13.472J/1.128J/2.158J/16.940J
COMPUTATIONAL GEOMETRY

Lectures 10 - 12

N. M. Patrikalakis

Massachusetts Institute of Technology
Cambridge, MA 02139-4307, USA

Contents

Reading in the Textbook

- Chapters 4 and 5, pp. 73 - 160

Lectures 10 - 12
Intersection Problems, Nonlinear Solvers and Robustness Issues

10.1 Overview of intersection problems

Intersections are fundamental in computational geometry, geometric modeling and design, analysis and manufacturing applications. Examples of intersection problems include:

- Shape interrogation (eg. visualization) through contouring (intersection with a series of parallel planes, coaxial cylinders, and cones etc.)
- Numerical control machining (milling) involving intersection of offset surfaces with a series of parallel planes, to create machining paths for ball (spherical) cutters.
- Representation of complex geometries in the "Boundary Representation" scheme; for example, the description of the internal geometry and of structural members of cars, airplanes, ships, etc, involves
  - Intersections of free-form parametric surfaces with low order algebraic surfaces (planes, quadrics, torii).
  - Intersections of low order algebraic surfaces.

  in a process called boundary evaluation, in which the Boundary Representation is created by "evaluating" the Constructive Solid Geometry model of the object. During this process, intersections of the surfaces of primitives (see Figure 10.1) must be found during Boolean operations.

Boolean opertations on point sets A, B include (see Figure 10.2)

- Union: $A \cup B$,
- Intersection: $A \cap B$, and
- Difference: $A - B$.

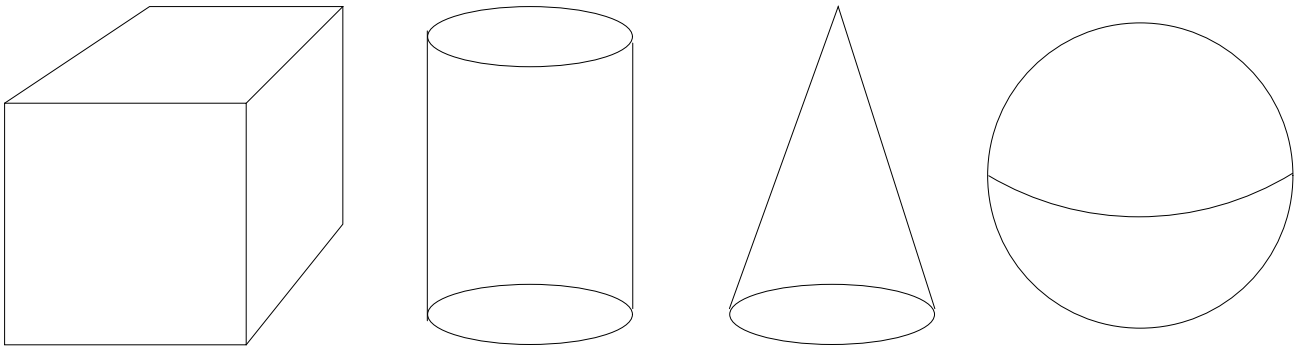


Figure 10.1: Primitive solids.

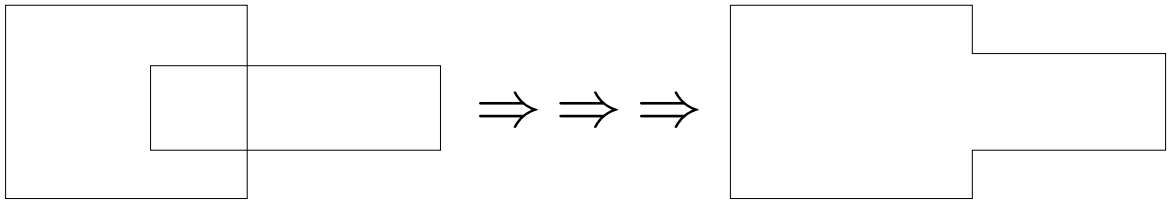


Figure 10.2: Example of a Boolean operation: union.

All such operations involve intersections of surfaces to surfaces. In order to solve general surface to surface intersection problems, the following auxiliary intersection problems (similar to distance computation problems used in CAM for inspection of manufactured objects) need to be considered

- point/point (P/P)
- point/curve (P/C)
- point/surface (P/S)
- curve/curve (C/C)
- curve/surface (C/S)

All above types of intersection problems are also useful in geometric modeling, robotics, collision avoidance, manufacturing simulation, scientific visualization, etc.

When the geometric elements involved in intersections are nonlinear (curved), intersection problems typically reduce to solving complex systems of nonlinear equations, which may be either polynomial or more general in character. Solution of nonlinear systems is a very complex process in general in numerical analysis and there are specialized textbooks on the topic. However, geometric modeling applications pose severe robustness, accuracy, automation, and efficiency requirements on solvers of a nonlinear systems as we will see later. Therefore, geometric modeling researchers have developed specialized solvers to address these requirements explicitly using geometric formulations.

10.2 Intersection problem classification

Intersection problems can be classified according to the dimension of the problems and according to the type of geometric equations involved in defining the various geometric elements (points, curves and surfaces). The solution of intersection problems can also vary according to the number system in which the input is expressed and the solution algorithm is implemented.

10.2.1 Classification by dimension

- P/P, P/C, P/S
- C/C, C/S
- S/S

10.2.2 Classification by type of geometry

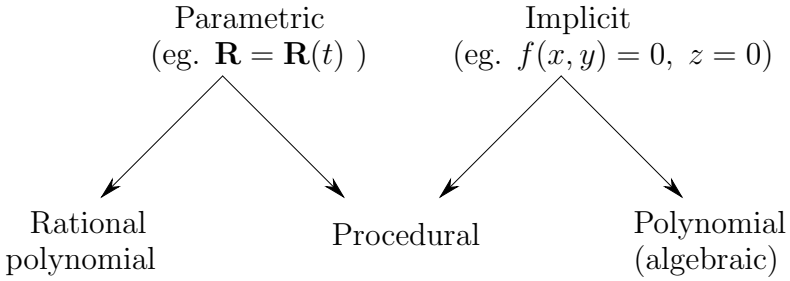


Figure 10.3: Curve geometry classification

1. *Points*
   - Explicit: $\mathbf{R} = \mathbf{R}_0$; $\mathbf{R} = [x, y, z]$
   - Procedural: Intersection of two procedural curves, procedural curve and surface, or three procedure surfaces, eg. offset or blending surfaces.
   - Algebraic: $f(\mathbf{R}) = g(\mathbf{R}) = h(\mathbf{R}) = 0$; where f, g, and h are polynomials.
2. *Curves*
   A classification of curves is illustrated in Figure 10.3.
   - *Parametric*: $\mathbf{R} = \mathbf{R}(t)\ A \leq t \leq B$
     (a) Rational Polynomials (eg: NURBS, rational Bézier).
     (b) Procedural, eg: offsets, evolutes, ie. the locus of the centers of curvature of a curve.
   - *Implicit*: These require solution of (usually nonlinear) equations

(a) Algebraics (polynomial)

$$f(\mathbf{R}) = g(\mathbf{R}) = 0 \qquad space \quad curves$$

$$z = 0,\ f(x, y) = 0 \qquad planar \quad curves$$

(b) Procedural offsets (eg. non-constant distance offsets involving convolution, see Pottmann 1997)

3. *Surfaces*

- Parametric $\mathbf{R} = \mathbf{R}(u, v)$ where u, v vary in some finite domain, the parametric space.
  (a) (Rational) Polynomial (eg: NURBS, Bézier, rational Bézier etc.)
  (b) Procedural
    – offsets
    – blends
    – generalized cylinders
- Implicit: Algebraics

$$f(\mathbf{R}) = 0$$

where f is a polynomial.

10.2.3 Classification by number system

In our discussion of intersection problems, we will refer to various classes of numbers:

- integer numbers;
- rational numbers, m/n, $n \neq 0$, where m, n are integers;
- floating point numbers in a computer (which are a subset of the rational numbers);
- radicals of rational numbers, eg. $\sqrt{m/n}$, $n \neq 0$, where m, n are integers;
- algebraic numbers (roots of polynomials with integer coefficients);
- transcendental (e, π, trigonometric, etc.) numbers.
- real numbers;
- interval numbers, $[a, b]$, where a, b are real numbers;
- rounded interval numbers, $[c, d]$, where c, d are floating point numbers.

Issues relating to floating point and interval numbers affecting the robustness of intersection algorithms are addressed in the next section on nonlinear solvers.

10.3 Point/point "intersection"

- Check if $|\mathbf{R}_1 - \mathbf{R}_2| < \epsilon$, where ϵ represents the maximum allowable tolerance.
- Choice of "tolerances" in a geometric modeller is difficult–an open question.
- Lack of transitivity, see Figure 10.4:

$$\begin{array}{l} \mathbf{R}_1 = \mathbf{R}_2 \\ \mathbf{R}_2 = \mathbf{R}_3 \end{array} \Rightarrow \mathbf{R}_1 \neq \mathbf{R}_3$$

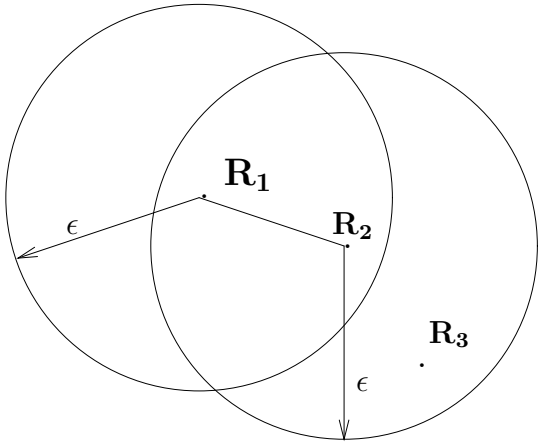


Figure 10.4: Intersections of points within a tolerance is intransitive.

- What should ϵ reflect?

10.4 Point/curve intersection

10.4.1 Point/Implicit curve intersection

$$\mathbf{R}_0 \cap \{z = 0,\ f(x, y) = 0\}$$

where $f(x, y)$ is usually a polynomial (and $f(x, y) = 0$ represents an algebraic curve). In an exact arithmetic context, we can substitute $\mathbf{R}_0$ in $\{z,\ f(x, y) = 0\}$ and verify if the results are zero. Similarly, we could handle:

$$\mathbf{R}_0 \cap \{f(\mathbf{R}) = g(\mathbf{R}) = 0\}$$

where $f(\mathbf{R}) = g(\mathbf{R}) = 0$ represents an implicit 3D space curve.

What does verify mean in "floating point" arithmetic?

- *Example A*:

  Let $z_0 = 0$ and x_0, y_0 satisfy

  $$|f(x_0, y_0)| < \epsilon \ll 1 \tag{10.1}$$

  where ϵ is a small constant and $|f(x, y)| \leq 1$ in the domain of interest including (x_0, y_0), then a "distance" check can be performed by:

  $$\frac{|f(x_0, y_0)|}{|\bigtriangledown f(x_0, y_0)|} < \delta \ll 1 \tag{10.2}$$

  provided $|\bigtriangledown f(x_0, y_0)| \neq 0$. Equation 10.1 is called the "algebraic distance" and Equation 10.2 is called the "non-algebraic distance". The true geometric distance is given by:

  $$d = min|\mathbf{R} - \mathbf{R}_0|; \quad where\ \ \mathbf{R} = (x, y), \quad f(\mathbf{R}) = 0 \tag{10.3}$$

  The true geometric distance is difficult and expensive to compute (particularly for implicit $f(\mathbf{R}) = 0$ and involves computing the *global* minimum of $|\mathbf{R} - \mathbf{R}_0|$. Equation 10.2 results from the first order approximation of Equation 10.3 as derived by Taylor expansion and is exact when $f(\mathbf{R})$ is represents a plane.

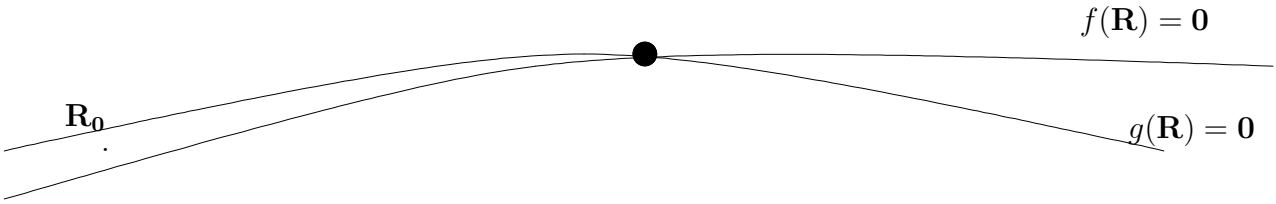


Figure 10.5: Curves meet at small angle.

- *Example B*:

$$\mathbf{R}_0 \cap \{f(\mathbf{R}) = g(\mathbf{R}) = 0\}$$

When curves $f = 0$, $g = 0$ meet at a small angle ($\frac{\nabla f}{|\nabla f|} \cdot \frac{\nabla g}{|\nabla g|} \cong 1$), then the condition

$$|f| < \epsilon \; and \quad \delta_1 = \frac{|f|}{|\nabla f|} < \delta$$
$$|g| < \epsilon \; and \quad \delta_2 = \frac{|g|}{|\nabla g|} < \delta$$

(where $|f|, |g|, \delta_1, \delta_2$ are evaluated with $\mathbf{R} = \mathbf{R}_0$ and $\epsilon, \delta \ll 1$) are not enough to guarantee proximity of $\mathbf{R}_0$ to the intersection of $\mathbf{f}$, $\mathbf{g}$, see Figure 10.5.

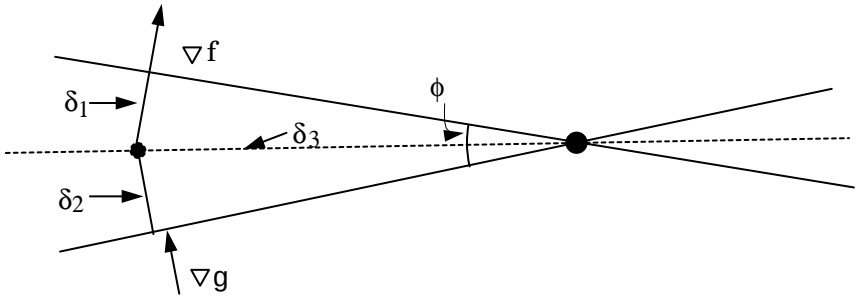


Figure 10.6: Approximate curves with straight lines.

Using a linear approximation, and letting

$$\phi = \cos^{-1} |\frac{\nabla f}{|\nabla f|} \cdot \frac{\nabla g}{|\nabla g|}|$$

be the angle of intersection as in Figure 10.6 near the intersection point, a better criterion for evaluating if $\mathbf{R}_0$ is near the intersection of $\mathbf{f}$ and $\mathbf{g}$ is

$$\delta_3 = \phi^{-1}\{\frac{|f|}{|\nabla f|} + \frac{|g|}{|\nabla g|}\} < \delta \ll 1$$

10.4.2 Point/Parametric curve intersection

1. *Rational polynomial curves*

$$\mathbf{R}_0 \cap \mathbf{R} = \mathbf{R}(t) \quad A \leq t \leq B$$

- *Brute force elementary method*:
  We solve each of the following three nonlinear polynomial equations separately and we search for common real roots in $A \leq t \leq B$.

$$\begin{array}{rcl} x(t) - x_0 = 0 & \rightarrow & t'_1, \cdots, t'_n \\ y(t) - y_0 = 0 & \rightarrow & t''_1, \cdots, t''_n \\ z(t) - z_0 = 0 & \rightarrow & t'''_1, \cdots, t'''_n \end{array}$$

  In principle, this elementary approach is "easy" for polynomials. However, in practice, this process is complex and inefficient and prone to numerical inaccuracies.

