossignac and A. A. G. Requicha, "Piecewise Constant Curvature Approximations for Geometric Modelling," *Research Report RC-12171*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, October 1986.
16. R. N. Wolfe, M. A. Wesley, J. C. Kyle, F. Gracer, and W. J. Fitzgerald, "Solid Modeling for Production Design," *IBM J. Res. Develop.* **31,** 277–295 (May 1987, this issue).
17. C. M. Brown, "PADL-2: A Technical Summary," *IEEE Comput. Graph. & Appl.* **2,** No. 2, 69–84 (March 1982).
18. W. E. Carlson, "An Algorithm and Data Structure for 3D Object Synthesis Using Surface Patch Intersection," *Proceedings of ACM SigGraph '82*, Boston, July 26–30, 1982, pp. 255–263.
19. P. A. Koparkar and S. P. Mudur, "Computational Techniques for Processing Parametric Surfaces," *Comput. Vision, Graph. & Image Proc.* **28,** No. 3, 303–322 (December 1984).
20. H. G. Timmer, "A Solution to the Surface Intersection Problem," *Report No. MDC J7789*, McDonnell Douglas Corporation, Cyprus, CA, November 1977.
21. M. Sabin, "Contouring—A Review of Methods for Scattered Data," *Mathematical Methods in Computer Graphics and Design*, K. W. Brodlie, Ed., Academic Press, Inc., New York, 1980.

22. G. E. M. Jared and T. Varady, "Synthesis of Volume Modelling and Sculptured Surfaces in BUILD," *Proceedings of CAD '84*, Brighton, U.K., April 3–5, 1984, pp. 481–495.
23. J. R. Rossignac, "Constraints in Constructive Solid Geometry," presented at the 1986 Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 22–24, 1986. Proceedings to be published by the ACM. Also available as *Research Report RC-12356*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1986.
24. K. Harada and E. Nakamae, "Application of the Bezier Curve to Data Interpolation," *Comput. Aided Design* **14**, No. 1, 55–59 (January 1982).

Received October 3, 1986; accepted for publication February 24, 1987

Jaroslaw R. Rossignac *IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.* Dr. Rossignac received the Diplôme d'Ingénieur from the Ecole Nationale Supérieure d'Electricité et de Mécanique and a Maîtrise degree in mechanical engineering from the University of Nancy-1, France. In 1985 he received a Ph.D. in electrical engineering from the University of Rochester, New York, for his doctoral research in solid modeling at the Production Automation Project. He worked for the French Navy developing simulation tools in a multiprocess environment and taught real-time programming in the French industry. Since 1985 he has been a research scientist in the design automation group at the IBM Thomas J. Watson Research Center. He has been involved with the design and implementation of tools and experimental systems for geometric modeling and its applications to manufacturing. His research interest include the approximation of curves and surfaces, design of fundamental algorithms for solid modeling, elimination of redundancy in constructive solid geometry, modeling of offsetting, blending, and sweeping operations, generation and interactive manipulation of shaded images of solids and mechanisms, specification of geometry in terms of constraints, and representation and simulation of manufacturing processes.

Aristides A. G. Requicha *University of Southern California, Los Angeles, California 90089.* Dr. Requicha received the *Engenheiro Electrotécnico* degree from the Instituto Superior Técnico, Lisbon, Portugal, in 1962, and the Ph.D. degree in electrical engineering from the University of Rochester, New York, in 1970. He is a professor of computer science and electrical engineering at the Robotics Institute of the University of Southern California in Los Angeles. Before joining U.S.C. in 1986, he was with the Production Automation Project at the University of Rochester, where he engaged in research and education in geometric modeling and programmable automation for thirteen years. He has also been a lecturer in physics at the University of Lisbon and a research scientist for NATO's SACLANT Research Centre in La Spezia, Italy. Dr. Requicha is a senior member of the Institute of Electrical and Electronics Engineers, and a member of the American Association for the Advancement of Science, the American Association for Artificial Intelligence, the Association for Computing Machinery, and Sigma Xi. He is an associate editor for the *ACM Transactions on Graphics*, and was the program chairman for the Eurographics '86 Conference in Lisbon, Portugal.