- *Preprocessing and subdivision method*
  - Use bounding box of $\mathbf{R}(t)$ to eliminate easily resolvable cases, with some level of subdivision (splitting) to reduce box size.
  - Concept of subdivision in rational arithmetic: To eliminate numerical error in the subdivision process (which can lead to erroneous decisions), rational arithmetic may be employed (if the input coefficients of $\mathbf{R}(t)$ are rational or floating point numbers). This can be easily done in object-oriented languages such as C++ using operator overloading.
  - Continue subdivision until box is small.
  - Then, we could use a numerical technique , such as:

$$F(t) = min\{|\mathbf{R}_0 - \mathbf{R}(t)|^2\} \quad t \in D_1 \subset [A, B]$$

    and use some t from the interval D_1 as the initial approximation. Use of the square of the distance function is necessary to avoid possible divergence of the derivative of the distance function, if it approaches zero.
  - If the minimization process converges to t_0 and $\sqrt{F(t_0)} < \delta$, $t = t_0$ is the desired solution.

- *Implicitization* (perhaps with box preprocessing) such as

$$(x_0, y_0) \ \cap \ \{x = x(t), \ y = y(t)\}$$

  Let us consider an example where $x(t), y(t)$ are quadratic polynomials (the curve is a parabola). We will attempt to eliminate t to get a polynomial $f(x, y) = 0$ which the x, y coordinates of all points on the curve satisfy. We start with the system

$$\begin{array}{l} x = a_0 t^2 + b_0 t + c_0 \\ y = a'_0 t^2 + b'_0 t + c'_0 \end{array} \quad \Rightarrow \quad \begin{array}{l} a_0 t^2 + b_0 t + c_0 - x = 0 \\ a'_0 t^2 + b'_0 t + c'_0 - y = 0 \end{array}$$

which can be rewritten in matrix form as follows:

$$\Rightarrow \quad \begin{bmatrix} c_0 - x & b_0 & a_0 & 0 \\ 0 & c_0 - x & b_0 & a_0 \\ c_0' - y & b_0' & a_0' & 0 \\ 0 & c_0' - y & b_0' & a_0' \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{10.4}$$

The maximum degree of t in the above vector is determined by the degree m of the x polynomial and the degree n of the y polynomial, and is given by $m + n - 1$. In this case $m + n - 1 = 3$.

A necessary and sufficient condition for the above system to be solvable is

$$\begin{vmatrix} c_0 - x & b_0 & a_0 & 0 \\ 0 & c_0 - x & b_0 & a_0 \\ c_0' - y & b_0' & a_0' & 0 \\ 0 & c_0' - y & b_0' & a_0' \end{vmatrix} = f(x, y) = 0$$

The equation $f(x, y) = 0$ is the implicit equation of the curve. Consequently in an exact arithmetic context, we need to check if $f(x_0, y_0) = 0$, to verify if (x_0, y_0) is on the initial curve.

In general, if

$$x = x(t^n), \quad y = y(t^n) \Rightarrow F(x^n, y^n) = 0$$

where n is the total degree.

- *Inversion*: If $f(x, y) = 0$ then we could use the first 3 equations 10.4:

$$\begin{bmatrix} b_0 & a_0 & 0 \\ c_0 - x_0 & b_0 & a_0 \\ b_0' & a_0' & 0 \end{bmatrix} \begin{bmatrix} t \\ t^2 \\ t^3 \end{bmatrix} = - \begin{bmatrix} c_0 - x_0 \\ 0 \\ c_0' - y_0 \end{bmatrix}$$

$$\Rightarrow t = \frac{\phi(x_0, y_0)}{\psi(x_0, y_0)}$$

Where ϕ and ψ are polynomials in x_0 and y_0, and x_0, y_0 satisfy

$$f(x_0, y_0) = 0$$

  - The method is efficient and (usually) accurate for $n \leq 3$ (but no real guarantees on accuracy and robustness exist if the method is implemented in floating point).
  - Subdivision methods are preferable for higher n, and as we will see later when coupled with rounded interval arithmetic are robust, accurate and efficient.

Intersection of points (x_0, y_0, z_0) and 3D polynomial curves $\mathbf{R} = \mathbf{R}(t)$ via implicitization of such curves involves a process of projection on x, y plane and finding t_0 by inversion and verification of $z_0 = z(t_0)$.

10.4.3 Point/Procedural parametric (offset, evolute, etc.) curve intersection

$$\mathbf{R}_0 \cap \mathbf{R} = \mathbf{R}(t) \qquad A \leq t \leq B$$

- In general there is no known and easily computable convex box decreasing arbitrarily with subdivision!
- An approximate solution method may involve minimization of

  $$F(t) = |\mathbf{R}(t) - \mathbf{R}|^2$$

  where $t \in [A, B]$. This would involve
  - Checking end points, ie. if $F(A)$, $F(B)$ are very small.
  - Initial estimate for the possible minima, perhaps using linear approximation of $\mathbf{R}(t)$ to start the process.

  However,
  - Convergence of the above minimization processes is not guaranteed in general.
  - There may exist more than one minima.
  - Convergence to local and not global minimum (where $F(t) \neq 0$) is possible.

For certain classes of procedural curves such as offsets and evolutes of rational curves involving radicals of polynomials, it is possible to use the "auxiliary variable method" to reduce the point to curve intersection (or minimum distance) problem to a set of (a larger number of) nonlinear polynomial equations. Such systems can be solved robustly and efficiently using the nonlinear solver describe in the next section.

10.5 Point/surface intersection

10.5.1 Point/Implicit (usually algebraic) surface intersection

The condition for $\mathbf{R}_0 \cap \{f(\mathbf{R}) = 0\}$, where $f(\mathbf{R}) = 0$ is an implicit surface, is:

$$|f(\mathbf{R}_0)| < \epsilon, \quad \frac{|f(\mathbf{R}_0)|}{|\bigtriangledown f(\mathbf{R}_0)|} < \delta$$

where ϵ, δ are small constants.

10.5.2 Point/Rational polynomial surface intersection

1. Implicitization is possible for all such surfaces but computationally expensive and possibly inaccurate. For a tensor product rational polynomial surface with maximum degrees in u and v equal to m and n, of the form

   $$\mathbf{R} = \mathbf{R}(u^m, v^n),$$

   the implicit equation is

   $$f(x^q, y^q, z^q) = 0$$

   $$where \;\; q \leq 2mn$$

   $$Therefore, for \;\; m = n = 3 \longrightarrow q \leq 18, \;\; m = n = 2 \longrightarrow q \leq 8$$

   The above method is useful for special surfaces such as cylindrical and conical ruled surfaces, surfaces of revolution, etc.

   Examples:

   (a) Implicitization of a surface of revolution.

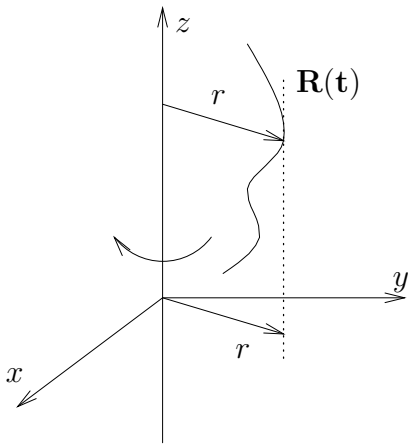


Figure 10.7: Surface of revolution.

Let us consider a profile curve to be a rational polynomial of degree n, see Figure 10.7

$$\mathbf{R}(t) = [r(t), z(t)]$$

By simple implicitization of $\mathbf{R} = \mathbf{R}(t)$, we get:

$$f_n(r, z) = 0 \tag{10.5}$$

where n is the maximum total degree of f. Also,

$$r^2 = x^2 + y^2 \tag{10.6}$$

Next we eliminate r from equations 10.5 - 10.6 by rewriting equations 10.5 - 10.6 as follows:

$$\begin{aligned} f_n(r, z) &= a_0(z)r^n + a_1(z)r^{n-1} + \cdots + a_n(z) = 0 \\ &\Rightarrow \; -r^2 + (x^2 + y^2) = 0 \end{aligned}$$

The resultant of eliminating r from these two equations is

$$D = \begin{vmatrix} a_0(z) & a_1(z) & \cdots & a_{n-1}(z) & a_n(z) & 0 \\ 0 & a_0(z) & \cdots & a_{n-2}(z) & a_{n-1}(z) & a_n(z) \\ -1 & 0 & x^2+y^2 & \cdots & & \\ & -1 & 0 & x^2+y^2 & & \\ & & & & \ddots & \\ & & & -1 & 0 & x^2+y^2 \end{vmatrix} = 0$$

and the degree of $D \equiv f(x, y, z) = 0$ is $2n$. An example is a torus (degree 4 algebraic surface).

(b) Implicitization of a cylindrical ruled surface

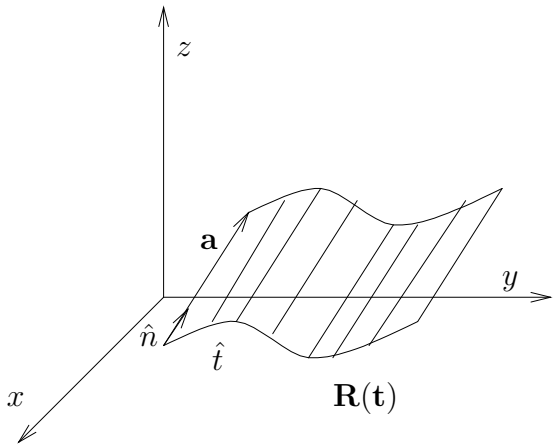


Figure 10.8: Cylindrical ruled surface.

Let

$$\mathbf{R}(t) = [X(t), Y(t)]$$

be a degree n planar rational polynomial curve in the x, y plane. The resulting implicit equation of the curve

$$f(X, Y) = 0$$

is a polynomial of degree n. Let

$$\mathbf{a} = [a_1,\ a_2,\ a_3]$$

be a direction vector. Then the three equations

$$\begin{aligned} x &= X + ua_1 \\ y &= Y + ua_2 \\ z &= ua_3 \end{aligned}$$

describe a cylindrical ruled surface. Hence, the implicit surface equation becomes:

$$f(x - \frac{z}{a_3}a_1, y - \frac{z}{a_3}a_2) = 0$$

This equation can be transformed to the standard form using a symbolic manipulation program such as Macsyma.

(c) Implicitization of a conical ruled surface

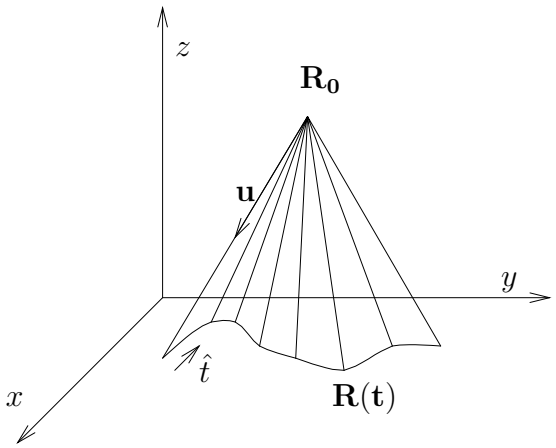


Figure 10.9: Conical ruled surface.

Let

$$\mathbf{R}_0 = [x_0,\ y_0,\ z_0]$$

be the apex of the conical ruled surface and

$$\mathbf{R}(t) = [X(t),\ Y(t)]$$

a degree n planar rational polynomial curve on the x, y plane. Its implicit equation

$$f(x, y) = 0$$

is a degree n polynomial. The equation of the resulting conical ruled surface is

$$\begin{aligned} x &= x_0(1-u) + Xu \\ y &= y_0(1-u) + Yu \\ z &= z_0(1-u) \end{aligned}$$

Eliminating $u = 1 - \frac{z}{z_0}$ and solving for $X,\ Y$ yields:

$$f\left[\frac{z_0}{z_0 - z}x - \frac{x_0}{z_0 - z}z, \frac{z_0}{z_0 - z}y - \frac{y_0}{z_0 - z}z\right] = 0$$

This equation can be transformed to the standard form using a symbolic manipulation program such as Macsyma.

2. Newton's method:
Solve $x_0 = x_0(u, v)$, $y_0 = y_0(u, v)$ and verify the third equation. Use a linear approximation to start the process. Preprocessing using convex bounding box should always be used, coupled with some level of subdivision.

3. Convex box and possibly subdivision followed by minimization in $0 \leq u, v \leq 1$ or within a rectangular subdomain of the following function (see Figure 10.10).

$$F(u, v) = |\mathbf{R}(u, v) - \mathbf{R}_0|^2 \geq 0$$

A point $u_0,\ v_0$ where $F(u_0, v_0) = 0$ yields the solution.

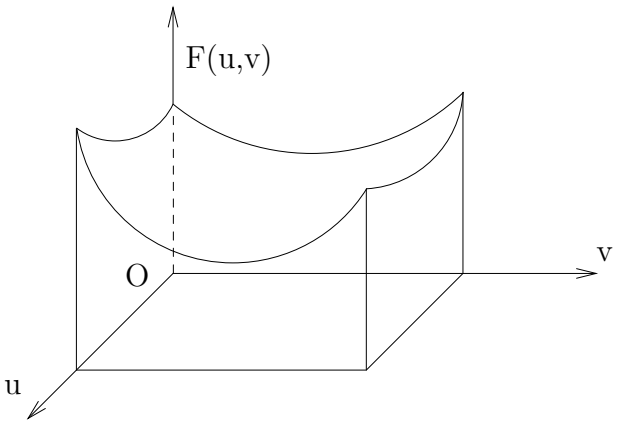


Figure 10.10: Distance function squared

In order to solve this minimization problem, we need to compute

- Minimum of all local minima of $F(u, v)$ ($F_u = F_v = 0$) in domain;

- Minimum of all local minima of boundary "curves" eg. $F(0,v)$ (i.e. $F_v = 0$);
- Values of $F(u,v)$ at corners, ie. $F(0,0)$, $F(0,1)$, $F(1,0)$, $F(1,1)$;

and then choose u_0, v_0 from above solutions where $F(u_0, v_0) = 0$.

The disadvantages of a minimization method are:

(a) Initial approximation is required;

(b) Possibility of divergence;

(c) No guarantee that all minima are located. (We need to enhance confidence by subdivision.)

(d) First and second derivativers of $F(u,v)$ are required.

Note: When $\mathbf{R} = \mathbf{R}(u,v)$ is a polynomial parametric surface patch, it is helpful to reformulate $F(u,v)$ to

$$F(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{m} w_{ij} B_{i,k}(u) B_{j,m}(n) \tag{10.7}$$

If $w_{ij} > 0$ for all $i,\ j$ then there is no solution. We could use w_{ij} to construct initial approximations for the various local minima to be computed by usual descent numerical methods. These initial approximation may be obtained by discrete sampling or subdivision.

Let $F(u,v)$ be expressed in the Bernstein basis, as in equation (10.7). Then, let us also express $F_u,\ F_v$ in the Bernstein basis:

$$\begin{aligned} F_u(u,v) &= \sum_{i=0}^{k-1} \sum_{j=0}^{m} A_{ij} B_{i,k-1}(u) B_{j,m}(v) = 0 \\ F_v(u,v) &= \sum_{i=0}^{k} \sum_{j=0}^{m-1} B_{ij} B_{i,k}(u) B_{j,m-1}(v) = 0 \end{aligned} \tag{10.8}$$

The equations $F_u(u,v) = 0$ and $F_v(u,v) = 0$ represent planar algebraic curves illustrated in Figure 10.11. Their intersection are the required extrema from which the minima can be selected using elementary calculus.

A geometrically motivated solution of the system 10.8 is possible using the convex hull property and subdivision to isolate an area where convex hulls intersect.

Taking $G(u,v) = F_u(u,v)$ and $n = k-1$ for example, we can write

$$w = G(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} A_{ij} B_{i,n}(u) B_{j,m}(v)$$

We can reformulate this "height" function into a parametric surface as follows:

$$\begin{aligned} \mathbf{w} = [u, v, G] &= \sum \sum \mathbf{L}_{ij} B_{i,m}(u) B_{j,n}(v) \\ \mathbf{L}_{ij} &= [\frac{i}{m},\ \frac{i}{n},\ A_{ij}] \end{aligned}$$

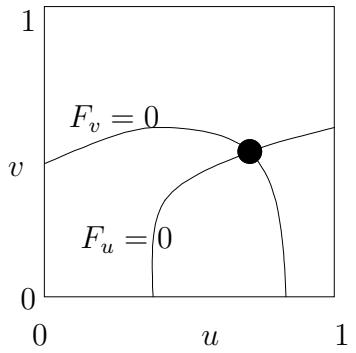


Figure 10.11: Intersection of algebraic curves.

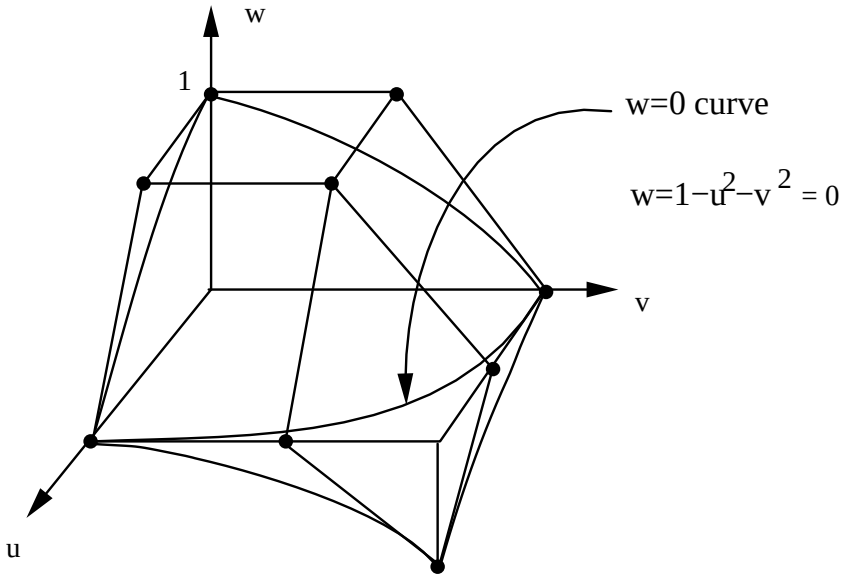


Figure 10.12: Control net

To solve $F_u = F_v = 0$, we find the projection of convex hulls of $\mathbf{w}(u, v)$ and of the corresponding surface for F_v on the coordinate planes $u = 0$, $v = 0$, then intersect them with the lines $w = 0$, see Figure 10.12 for an example. A more detailed description of this procedure is given in the next section where a nonlinear solver is described.

10.5.3 Point/Procedural surface intersection

A procedural surface may be an offset surface or a generalized cylinder surface or a blending surface. The typical solution method is minimization. In this case, no convex box assistance is possible in general, and we need a dense sampling for an initial approximation (which may be expensive) and no rigorous guarantees for the solution's reliability are generally available.

For certain classes of procedural surfaces such as offsets and evolutes of rational surfaces involving radicals of polynomials, it is possible to use the "auxiliary variable method" to reduce the point to surface intersection (or minimum distance) problem to a set of (a larger number of) nonlinear polynomial equations. Such systems can be solved robustly and efficiently using the nonlinear solver describe in the next section.

10.6 Curve/curve intersection

The curve types we will consider can be classified as follows:

- Rational polynomial parametric (RPP)
- Procedural parametric (PP)
- Implicit algebraic (IA)
- Implicit procedural (IP)

Procedural curves may be general offsets, evolutes, etc.

Curve to curve intersection cases are identified in table 10.1.

	RPP	PP	IA	IP
RPP	D1	D2	D3	D4
PP		D5	D6	D7
IA			D8	D9
IP				D10

Table 10.1: Curve to curve intersection cases

Conceptually, D3 (RPP/IA curve intersection) is the "simplest" of the above cases of intersection to describe and use for illustrating various general difficulties of intersection problems.

10.6.1 Case D3: RPP/IA curve intersection

We start with a planar RPP curve (which is conceptually the simplest case).

$$\begin{aligned} \mathbf{R}(t) &= [x(t), y(t)] = [\frac{X(t)}{w(t)}, \frac{Y(t)}{w(t)}] \\ &= \frac{\sum_{i=0}^{n} \mathbf{R}_i h_i B_{i,n}(t)}{\sum_{i=0}^{n} h_i B_{i,n}(t)} \qquad 0 \le t \le 1 \end{aligned}$$

where $h_i > 0$ are weights and $B_{i,n}(t)$ are Bernstein basis functions.

The implicit algebraic curve $f_m(x, y) = 0$ of total degree m is described by

$$f_m(x, y) = \sum_{i=0}^{m} \sum_{j=0}^{m-i} c_{ij} x^i y^j = 0$$

For convenience, we convert it to homogeneous form, by setting $x = \frac{X}{w}$, $y = \frac{Y}{w}$ and multiplying by w^m, which leads to

$$f_m(X, Y, w) = \sum_{i=0}^{m} \sum_{j=0}^{m-i} c_{ij} X^i Y^j w^{m-i-j} = 0$$

Every term of the above sum has a total degree m. Substituting $\mathbf{R}(t)$ into $f_m(X, Y, w)$ leads to a polynomial of degree up to mn

$$F(t) = 0$$

Therefore, now the problem of intersection is equivalent to finding the real roots of $F(t)$ in $0 \leq t \leq 1$. The most usual form of $F(t)$ is the power basis. The coefficients a_i can be evaluated symbolically by substitution and collection of terms. This can be readily done in a standard symbolic manipulation program (such as Macsyma, Reduce, Maple etc.). Such programs are oriented to processing rational numbers exactly.

Example:

Let the algebraic curve be an ellipse $\frac{x^2}{4} + y^2 - 1 = 0$, as illustrated in Figure 10.13. Let the parametric curve be a cubic Bézier curve with control points:

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -4 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \textit{ and } \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

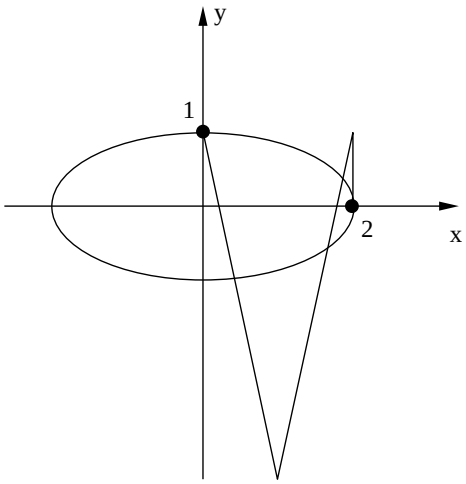


Figure 10.13: Ellipse and cubic Bézier curve polygon

Using Macsyma (or any similar symbolic manipulation program) and simplifying, we get (in exact arithmetic mode):

$$F(t) = 1025t^6 - 3840t^5 + 5514t^4 - 3728t^3 + 1149t^2 - 120t = 0$$

Next we find the (real) roots of $F(t)$ in $t \in [0, 1]$ using Macsyma's factoring capability over integers, which leads to

$$F(t) \equiv t(t-1)^2 G(t)$$

where

$$G(t) = [1025t^3 - 1790t^2 + 909t - 120]$$

Using a standard numerical solver for polynomials in floating point (such as NAG C02AEF routine), we obtain the following numbers as solutions of $G(t) = 0$ (reported with four decimal digits)

$$t = 0.9228,\ 0.61843,\ 0.2051$$

Alternately solving $F(t) = 0$ using the same routine leads to the following roots $t = t_R + it_I$

t_R	1	1	0.9228	0.6183	0.2051	0
t_I	-0.22×10^{-6}	0.22×10^{-6}	0	0	0	0

Notice the sensitivity to errors for the 6^{th} degree polynomial, especially for multiple roots as $t = 1$. In floating point arithmetic, such roots split into a number of roots (complex or real). Obviously, complex roots are not usable (as we require only the real intersection points). The consequence is lost roots, which implies an erroneous solution of the intersection problem.

An alternate basis for the representation of $F(t) = 0$ is the Bernstein basis, which has better stability of its real roots under perturbations of its coefficients than the power form. We will introduce the concept of condition numbers for polynomial roots later in this subsection.Such representation can be obtained without first converting to a power basis and using a symbolic manipulation program. It rather requires polynomial arithmetic involving products such as

$$B_{i,k}(t)B_{j,l}(t) = \left(\begin{array}{c} k \\ i \end{array} \right) \left(\begin{array}{c} l \\ j \end{array} \right) \left(\begin{array}{c} k+l \\ i+j \end{array} \right)^{-1} B_{i+j,k+l}(t)$$

In this way we obtain the representation:

$$F(t) = \sum_{i=0}^{mn} c_i B_{i,mn}(t) = 0$$

In the above example of the ellipse and cubic Bézier curve,

$$c_0 = 0,\ c_1 = -20,\ c_2 = 36.6,\ c_3 = -16.6,\ c_4 = 1.6,\ c_5 = 0,\ and\ c_6 = 0$$

Using the linear precision property

$$t = \sum_{i=0}^{mn} \frac{i}{mn} B_{i,mn}(t)$$

we can construct

$$\mathbf{f}(t) = [t, F(t)] = \sum_{i=0}^{mn} \left(\begin{array}{c} \frac{i}{mn} \\ c_i \end{array} \right) B_{i,mn}(t)$$

which is now a standard degree mn Bézier curve as in Figure 10.14

Notice that in our example $c_0 = 0$ which implies that $t = 0$ is a root. Also $c_5 = c_6 = 0$ implies that $t = 1$ is a double root. Note that

- If all $c_i > 0$ or $c_i < 0$ then there are not roots in $[0, 1]$.
- We can use subdivision (split in half) to identify subintervals of $[0, 1]$ where coefficients change sign once. Variation diminishing property implies one root in such areas. The Newton method can be used for fast convergence within such subintervals.

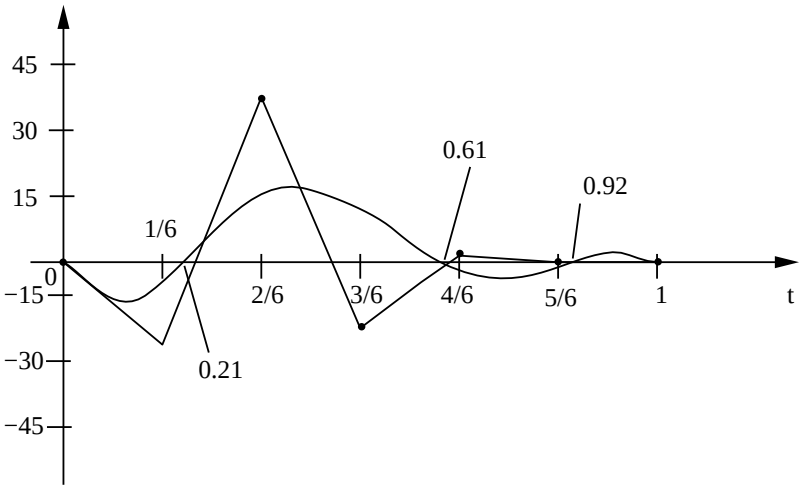


Figure 10.14: Intersection of a Bézier curve/straight line.

Another solution method is illustrated in Figure 10.15 for a quadratic curve (parabola). In this method the curve's convex hull is intersected with the axis $w = 0$ to give the $[A, B]$ interval as in Figure 10.15. The part of the $[0, 1]$ interval outside $[A, B]$ does not contain roots. By curve subdivision at A and B a smaller curve segment can be obtained in the Bézier form and the process can be continued. In this case, we use binary subdivision of AB when the rate of decrease of AB slows. See paper by Sherbrooke and Patrikalakis for details and also the next section on the nonlinear solver.

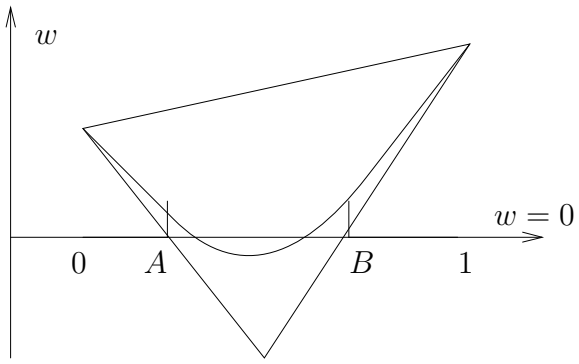


Figure 10.15: Subdivision at $A,\ B$.

Numerical condition of polynomials in Bernstein form

1. Condition numbers for polynomial roots
   Consider the displacement δx of a real root x_o of a polynomial in the basis $\{\phi_k(x)\}$:

$$P(x) = \sum_{k=0}^{n} \lambda_k \phi_k(x),$$

due to a perturbation $\delta\lambda_r$ in a single coefficient λ_r in this basis. Since $x_o + \delta x$ is a root of the perturbed polynomial, it satisfies:

$$P(x_o + \delta x) = -\delta\lambda_r \phi_r(x_o + \delta x).$$

Performing a Taylor series expansion about x_o on both sides of the above equation and noting that $P(x_o) = 0$, we obtain:

$$\sum_{k=1}^{n} \frac{(\delta x)^k}{k!} P^{(k)}(x_o) = -\delta\lambda_r \sum_{k=0}^{n} \frac{(\delta x)^k}{k!} \phi_r^{(k)}(x_o).$$

If x_o is a simple root of $P(x)$, then $P'(x_o) \neq 0$, and in the limit of infinitesimal perturbations the above equation gives:

$$\lim_{\delta\lambda_r \to 0} \frac{\delta x}{\delta\lambda_r/\lambda_r} = -\frac{\lambda_r \phi_r(x_o)}{P'(x_o)}.$$

We define

$$C = |\lambda_r \phi_r(x_o)/P'(x_o)|$$

the condition number of the root x_o with respect to the single coefficient λ_r.

If x_o is an m-fold root, $m \geq 2$, then we define a multiple-root condition number $C^{(m)}$ in the form

$$C^{(m)} = [\frac{m!}{|P^{(m)}(x_o)|} \sum_{r=0}^{n} |\lambda_r \phi_r(x_o)|]^{1/m}.$$

Theorem For an arbitrary polynomial $P(x)$ with a simple root $x_o \in [0, 1]$, let $C_p(x_o)$ and $C_b(x_o)$ denote the condition numbers of the root in the power and Bernstein bases on $[0, 1]$, respectively. Then $C_b(x_o) \leq C_p(x_o)$ for all $x_o \in [0, 1]$. In particular $C_b(0) = C_p(0) = 0$, while for $x_o \in (0, 1]$ we have the strict inequality $C_b(x_o) < C_p(x_o)$.

2. Example – Wilkinson polynomial

Consider the polynomial with the linear distribution of real roots $x_o = k/n, k = 1, 2, ..., n$, on the unit interval [0, 1] for $n = 20$:

$$P(x) = \prod_{k=1}^{20} (x - k/20).$$

The condition numbers for each root with respect to a perturbation in the single coefficient a_{19} are shown in Table 10.2. It is evident from Table 10.2 that the Bernstein form affords a dramatic inprovement in root condition numbers compared to the power form in this example, the condition number of the most unstable root being reduced by a factor of about 10^7.

We now perturb the single power coefficient $a_{19} = -\frac{21}{2}$ by an amount $-2^{-23}/20$. This corresponds to a fractional perturbation $\epsilon = 2^{-23}/210 \cong 5.7 \times 10^{-10}$. The roots of the

Table 10.2: Condition numbers for Wilkinson polynomial

k	$C_p(x_o)$	$C_b(x_o)$
1	2.100×10^1	3.413×10^0
2	4.389×10^3	1.453×10^2
3	3.028×10^5	2.335×10^3
4	1.030×10^7	2.030×10^4
5	2.059×10^8	1.111×10^5
6	2.667×10^9	4.153×10^5
7	2.409×10^{10}	1.115×10^6
8	1.566×10^{11}	2.215×10^6
9	7.570×10^{11}	3.321×10^6
10	2.775×10^{12}	3.797×10^6
11	7.822×10^{12}	3.321×10^6
12	1.707×10^{13}	2.215×10^6
13	2.888×10^{13}	1.115×10^6
14	3.777×10^{13}	4.153×10^5
15	3.777×10^{13}	1.111×10^5
16	2.833×10^{13}	2.030×10^4
17	1.541×10^{13}	2.335×10^3
18	5.742×10^{12}	1.453×10^2
19	1.310×10^{12}	3.413×10^0
20	1.378×10^{11}	0

pertured polynomials are shown in Table 10.3. For comparison, we now perturb the coefficient

$$c_{19} = -\frac{14849255421}{12800000000000000000}$$

of the Bernstein form by the same fractional amount, and obtain approximations to the roots of this perturbed polynomial, which are shown in Table 10.3.

Table 10.3: Roots of perturbed polynomials

k	power	Bernstein
1	0.05000000	0.0500000000
2	0.10000000	0.1000000000
3	0.15000000	0.1500000000
4	0.20000000	0.2000000000
5	0.25000000	0.2500000000
6	0.30000035	0.3000000000
7	0.34998486	0.3500000000
8	0.40036338	0.4000000000
9	0.44586251	0.4500000000
10	$0.50476331\pm$	0.5000000000
11	$0.03217504i$	0.5499999997
12	$0.58968169\pm$	0.6000000010
13	$0.08261649i$	0.6499999972
14	$0.69961791\pm$	0.7000000053
15	$0.12594150i$	0.7499999930
16	$0.83653687\pm$	0.8000000063
17	$0.14063124i$	0.8499999962
18	$0.97512197\pm$	0.9000000013
19	$0.09701652i$	0.9499999998
20	1.04234541	1.0000000000

3D-Space geometry: Case D3: RPP/IA (continued)

$$\mathbf{R}(t) = [x(t), y(t), z(t)] \quad \cap \quad f(\mathbf{R}) = g(\mathbf{R}) = 0$$

1. Substitute

$$f(\mathbf{R}(t)) \equiv F_1(t) = 0$$
$$g(\mathbf{R}(t)) \equiv F_2(t) = 0$$

2. Compute the resultant of $F_1(t), F_2(t)$ by eliminating t.

3. If $R(F_1(t), F_2(t)) \equiv 0$, then there is a common root between the two equations.

4. Use the inversion algorithm to find t.

10.6.2 Case D1: RPP/RPP Curve Intersection

$$\begin{cases} \mathbf{R}_1 = \mathbf{R}_1(t), & 0 \le t \le 1 \\ \mathbf{R}_2 = \mathbf{R}_2(v), & 0 \le v \le 1 \end{cases}$$

Setting $\mathbf{R}_1(t) = \mathbf{R}_2(v)$ leads to 3 nonlinear equations with 2 unknowns (overdetermined system). A possible approach is to choose 2 equations to solve for t, v, and then substitute the results into the third equation for verification. Alternatively the subdivision-based nonlinear solver described in the next section can directly solve such systems without the above 2 step approach.

Preprocessing idea in 3D:
Check bounding boxes for intersection. If there is such intersection, examine x, y projection.

Method 1 .

1. Implicitize, e.g., $x_2 = x_2(v)$, $y_2 = y_2(v)$, to get $f(x, y) = 0$.
2. Substitute $x_1(t), y_1(t)$ into $f(x, y)$ to get $F(t) = 0$ and solve it for real roots in $[0, 1]$.
3. Use the inversion algorithm: $v = \frac{\phi(x,y)}{q(x,y)}$

Method 2 Recursive subdivision and use of bounding boxes.

Some issues:

Method 1 is more efficient for $n \le 3$ but its robustness is unclear in the presence of ill-conditioned (tangent) intersections.

Method 2 is more efficient for $n \ge 4$ and its robustness with respect to missing roots can be guaranteed if the method is inplemented in rounded interval arithmetic (see next section). If two bounding boxes intersect, and they are of finite size, we can find roots using linear approximation.

In Figure 10.16(a), the boxes intersect, the linear approximation do not, and the curves intersect. Similar behavior is observed in (b) where polygon is used as the curve approximation.

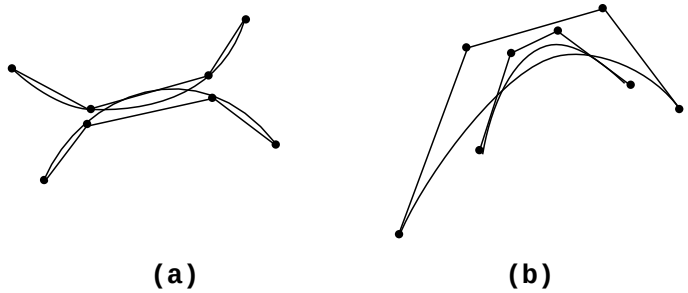


Figure 10.16: Ill-conditioned curve intersections.

Use hodograph (as in Figure 10.17) to find the range of tangent variation. Construct bounding angular sectors of hodographs of two curves and make vertices coincident. If the sectors do not intersect, then there is at most one root; otherwise, subdivide the two curves. For a precisely "tangent" root, this method would lead to infinite subdivision steps.

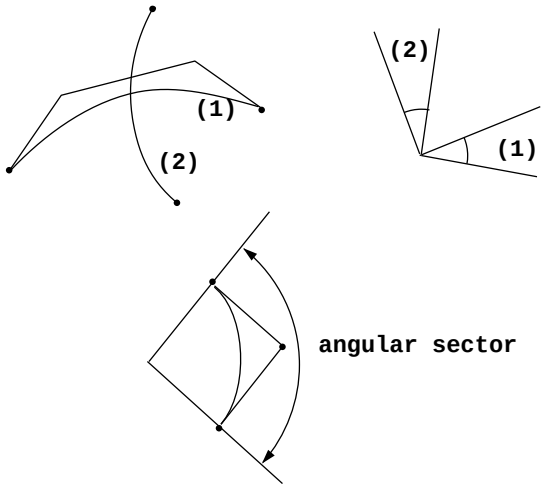


Figure 10.17: Hodograph concept.

10.6.3 Case D2/D5: RPP/PP and PP/PP Curve Intersections

3 equations with 2 unknowns

$$\begin{cases} \mathbf{R}_1 = \mathbf{R}_1(t), & 0 \le t \le 1 \\ \mathbf{R}_2 = \mathbf{R}_2(v), & 0 \le v \le 1 \end{cases}$$

Possible Approach
Minimize

$$F(t, v) = |\mathbf{R}(t) - \mathbf{R}(v)|^2, \quad 0 \le t, v \le 1$$

See comments on *Point/RPP surface* intersection.

• More difficult to compute derivatives of $F(t, v)$ exactly. May need numerical techniques (slower and usually more inaccurate).

10.6.4 Case D6: PP/IA Curve Intersection

$$\begin{cases} \mathbf{R}_1 = \mathbf{R}_1(t), & 0 \le t \le 1 \\ g(\mathbf{R}) = f(\mathbf{R}) = 0 \end{cases}$$

It could be reduced to *PP Curve/IA Surface* intersection and comparison of solutions for $g = 0$ and $f = 0$.

10.6.5 Case D8: IA/IA Curve Intersection

The planar case is of interest in processing trimmed patches.

$$\left.\begin{array}{l} f(u,v)=0 \\ g(u,v)=0 \end{array}\right\} (u,v) \in \text{ Parametric domain}$$

Method 1

Eliminate v to form the resultant $F(u)$, then solve $F(u)=0$ for u and use the inversion algorithm to get v.

• Example: Let us consider an ellipse and a circle

$$\begin{aligned} f &= \frac{x^2}{4}+y^2-1=0 \\ g &= (x-1)^2+y^2-1=0 \end{aligned}$$

as in Figure 10.18.

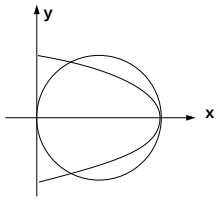


Figure 10.18: Ellipse and circle intersection

Let us eliminate y from these two equations. This leads to

$$3x^2-8x+4=0$$

which has as roots $x=2$ and $x=\frac{2}{3}$. Each of these leads to $y^2=0$ and $y^2=\frac{8}{9}$ respectively.

However there are possible numerical problems at the "tangent" solution $x=2,\ y=0$. Let us assume that due to error

$$x=2+\varepsilon$$

hence

$$y^2=-\varepsilon(1+\frac{\varepsilon}{4})<0$$

This implies that y is imaginary and that no real roots exist. This would have as a consequence missing an intersection solution, leading to a robustness problem.

Method 2

After tracing of $f(u,v)=0$ and $g(u,v)=0$, linear approximation is available. Find intersections of linear approximations and minimum distance points between them. Use this to drive a *Newton Method* on $f=g=0$ or a minimization of $F=f^2+g^2$.

10.7 Curve/surface intersection

Such intersections are classified as in Table 10.4. We will start with case E_3 which is quite representative.

	Surface Type			
Curve Type	RPP	PP	IA	IP
RPP	E1	E2	E3	E4
PP	E5	E6	E7	E8
IA	E9	E10	E11	E12
IP	E13	E14	E15	E16

Table 10.4: Curve/Surface Intersections

Curve to surface intersection involves the intersection of straight line to surface of all cases. Such intersection is useful for

- ray tracing
- point classification
- for procedural surface interrogation

10.7.1 Case E3: RPP Curve/IA Surface Intersection

Let us consider an (implicit) algebraic surface of total degree m

$$f_m(x,y,z) = \sum_{i=0}^{m} \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} x^i y^j z^k = 0$$

We first convert it to a degree homogeneous form by setting $x = X/W, y = Y/W, z = Z/W$ and multiply by W^m leading to

$$f_m(X,Y,Z,W) = \sum_{\substack{i,j,k,q=0 \\ i+j+k+q=m}}^{m} c_{ijk} X^i Y^j Z^k W^q = 0$$

Let us also consider a rational curve of degree n

$$\mathbf{R}(t) = [x(t), y(t), z(t)] = [\frac{X(t)}{W(t)}, \frac{Y(t)}{W(t)}, \frac{Z(t)}{W(t)}], \quad 0 \le t \le 1$$

We can easily substitute $\mathbf{R}(t)$ in $f_m(X,Y,Z,W)$ to obtain a polynomial equation $F(t) = 0$ of $degree \le mn$. We then find its (real) roots in $[0,1]$, see comments under section 10.6.1.

10.7.2 Case E1: RPP Curve/RPP Surface Intersection

3 nonlinear equations in 3 unknowns $t, u, v, \mathbf{R}(t) = \mathbf{Q}(u, v)$ where

$$\begin{aligned} \mathbf{R} &= \mathbf{R}(t), \quad 0 \le t \le 1 \\ \mathbf{Q} &= \mathbf{Q}(u, v), \quad 0 \le u, v \le 1 \end{aligned}$$

A Preprocessing step is to check bounding boxes for absence of intersection to eliminate easy cases.

Method 1 Implicitization of $\mathbf{Q}(u, v)$ for simple surface forms and reduction to Case *E3*.

Method 2 Recursive subdivision and use of bounding boxes. See also next section for nonlinear solver. Eventual use of a linear approximation technique for $\mathbf{R}$ and $\mathbf{Q}$ to obtain approximate solution, which is to be used to initiate a *Newton method* on $\mathbf{R}(t) - \mathbf{Q}(u, v) = 0$ or a *minimization method* on $F(t, u, v) = |\mathbf{R} - \mathbf{Q}|^2$. See section 10.6.2 for problem areas for $F_t = F_u = F_v = 0$ (3 equations.)

10.7.3 Case E2/E6: RPP/PP, PP/PP Curve/Surface Intersection

3 nonlinear equations in 3 unknowns $t, u, v, \mathbf{R}(t) = \mathbf{Q}(u, v)$ where

$$\begin{cases} \mathbf{R} = \mathbf{R}(t), \quad 0 \le t \le 1 \quad \cap \\ \mathbf{Q} = \mathbf{Q}(u, v), \quad 0 \le u, v \le 1 \end{cases}$$

Possible Approach

Minimize

$$F(t, u, v) = |\mathbf{R}(t) - \mathbf{Q}(u, v)|^2$$

in a cube $0 \le t, u, v \le 1$. See comments under *point-surface* intersection (e.g. examine vertices, edges, etc.)

10.7.4 Case E7: PP Curve/IA Surface Intersection

4 non-linear equations in 4 unknowns $t, \mathbf{R}$

$$\begin{cases} \mathbf{R} = \mathbf{R}(t), \quad 0 \le t \le 1 \\ f(\mathbf{R}) = 0 \end{cases}$$

Possible approach

Could use a *Newton method* initiated by a linear approximation of $\mathbf{R} = \mathbf{R}(t)$, which can be intersected more easily with $f(\mathbf{R}) = 0$ using the method of Case *E3*.

Problems

1. All roots?
2. Convergence?

3. Efficiency?

In case of convergence problems, embedding or continuation methods are normally helpful. For example,

1. Let
$$\mathbf{Q}(t) = \mathbf{Q}_0 + (\mathbf{Q}_1 - \mathbf{Q}_0)\frac{t-a}{b-a}, \quad t \in [a,b]$$
where $[a,b] \subset [0,1]$, be an approximation of $\mathbf{R}(t)$ within $[a,b]$ interval.
2. Compute $\mathbf{Q}(t) \cap f(\mathbf{R}) = 0$ in $t \in [a,b]$ using the method of Case *E3*.
3. Define sequence of problems
$$\mathbf{R}(t;\varepsilon) = \mathbf{Q}(t) + \varepsilon(\mathbf{R}(t) - \mathbf{Q}(t))$$
where $\varepsilon = \varepsilon_1, \varepsilon_2, \cdots, \varepsilon_N$ such that $0 = \varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_N = 1$.
4. Solve $\mathbf{R}(t;\varepsilon_i) \cap f(\mathbf{R}) = 0$ using as initial approximation the solution for $\varepsilon = \varepsilon_{i-1}$.

Such a method does not by itself provide initial approximations of all possible solutions; rather, it assists in the computation of a particular solution.

10.7.5 Case E11: IA Curve/IA Surface Intersection

3 equations in 3 unknowns
$$\underbrace{f(\mathbf{R}) = g(\mathbf{R})}_{curve} = \underbrace{h(\mathbf{R})}_{surface} = 0$$

Possible approaches

1. Elimination Methods
2. Newton Methods
3. Minimization Methods
$$F(\mathbf{R}) = f^2 + g^2 + h^2$$
Possible reformulation in a box Π
$$F(\mathbf{R}) = \sum_{i=0}^{M}\sum_{j=0}^{N}\sum_{k=0}^{Q} w_{ijk} B_{i,M}(\bar{x}) B_{j,N}(\bar{y}) B_{k,Q}(\bar{z})$$
$$(x,y,z) \in \Pi = [a_1,a_2] \times [b_1,b_2] \times [c_1,c_2]$$
and $\bar{x} = (x-a_1)/(a_2-a_1)$ and similarly for $\bar{y}$ and $\bar{z}$. If $w_{ijk} > 0$ for all i,j,k, there is no solution is Π.
Initial approximation: find $\min_{i,j,k} w_{ijk} < 0$ and start at $(\frac{i}{M}, \frac{j}{N}, \frac{k}{Q})$.
4. Approximate $f(\mathbf{R}) = g(\mathbf{R}) = 0$ curve with a linear spline, reduce to *E3* and refine using minimization.

10.7.6 IA Curve/RPP Surface Intersection

$$\begin{cases} \text{curve} & f(\mathbf{R}) = g(\mathbf{R}) = 0 \\ \text{surface} & \mathbf{R} = \mathbf{R}(u,v), \;\; 0 \le u, v \le 1 \end{cases}$$

Substitute $\mathbf{R} = \mathbf{R}(u,v)$ in $f(\mathbf{R}) = 0,\ g(\mathbf{R}) = 0$ to obtain two algebraic curves $f(u,v) = 0, g(u,v) = 0$, as in Figure 10.19. This formulation reduces to Case *D8* in Section 10.6.5: IA/IA curve intersection. Algebraic curves are treated under intersections of algebraic and RPP surfaces.

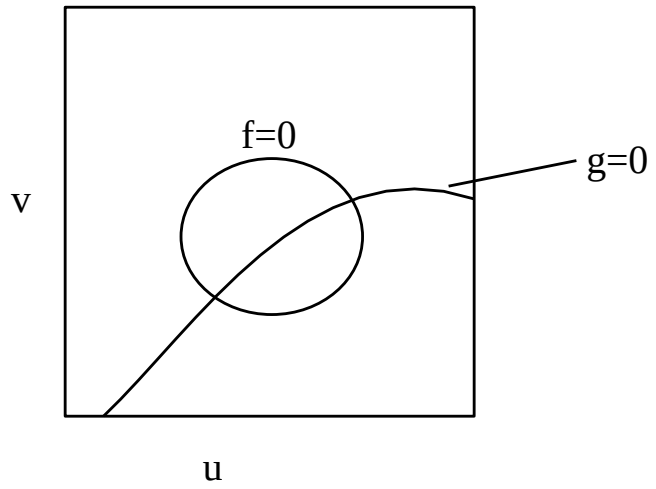


Figure 10.19: Intersection of two algebraic curves

10.8 Surface/Surface Intersections

The surface types we will consider can be classified as follows:

- Rational polynomial parametric (RPP)
- Procedural parametric (PP)
- Implicit algebraic (IA)
- Implicit procedural (IP)

Procedural surfaces may include general offsets, focal surfaces, etc.

Surface to surface intersection cases are identified in table 10.5.

	Surface Type			
Surface Type	RPP	PP	IA	IP
RPP	F1	F2	F3	F4
PP		F5	F6	F7
IA			F8	F9
IP				F10

Table 10.5: Classification of Surface/Surface Intersections

The solution of a surface/surface intersection problem may be empty, or include a curve (possibly made of several branches), a surface patch, or a point. Conceptually, F3 (RPP/IA surface intersection) is the "simplest" of the above cases of intersection to describe and use for illustrating general difficulties of surface intersection problems.

10.8.1 Case F3: RPP/IA Surface Intersection

We start with a RPP surface patch

$$\mathbf{R}(u,v) = [\frac{X(u,v)}{W(u,v)}, \frac{Y(u,v)}{W(u,v)}, \frac{Z(u,v)}{W(u,v)}] \tag{10.9}$$

where X, Y, Z are all of degree p in u, q in v, $(u,v) \in [0,1] \times [0,1]$.

Next we consider an implicit algebraic surface $f_m(x,y,z) = 0$ of total degree m described by

$$f_m(x,y,z) = \sum_{i=0}^{m} \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} x^i y^j z^k \tag{10.10}$$

Examples of such surfaces in practical use are low order surfaces such as planes (degree 1), the natural quadrics (cylinder, sphere, cone) (degree 2), and torii (degree 4). In fact in a survey of mechanical parts (mechanical elements), over 90% of all surfaces involved are of these types. It is also well known that all these surfaces also have a low degree rational polynomial parametric

representation, so that when two surfaces of the above types are intersected, the methods of this section may be used.

For convenience, we convert $f_m(x, y, z) = 0$ to its homogeneous form by setting

$$x = \frac{X}{W}, \quad y = \frac{Y}{W}, \quad z = \frac{Z}{W} \tag{10.11}$$

and multiplying by W^m, which leads to

$$f_m(X, Y, Z, W) = \sum_{\substack{i,j,k,q \geq 0 \\ i+j+k+q=m}} c_{ijk} X^i Y^j Z^k W^q = 0 \tag{10.12}$$

Consequently, the intersection problem

$$f_m(X, Y, Z, W) = 0 \tag{10.13}$$

$$X = X(u, v), \;\; Y = Y(u, v), \;\; Z = Z(u, v), \;\; W = W(u, v) \tag{10.14}$$

may be thought of as a nonlinear system of 5 equations in 6 unknowns. A reduction of the dimensionality of the system may be obtained by substituting equations 10.14 in equation 10.13, which since all functions involved are polynomial leads to an algebraic curve

$$f(u, v) = 0 \tag{10.15}$$

of degree $M = mp$ and $N = mq$ in u, v, respectively. Consequently, the problem of intersection reduces to the problem of tracing $f(u, v) = 0$ without omitting any special features of the curve, e.g., small loops, singularities, and accurately computing all its branches. This is a fundamental problem in *algebraic geometry* and much work has been done to understand its solution. In the context of algebraic geometry the coefficients of $f(u, v) = 0$ are integers. In the context of CAD and computer implementation, the coeffficients of $f_m = 0$, and $\mathbf{R} = \mathbf{R}(u, v)$ are floating point numbers. Consequently, if the above substitution is performed in floating point arithmetic the coefficients of $f(u, v) = 0$ involve error. To avoid such error, rational arithmetic may be used for robustness. These issues will also be discussed in the next section.

The algebraic curve

$$f(u, v) = \sum_{i=0}^{M} \sum_{j=0}^{N} a_{ij} u^i v^j = 0 \tag{10.16}$$

can be reformulated as

$$f(u, v) = \sum_{i=0}^{M} \sum_{j=0}^{N} w_{ij} B_{i,M}(u) B_{j,N}(v) = 0 \tag{10.17}$$

where $(u, v) \in [0, 1]^2$.

As an example consider a plane in homogeneous form

$$AX + BY + CZ + DW = 0 \tag{10.18}$$

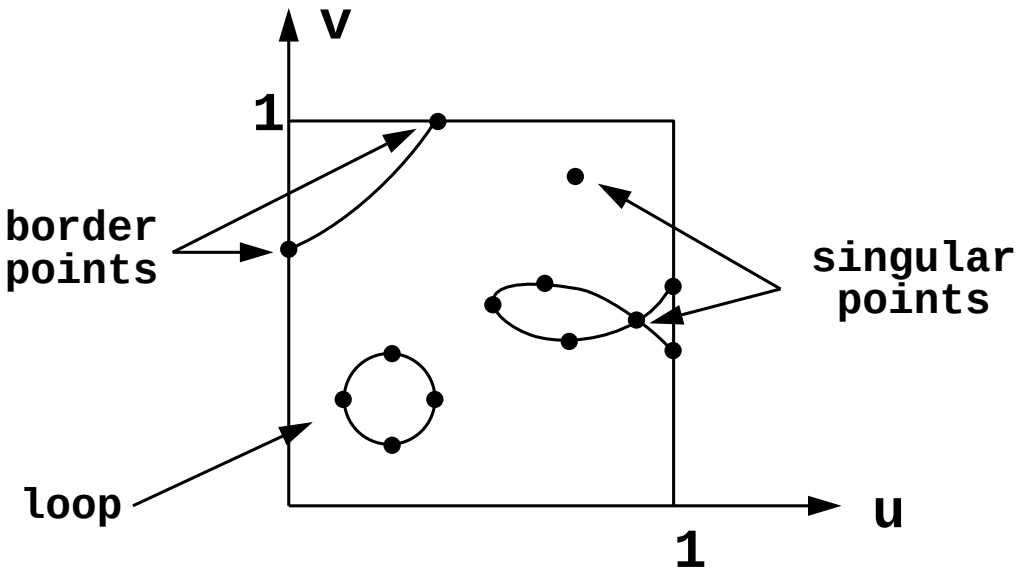


Figure 10.20: Parameter space of $\mathbf{R}(u,v)$ and resulting algebraic curve $f(u,v)=0$

and a rational Bézier patch of degree p in u, q in v

$$\mathbf{R}(u,v)=\frac{\sum_{i=0}^{p}\sum_{j=0}^{q}h_{ij}\mathbf{R}_{ij}B_{i,p}(u)B_{j,q}(v)}{\sum_{i=0}^{p}\sum_{j=0}^{q}h_{ij}B_{i,p}(u)B_{j,q}(v)} \tag{10.19}$$

where $\mathbf{R}_{ij}=[x_{ij},y_{ij},z_{ij}]$ and weights $h_{ij}\geq 0$.

The resulting algebraic curve is of the form of equation 10.17 with

$$w_{ij}=(Ax_{ij}+By_{ij}+Cz_{ij}+D)h_{ij} \tag{10.20}$$

In fact the power basis form of $f(u,v)=0$ need not be computed at all, if polynomial arithmetic for Bernstein polynomials is used.

The advantage of the Bernstein form is its numerical stability and convex hull property. If $w_{ij}>0$ or <0 for all i,j, there is no solution and the two surfaces do not intersect. More precisely, the (entire) algebraic surface $f_m(\mathbf{R})=0$ does not intersect the surface patch $\mathbf{R}=\mathbf{R}(u,v)$ for $(u,v)\in[0,1]^2$.

What happens when all $w_{ij}=0$ (or $\alpha_{ij}=0$)? Obviously, the two surfaces coincide in their entirety. A somewhat complex algebraic curve $f(u,v)=0$ is shown in Figure 10.20 involving various branches (from border to border), internal loops, and singularities, see also Figure 10.24.

Given a point on *every* branch (connected component) of an algebraic curve, tracing the curve using differential curve properties is an effective procedure (marching method).

Marching Method:

Let us expand $f(u,v)$ as a Taylor series

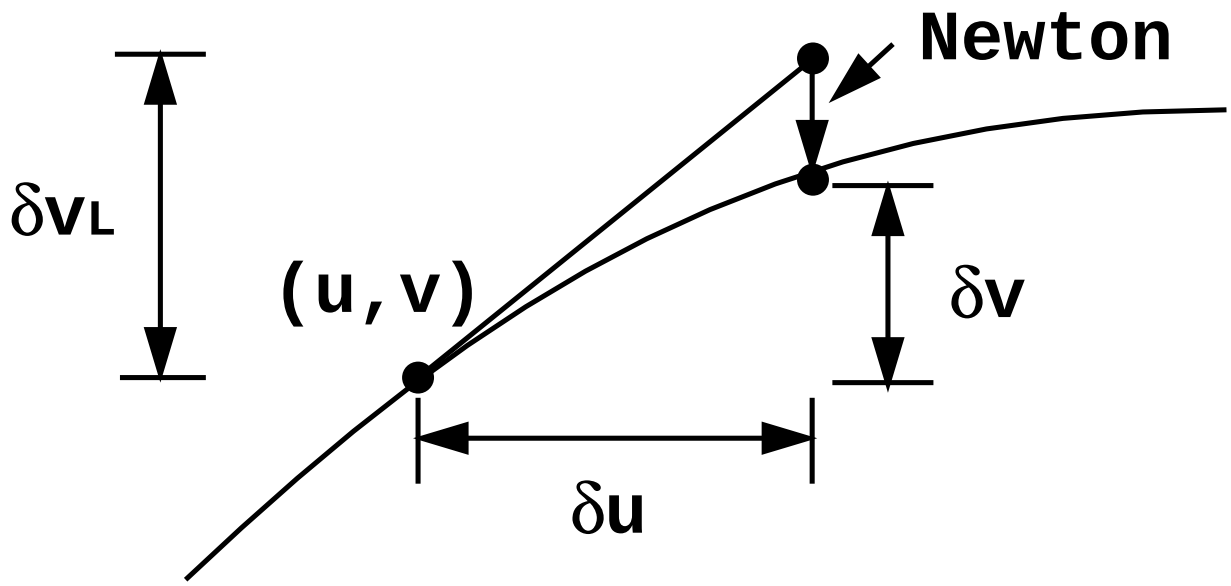


Figure 10.21: A zoomed view of an algebraic curve near a point (u, v)

$$\begin{aligned} f(u+\delta u, v+\delta v) &= f(u,v) + f_u\delta u + f_v\delta v \\ &\quad + \frac{1}{2}(f_{uu}\delta u^2 + 2f_{uv}\delta u\delta v + f_{vv}\delta v^2) + \cdots \end{aligned} \tag{10.21}$$

- To first order and if $f_u^2 + f_v^2 > 0$, in order to have $f(u, v) = 0$ and $f(u + \delta u, v + \delta v) = 0$,

$$f_u\delta u + f_v\delta v_L = 0 \Rightarrow \delta v_L = -\frac{f_u}{f_v}\delta u \tag{10.22}$$

  see also Figure 10.21.

- The Newton method on $f(u + \delta u, v) = 0$ with initial approximation $v_I = v + \delta v_L$ may be used to compute $v_F = v + \delta v$ with high accuracy and in an efficient manner.

- For "vertical" branches, ie. when $|f_v|$ is very small, we may use $\delta u_L = -\frac{f_v}{f_u}\delta v$.

To avoid these special stepping procedures the equation $f_u\delta u + f_v\delta v = 0$ may be converted to

$$f_u\dot{u} + f_v\dot{v} = 0 \tag{10.23}$$

where u, v are considered functions of a parameter t, $u = u(t)$, $v = v(t)$. This equation is satisfied if

$$\dot{u} = -\xi f_v(u,v) \tag{10.24}$$

$$\dot{v} = \xi f_u(u,v) \tag{10.25}$$

where ξ is an arbitrary constant. For example, ξ can be chosen to be equal to $[f_u^2 + f_v^2]^{-\frac{1}{2}}$, in order that t is an arc length parameter in the $[u, v] \in [0, 1]^2$ parameter space. This is a system

of two first order nonlinear differential equations which can be solved by the Runge-Kutta or other methods with adaptive step size.

Problems of Marching Methods

1. Starting points on all branches need to be provided in advance.
2. Marching through singularities ($f_u^2 + f_v^2 \simeq 0$) is problematic.
3. Step size selection is complex and too large a step size may lead to straying or looping, as in Figure 10.22.

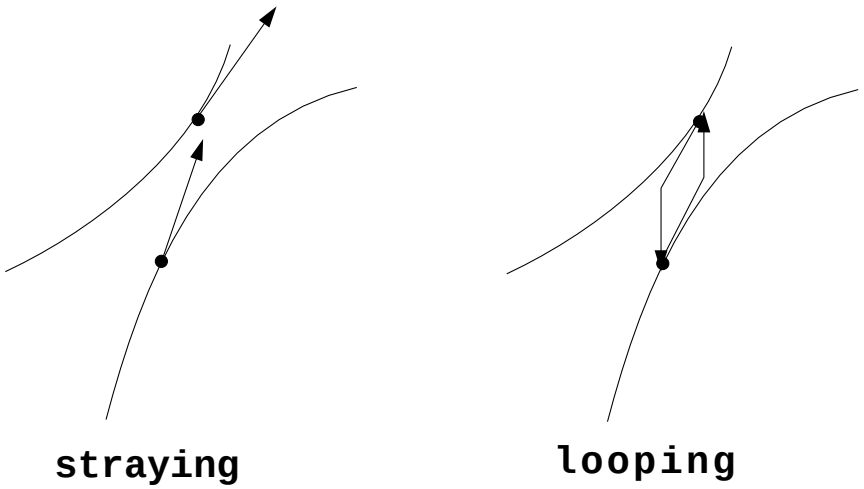


Figure 10.22: Step size problems in marching method

Computation of starting points

Starting points for tracing algebraic curves are certain "characteristic" points defined below:

1. Border points, ie. the intersections of $f(u, v) = 0$ with the boundary of the parameter space $[0, 1]^2$, e.g., $f(0, v) = 0,\ 0 \leq v \leq 1$.
2. Turning points $f = f_u = 0$ or $f = f_v = 0$, which are illustrated in Figure 10.23. If f is of degree (M, N) in (u, v), then f_u is of degree $(M-1, N)$ and f_v is of degree $(M, N-1)$. It can be seen that the total number of roots of two simultaneous bivariate polynomials of degree (m, n) and (p, q), respectively, is $mq + np$.Thus, there can be at most $2MN - M$ u-turning points and $2MN - N$ v-turning points.
3. Singular points $f = f_u = f_v = 0$. Notice $f_u = \nabla f \cdot \mathbf{R}_u, f_v = \nabla f \cdot \mathbf{R}_v$, and therefore $f_u = f_v = 0$ means that $\nabla f \parallel \mathbf{R}_u \times \mathbf{R}_v$ or that the normals of two surfaces are parallel and since $f(u, v) = 0$ at these points the two surfaces intersect tangentially. If f is of degree (M, N) in (u, v), f_u is of degree $(M-1, N)$ and f_v is of degree $(M, N-1)$, there can be at most $2MN - M - N + 1$ singular points.

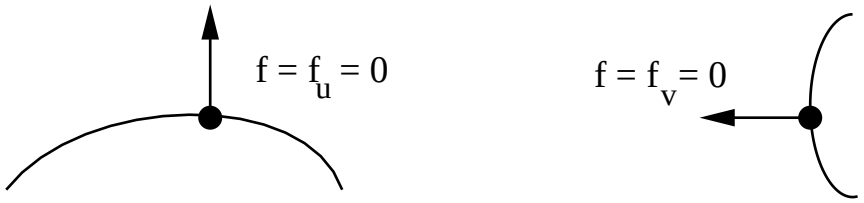


Figure 10.23: Turning points

From the above discussions we can get upper bounds for the maximum number of u-turning, v-turning and singular points, see table 10.6. These bounds refer to the maximum possible number of solutions (u, v) in the entire complex plane. It turns out that the number of such points in the real square $[0, 1]^2$ is much smaller, but still quite large. Consequently methods which focus on the possible solutions only in the $[0, 1]^2$ are advantageous. The subdivision method of Section 10 is one such method. Interval Newton methods are also potentially useful in this context.

S_1	S_2	algebraic curve $f(u,v)$ degree M, N	max number u-turning pts $2MN - M$	max number v-turning pts $2MN - N$	max number singular points $2MN - M - N + 1$
plane	biquadratic	2, 2	6	6	5
plane	bicubic	3, 3	15	15	13
quadric	biquadratic	4, 4	28	28	25
quadric	bicubic	6, 6	66	66	61
torus	biquadratic	8, 8	120	120	113
torus	bicubic	12, 12	276	276	265
biquadratic	biquadratic	16, 16	496	496	481
bicubic	biquadratic	36, 36	2556	2556	2521
bicubic	bicubic	54, 54	5778	5778	5725

Table 10.6: Number of turning and singular points in various cases

Analysis of singular points:

Let (u_0, v_0) satisfy $f(u_0, v_0) = 0$. We construct a straight line L

$$u = u_0 + \alpha t, \quad v = v_0 + \beta t \tag{10.26}$$

and we find its intersections with the algebraic curve $f(u, v) = 0$ by substitution as follows:

$$\begin{aligned} 0 &= f(u_0 + \alpha t, v_0 + \beta t) \\ &= f(u_0, v_0) + \alpha t f_u(u_0, v_0) + \beta t f_v(u_0, v_0) \\ &\quad + \frac{1}{2}(\alpha^2 t^2 f_{uu} + 2\alpha\beta t^2 f_{uv} + \beta^2 t^2 f_{vv})|_{(u=u_0, v=v_0)} \end{aligned}$$

$$
\begin{aligned}
&\quad +\text{h.o.t. (up to } t^\lambda, \lambda \text{ is finite)} \\
&= \quad t(\alpha f_u + \beta f_v) + \frac{t^2}{2}(\alpha^2 f_{uu} + 2\alpha\beta f_{uv} + \beta^2 f_{vv}) + \text{h.o.t.} \qquad (10.27)
\end{aligned}
$$

(1) Case A: $f_u^2 + f_v^2 > 0$, $f = 0$ at (u_0, v_0).

L is tangent to $f = 0$ at (u_0, v_0) if $t = 0$ is a double root. Thus

$$
\begin{aligned}
\alpha f_u + \beta f_v &= 0 \\
\alpha &= \mp f_v \\
\beta &= \pm f_u
\end{aligned}
$$

This is also a proof of fact that ∇f is $\perp$ to curve.

(2) Case B: $f_u = f_v = f = 0$ at (u_0, v_0).

$t = 0$ is a triple root if $f_{uu}^2 + f_{uv}^2 + f_{vv}^2 > 0$ and at least one of the 3rd derivatives is nonzero and

$$
\alpha^2 f_{uu} + 2\alpha\beta f_{uv} + \beta^2 f_{vv}|_{\substack{u=u_0 \\ v=v_0}} = 0 \qquad (10.28)
$$

We can solve this quadratic equation for $\frac{\alpha}{\beta}$ or $\frac{\beta}{\alpha}$ and there are three possiblities:

(1) 2 real distinct roots $\Rightarrow$ 2 distinct tangents (self-intersection)

(2) 1 real double root $\Rightarrow$ 1 tangent (cusp)

(3) 2 complex roots $\Rightarrow$ no real tangents (isolated point)

See also Figure 10.24 for illustration of the three cases.

Example 1: Let $f(u, v) = u^3 + u^2 + v^2 = 0$ so that

$$
f_u = u(3u + 2),\ f_v = 2v,\ f_{uu} = 6u + 2,\ f_{vv} = 2,\ f_{uv} = 0
$$

Turning points:

$$
\begin{aligned}
(1) \quad & f = f_u = 0, f_v \neq 0 \Rightarrow u = -\frac{2}{3} \\
& f = 0 \Rightarrow v^2 = -u^2(1 + u) < 0 \Rightarrow \text{ no real solution} \\
(2) \quad & f = f_v = 0, f_u \neq 0 \Rightarrow v = 0, u = -1
\end{aligned}
$$

Singular points:

$$
f = f_u = f_v = 0 \Rightarrow u = v = 0
$$

Tangents at $u = v = 0$ can be obtained from $2\alpha^2 + 2\beta^2 = 0$, $(\frac{\alpha}{\beta})^2 + 1 = 0$, which has no real solution, so $u = v = 0$ is an isolated point.

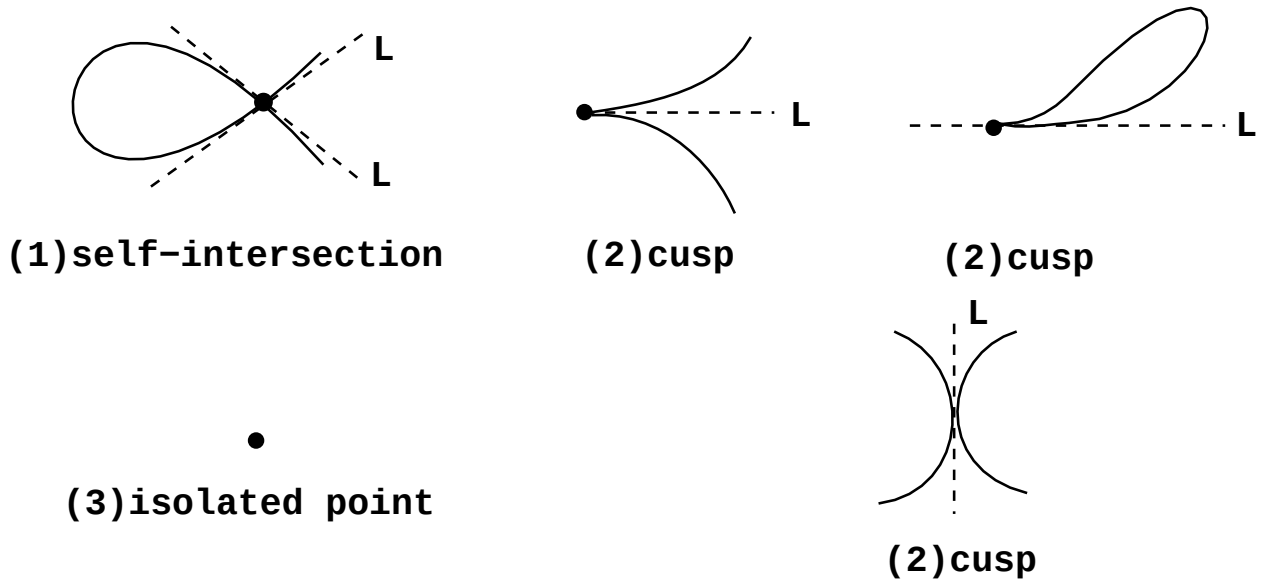


Figure 10.24: Singularities of planar algebraic curves

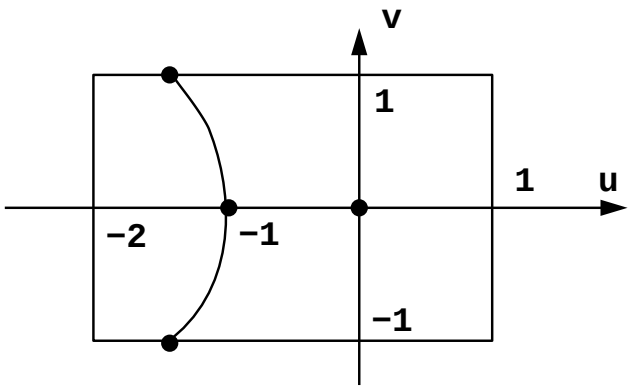


Figure 10.25: Example 1 algebraic curve with an isolated point

If the domain of interest is $[-2,1]\times[-1,1]$, border points are $(-1.465,\pm 1)$.

Example 2: Let $f(u,v)=v^2-u^3=0$. This curve has a cusp at $u=v=0$ with tangent $v=0$, see Figure 10.26.

Example 3: Let us consider the equation

$$f(u,v)=(u+1)u(u-1)(v+1)v(v-1)+\frac{1}{20}=0 \tag{10.29}$$

within the domain $[-2,2]^2$. This is a degree 6 algebraic curve illustrated in Figure 10.27. On every border line segment, there are three border points. The curve has no singular points, but involves two (internal) loops and six border-to-border branches. The algebraic curve $f(u,v)=0$ in this example has degrees $M=3,N=3$ in u and v. Consequently, using the previous formulas the number of u turning points, v turning points and singular

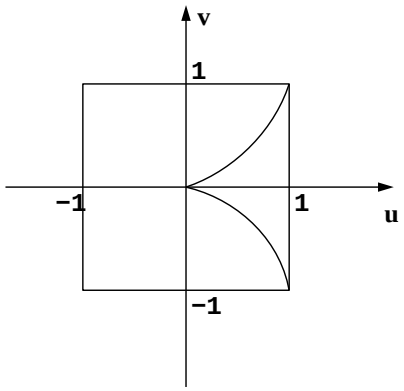


Figure 10.26: Example 2 algebraic curve with a cusp at $(u, v) = (0, 0)$

points (in the entire complex plane) is bounded by $2MN - M = 15$, $2MN - N = 15$, and $2MN - M - N + 1 = 13$. However, as we can be seen in Figure 10.27, these numbers overestimate the actual number of such points in the real square $[-2, 2]^2$.

Computing starting points for all branches

1. Border points: This involves solution of a polynomial, eg.

$$f(0, v) = \sum_{j=0}^{N} w_{0j} B_{j,N}(v) = 0$$

   A robust and efficient solution of this type of equation is addressed in Section 10.

2. Turning and singular point computation: Here we use the fact that

$$f_u(u, v) = M \sum_{i=0}^{M-1} \sum_{j=0}^{N} (w_{i+1,j} - w_{ij}) B_{i,M-1}(u) B_{j,N}(v) \tag{10.30}$$

$$f_v(u, v) = N \sum_{i=0}^{M} \sum_{j=0}^{N-1} (w_{i,j+1} - w_{ij}) B_{i,M}(u) B_{j,N-1}(v) \tag{10.31}$$

   Consequently, computing turning points ($f = f_u = 0$ and $f = f_v = 0$) is equivalent to solving a system of two nonlinear polynomial equations in two variables, and computing singularities $f = f_u = f_v = 0$ is equivalent to solving a system of three nonlinear polynomial equations in two variables. Robust and efficient solution of these systems of nonlinear polynomial equations is addressed in Section 10.

10.8.2 Case F1: RPP/RPP Surface Intersection

In this case we have two rational polynomial surface patches

$$\begin{cases} \mathbf{R}_1 = \mathbf{R}_1(u, v) \\ \mathbf{R}_2 = \mathbf{R}_2(s, t) \end{cases} \tag{10.32}$$

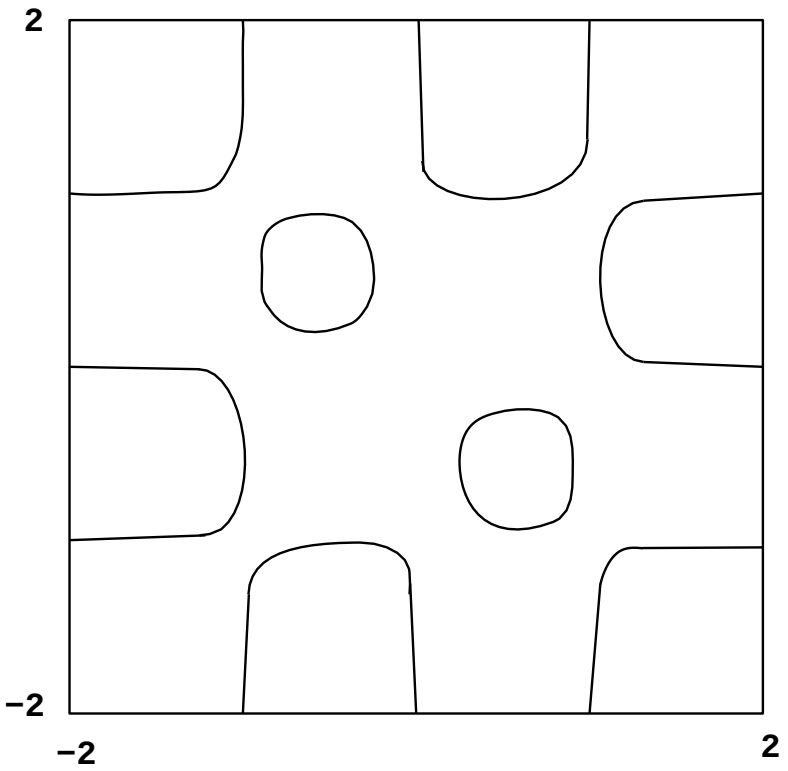


Figure 10.27: Example 3 algebraic curve

eg. two rational Bézier patches and by setting $\mathbf{R}_1 = \mathbf{R}_2$ we obtain three nonlinear polynomial equations for four unknowns u, v, s, t. This system can be solved by the nonlinear solver of Section 10. However, as the solutions are typically not isolated points but curves, such approach is inefficient. There are three major techniques for solving RPR/RPP surface intersections.

Method 1: Lattice method

In this method, one of the two surface patches is discretized to a grid of isoparameter curves at some fixed resolution. Each of the resulting curves is intersected with the other patch (using a technique as in Section 9.7). The resulting solution points are connected to form various curve branches based on empirical distance-based criteria. The method is typically inefficient (because it does not use the convex hull properties of RPP surface patches to their full extent) and generally not robust, leading to missing of small features of the intersection such as small loops, singularities. Also the connection of the points to form curves near constrictions and singularities is not robust as it is empirical.

Method 2: Subdivision method

Typically, subdivision methods involve the follwing steps, see Figure 10.28,

- Preprocessing by bounding boxes to eliminate subpatches that do not intersect.
- Subdivision (typically in four subpatches by multiple knot insertion at the mid point of parameter axes in NURBS patches)

- Approximation of the surface with triangular facets (either from the polyhedron or a grid on the subdivided surface)
- Intersections of triangular facets.

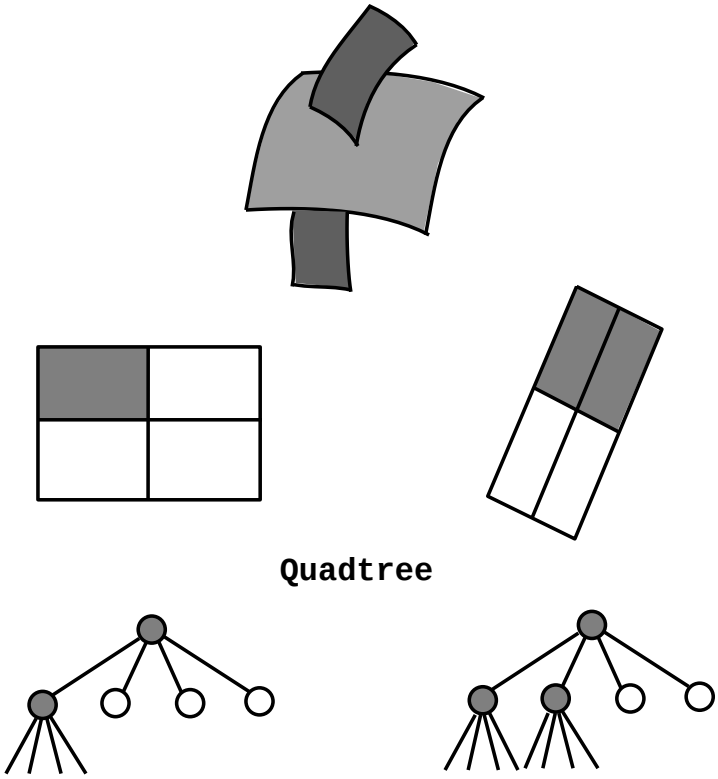


Figure 10.28: Subdivision method

Issues:

- Robust: resolution of loops and isolated points (under finite subdivision) is not guaranteed.
- Efficiency is typically better than lattice methods but usually inferior to marching methods.

Method 3: Marching along tangent to intersection curve

$$\mathbf{t} = (\mathbf{R}_{1u} \times \mathbf{R}_{1v}) \times (\mathbf{R}_{2s} \times \mathbf{R}_{2t}) \tag{10.33}$$

Issues:

- Finding starting points on all branches.
- Straying, looping, singularities

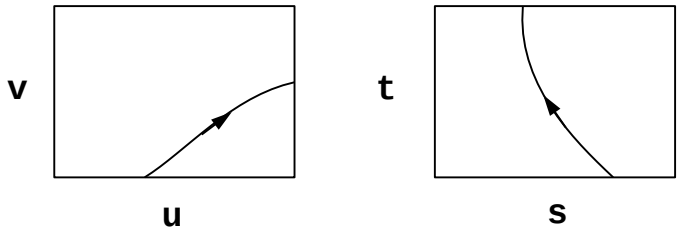


Figure 10.29: Parameter spaces of $\mathbf{R}_1(u,v)$ and $\mathbf{R}_2(s,t)$

Let us consider the intersection, $\mathbf{R}_1(u,v) = \mathbf{R}_2(s,t)$, eg. as illustrated in the parameter spaces of $\mathbf{R}_1$, $\mathbf{R}_2$ in Figure 10.29. A resulting intersection curve branch can be expressed as a parameter curve in terms of the parameter τ, as follows

$$u = u(\tau), \quad v = v(\tau), \quad s = s(\tau), \quad t = t(\tau) \tag{10.34}$$

The intersection curve tangent can be obtained as the tangent along the curve on the surfaces $\mathbf{R}_1(u(\tau), v(\tau))$ and $\mathbf{R}_2(s(\tau), t(\tau))$ using the chain rule of differentiation as follows

$$\dot{\mathbf{R}}_1 = \mathbf{R}_{1u}\dot{u} + \mathbf{R}_{1v}\dot{v} \tag{10.35}$$

$$\dot{\mathbf{R}}_2 = \mathbf{R}_{2s}\dot{s} + \mathbf{R}_{2t}\dot{t} \tag{10.36}$$

where $\dot{(\,)}$ denotes derivatives with respect to τ. However, $\dot{\mathbf{R}}_1 = \dot{\mathbf{R}}_2$ and this leads to

$$\mathbf{R}_{1u}\dot{u} + \mathbf{R}_{1v}\dot{v} = \mathbf{R}_{2s}\dot{s} + \mathbf{R}_{2t}\dot{t} \tag{10.37}$$

This is a system of three linear equations with four unknowns $\dot{u}, \dot{v}, \dot{s}, \dot{t}$, which can be solved to provide the following system of first order nonlinear ordinary differential equations (ODE):

$$\frac{ds}{d\tau} = \zeta \begin{vmatrix} x_t^{(2)} & x_u^{(1)} & x_v^{(1)} \\ y_t^{(2)} & y_u^{(1)} & y_v^{(1)} \\ z_t^{(2)} & z_u^{(1)} & z_v^{(1)} \end{vmatrix} = \zeta |A_1| \tag{10.38}$$

$$\frac{dt}{d\tau} = -\zeta \begin{vmatrix} x_s^{(2)} & x_u^{(1)} & x_v^{(1)} \\ y_s^{(2)} & y_u^{(1)} & y_v^{(1)} \\ z_s^{(2)} & z_u^{(1)} & z_v^{(1)} \end{vmatrix} = -\zeta |A_2| \tag{10.39}$$

$$\frac{du}{d\tau} = -\zeta \begin{vmatrix} x_s^{(2)} & x_t^{(2)} & x_v^{(1)} \\ y_s^{(2)} & y_t^{(2)} & y_v^{(1)} \\ z_s^{(2)} & z_t^{(2)} & z_v^{(1)} \end{vmatrix} = -\zeta |A_3| \tag{10.40}$$

$$\frac{dv}{d\tau} = \zeta \begin{vmatrix} x_s^{(2)} & x_t^{(2)} & x_u^{(1)} \\ y_s^{(2)} & y_t^{(2)} & y_u^{(1)} \\ z_s^{(2)} & z_t^{(2)} & z_u^{(1)} \end{vmatrix} = \zeta |A_4| \tag{10.41}$$

where

$$\mathbf{R}_1(u,v) = \left[x^{(1)}(u,v), y^{(1)}(u,v), z^{(1)}(u,v)\right], \tag{10.42}$$

$$\mathbf{R}_2(s,t) = \left[x^{(2)}(s,t), y^{(2)}(s,t), z^{(2)}(s,t)\right]. \tag{10.43}$$

Here ζ is an arbitrary non-zero factor that can be chosen to provide arc-length parametrization in the s, t domain as follows:

$$d\tau = \sqrt{ds^2 + dt^2} = \sqrt{\zeta^2(|A_1|^2 + |A_2|^2)}d\tau$$

hence

$$\zeta = \pm\frac{1}{\sqrt{|A_1|^2 + |A_2|^2}}.$$

This ODE system (10.38) to (10.41) can be solved using the Runge-Kutta method or a multistep method.

In order to compute approximate *starting points* for the above marching method we need to identify first the possible presence of (internal) loops. This can be done using the concept of collinear normal points.

Sederberg et al first recognized the importance of collinear normals in detecting the existence of closed intersection loops in intersection problems of two distinct parametric surface patches. Two points on two surfaces are said to be collinear normal points if their associated normal vectors lie on the same line.

TheoremIf two regular tangent plane continuous surface patches, $\mathbf{R}_1$ and $\mathbf{R}_2$, intersect in a closed loop, then there exists a line that is perpendicular to both $\mathbf{R}_1$ and $\mathbf{R}_2$ if the dot product of any two normal vectors (on the same patch or on different patches) is never zero. This means that the total range of normal directions for both patches considered simultaneously can not deviate more than 90°.

In other words, if the two surfaces do not contain a collinear normal (and do not turn more than 90°), then there are no closed loops of surface intersection. Denoting the two surfaces by $\mathbf{R}_1(u, v)$ and $\mathbf{R}_2(s, t)$, collinear normal points satisfy the following equations

$$\begin{aligned}&(\mathbf{R}_2 - \mathbf{R}_1) \cdot \mathbf{R}_{2s} = 0, \quad (\mathbf{R}_2 - \mathbf{R}_1) \cdot \mathbf{R}_{2t} = 0,\\ &(\mathbf{R}_{2s} \times \mathbf{R}_{2t}) \cdot \mathbf{R}_{1u} = 0, \quad (\mathbf{R}_{2s} \times \mathbf{R}_{2t}) \cdot \mathbf{R}_{1v} = 0.\end{aligned} \tag{10.44}$$

If $\mathbf{R}_1, \mathbf{R}_2$ are RPP surface patches, equation (10.44) form a system of four nonlinear polynomial equations that can be solved using the method of Section 10.

Now we split the patches in (at least) one parametric direction at these collinear normal points. Consequently, starting points are only border points on the boundaries of all subdomains created. Border points are intersections of the border curves of each patch with the other patch. Their computation involves a system of three nonlinear polynomial equations in three unknowns, which can be solved with the method of Section 10.

10.8.3 Case F8: IA/IA Surface Intersection

$$f(\mathbf{R}) = g(\mathbf{R}) = 0 \tag{10.45}$$

where f, g are polynomials. Here we have two equations and three unknowns $\mathbf{R}$.

One intersection method for low order f, g is to eliminate one variable (e.g. z) to find projection of intersection curves on plane of other two variables (e.g. x, y), then trace the algebraic curve and use the inversion algorithm to find z. A more complete analysis of this problem is beyond the scope of these notes.

Example 1: See Figure 10.30

$$\begin{aligned} f &= x^2 + y^2 + z^2 - 1 = 0 \\ g &= x^2 + (y - \frac{1}{2})^2 - \frac{1}{4} = 0 \end{aligned}$$

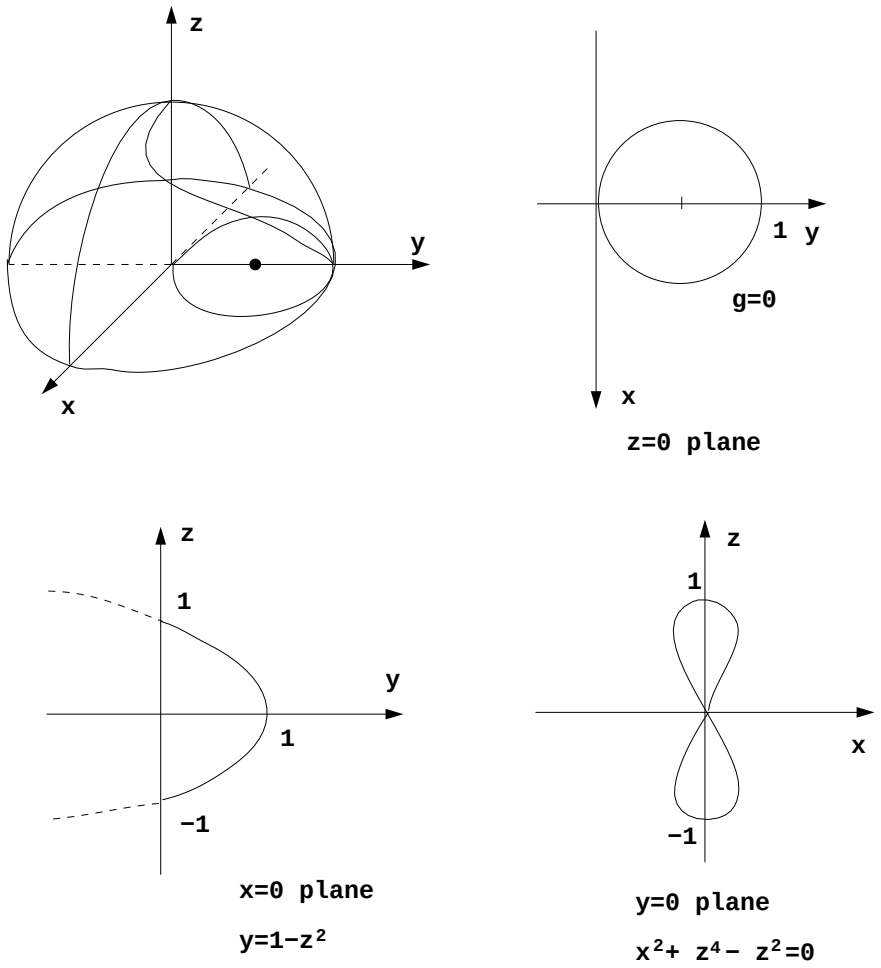


Figure 10.30: Intersection of two implicit quadrics (sphere and cylinder) from example 1

Appendix A

Tracing tangent intersection

Let an algebraic curve be such that

$$f(u,v) = f_u(u,v) = f_v(u,v) = 0 \tag{10.46}$$

on all points, and that

$$f_{uu}^2 + f_{vv}^2 + f_{uv}^2 = 0 \tag{10.47}$$

and at least one of the 3rd derivative is nonzero.

Then at a point (u_0, v_0) on $f(u_0, v_0) = 0$, the tangent $u = u_0 + \alpha t, v = v_0 + \beta t$ is defined by

$$\alpha^2 f_{uu} + 2\alpha\beta f_{uv} + \beta^2 f_{vv} = 0 \tag{10.48}$$

Now assume there is a single real tangent direction on all points of the curve. This occurs when

$$f_{uv}^2 - f_{uu} f_{vv} = 0 \tag{10.49}$$

The concept of *turning points* generalizes to $\alpha = 0$ or $\beta = 0$, because

$$\left.\begin{array}{l} \alpha = 0 \Rightarrow f = f_v = f_{vv} = 0 \\ \beta = 0 \Rightarrow f = f_u = f_{uu} = 0 \end{array}\right\} \text{ definition of turning points} \tag{10.50}$$

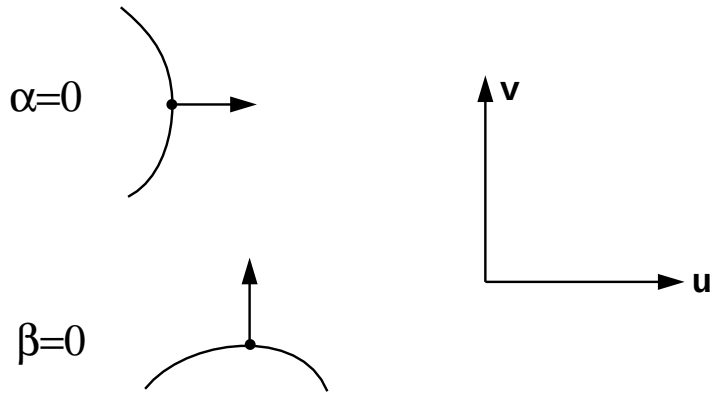


Figure 10.31: Turning points

From (10.49), $f_{uu} f_{vv} \geq 0$, so that

$$f_{uv} = \pm\sqrt{f_{uu} f_{vv}}$$

Hence, (10.48) becomes

$$\alpha^2 f_{uu} + \pm 2\sqrt{f_{uu} f_{vv}}\alpha\beta + \beta^2 f_{vv} = 0$$

Case 1 $f_{uu} \neq 0$

$$\begin{aligned} \Rightarrow \qquad & f_{uu}(\frac{\alpha}{\beta})^2 \pm 2\sqrt{f_{uu}f_{vv}}(\frac{\alpha}{\beta}) + f_{vv} = 0 \\ & \frac{\alpha}{\beta} = \pm\frac{\sqrt{f_{uu}f_{vv}}}{f_{uu}} = \pm\sqrt{\frac{f_{vv}}{f_{uu}}}sgn(f_{uu}) \end{aligned}$$

Case 2 $f_{vv} \neq 0$

$$\begin{aligned} \Rightarrow \qquad & f_{vv}(\frac{\beta}{\alpha})^2 \pm 2\sqrt{f_{uu}f_{vv}}(\frac{\beta}{\alpha}) + f_{uu} = 0 \\ & \frac{\beta}{\alpha} = \pm\frac{\sqrt{f_{uu}f_{vv}}}{f_{vv}} = \pm\sqrt{\frac{f_{uu}}{f_{vv}}}sgn(f_{vv}) \end{aligned}$$

We may choose

$$\begin{aligned} \alpha &= \sqrt{|f_{vv}|} = \dot{u} \\ \beta &= \sqrt{|f_{uu}|} = \dot{v} \end{aligned}$$

Normalize so that $\alpha^2 + \beta^2 = 1$

$$\left.\begin{aligned} \dot{u} &= K\sqrt{|f_{vv}|} \\ \dot{v} &= K\sqrt{|f_{uu}|} \end{aligned}\right\} \Rightarrow \dot{u}^2 + \dot{v}^2 = K^2(|f_{uu}| + |f_{vv}|) = 1$$

$$\begin{aligned} K &= \pm\frac{1}{\sqrt{|f_{uu}| + |f_{vv}|}} \\ \dot{u}(t) &= \pm\frac{\sqrt{|f_{vv}|}}{\sqrt{|f_{uu}| + |f_{vv}|}} \\ \dot{v}(t) &= \pm\frac{\sqrt{|f_{uu}|}}{\sqrt{|f_{uu}| + |f_{vv}|}} \end{aligned}$$

- Example: $f(u,v) = (u^2 + v^2 - 1)^2$

$$\left.\begin{aligned} f_u &= 4u(u^2 + v^2 - 1) \\ f_v &= 4v(u^2 + v^2 - 1) \end{aligned}\right\} \Rightarrow \text{ if } f = 0 \Rightarrow f_u = f_v = 0$$

$$\left.\begin{aligned} f_{uu} &= 4(3u^2 + v^2 - 1) \\ f_{vv} &= 4(3v^2 + u^2 - 1) \\ f_{uv} &= 8uv \end{aligned}\right\} \Rightarrow \text{ if } f = 0 \Rightarrow \left\{\begin{aligned} f_{uu} &= 8u^2 \\ f_{vv} &= 8v^2 \\ f_{uv}^2 &= f_{uu}f_{vv} \end{aligned}\right. \Rightarrow \left.\begin{aligned} \dot{u} &= \pm|v| \\ \dot{v} &= \pm|u| \end{aligned}\right\} \text{ on } f = 0$$

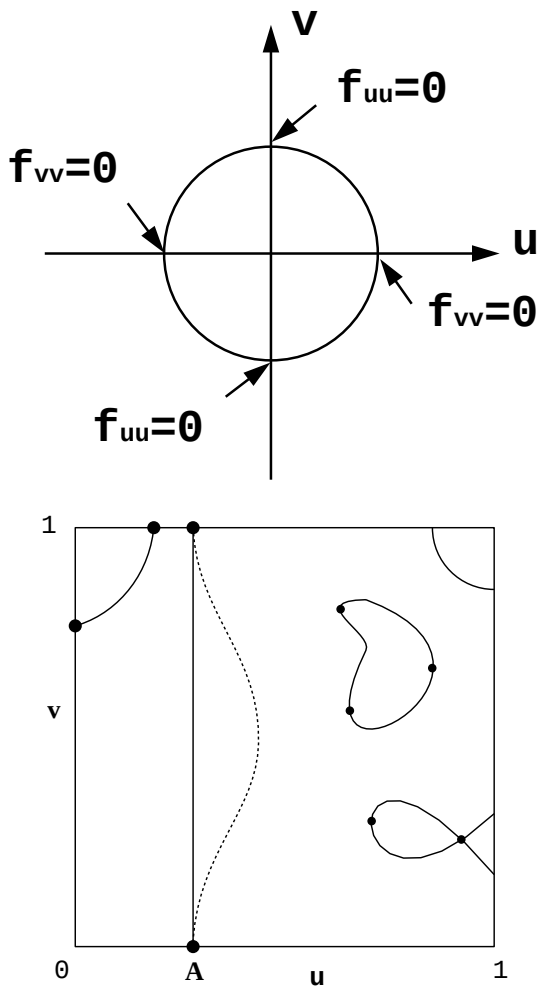


Figure 10.32: Tangent intersections

Case of infinite turning points

$$f(u,v) = (u-A)^k g(u,v) = 0$$

1. $\left.\begin{array}{l} f(u,0)=0 \\ f(u,1)=0 \end{array}\right\} \Rightarrow$ common solution $u = A$,

2. On which $f_v = (u-A)^k g_v(u,v) = 0$.

Try factoring $(u-A)$ sequentially until the v derivative of ratio is nonzero.

Condition (a) and (b) are necessary but not sufficient. However, subdivision would work in the limit, in determining $f = f_v = 0$.

Nonlinear solvers and robustness issues

10.9 Nonlinear Solvers

10.9.1 Motivation

As we have seen in earlier chapters, the geometric shape of curves and surfaces is usually represented by polynomial equations of various types (eg., implicit or parametric). As we have seen in Chapter 9, intersection problems reduce to solving systems of nonlinear equations which are usually polynomial if the geometries involved are polynomial.

Occassionally, the governing equations for general interrogation problems (intersections, curvature extrema, etc.) reduce to systems of nonlinear equations, involving also square roots of polynomials, which arise from normalization of the normal vector and analytical expressions of the curvatures.

Example 1 : Intersection between two planar implicit polynomial (algebraic) curves, eg. two circles, is shown in Figure 10.33. Their equations are

$$\begin{aligned} x^2 + y^2 - \frac{9}{16} &= 0 \\ (x-1)^2 + y^2 - \frac{1}{4} &= 0 \end{aligned}$$

which form a nonlinear polynomial system. The roots can be obtained by eliminating y, solving for x and then backsubstituting and solving for y, leading to

$$(x, y) = (\frac{21}{32}, \frac{\pm\sqrt{135}}{32}) \simeq (0.65625, \pm 0.36309)$$

Example 2 : Self-intersection of offset of a parabola is shown in Figure 10.34. Such intersections are needed in planning NC machining. Self-intersections of a parametric offset curve can be obtained by seeking pairs of distinct parameter values $s \neq t$ such that

$$\mathbf{r}(s) + d\mathbf{n}(s) = \mathbf{r}(t) + d\mathbf{n}(t) \tag{10.51}$$

where $\mathbf{r}(s)$ is a planar (progenitor) parametric curve, d is an offset distance and $\mathbf{n}(s)$ is a unit normal vector to $r(s)$ given by

$$\mathbf{n} = \mathbf{t} \times \mathbf{e}_z = \frac{(\dot{y}(s), -\dot{x}(s))}{\sqrt{\dot{x}^2(s) + \dot{y}^2(s)}} \tag{10.52}$$

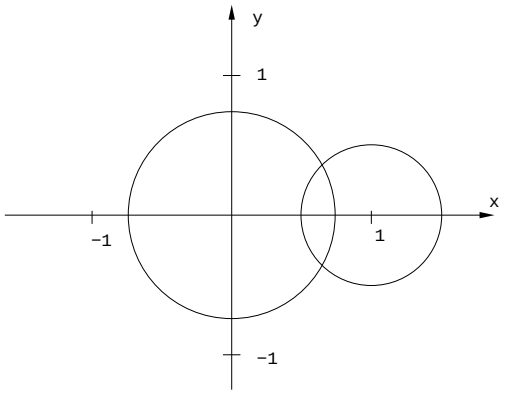


Figure 10.33: Intersection between two circles.

where $\mathbf{t}$ is the unit tangent vector and $\mathbf{e}_z$ the unit normal to the plane of the progenitor curve.

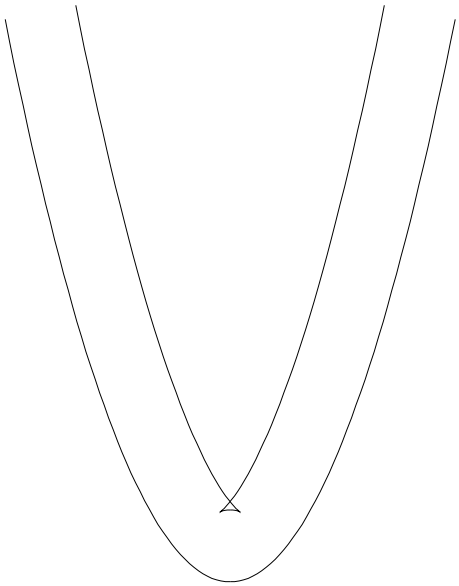


Figure 10.34: Self-intersection of an offset of a parabola.

Substituting equation (10.52) into equation (10.51) yields

$$\begin{aligned} x(s) + \frac{\dot{y}(s)d}{\sqrt{\dot{x}^2(s) + \dot{y}^2(s)}} &= x(t) + \frac{\dot{y}(t)d}{\sqrt{\dot{x}^2(t) + \dot{y}^2(t)}} \\ y(s) - \frac{\dot{x}(s)d}{\sqrt{\dot{x}^2(s) + \dot{y}^2(s)}} &= y(t) - \frac{\dot{x}(t)d}{\sqrt{\dot{x}^2(t) + \dot{y}^2(t)}} \end{aligned} \tag{10.53}$$

If $\mathbf{r}(s)$ is a polynomial function, equations 10.53 form a system of two nonlinear equations involving polynomials and radicals of polynomials. But if we set $\tau^2(s) = \dot{x}^2(s) + \dot{y}^2(s)$, we obtain 3 polynomial equations in which τ is the auxiliary variable.

10.10 Local Solution Methods

They are designed to compute roots based on initial approximations.

- Newton's Method in one variable [6]
  We want to find roots for $f(x) = 0$. Iteration formula is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2..... \tag{10.54}$$

  This is illustrated in Figure 10.35(a). A modified Newton's method for one variable is shown in Figure 10.35(b), where we take a fractional step as follows in order to reduce the possibility of divergence in Figure 10.35(b) of the full step method given by equation 10.54

$$x_{n+1} = x_n - \mu \frac{f(x_n)}{f'(x_n)} \tag{10.55}$$

  where $\mu = max[1, \frac{1}{2}, \frac{1}{4}, ...]$ such that $|f(x_{n+1})| < |f(x_n)|$.

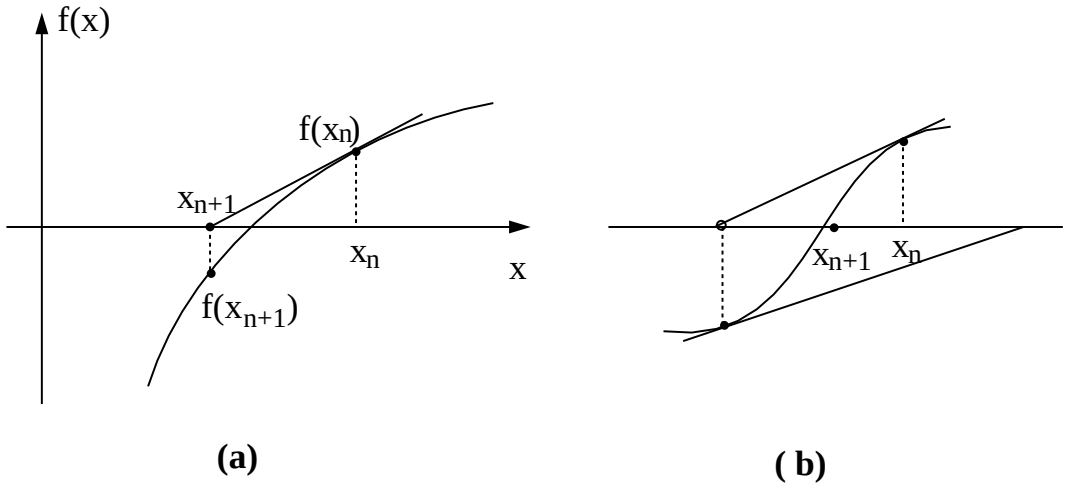


Figure 10.35: Newton's method for $f(x) = 0$

- Newton's Method for m equations in m unknowns
  We want to find the roots of the system

$$\mathbf{f}(\mathbf{r}) = 0 \tag{10.56}$$

  where $\mathbf{r} = [r_1, r_2, ..., r_m]^T$, and $\mathbf{f} = [f_1, f_2, ..., f_m]^T$.
  Iteration formula is given by

$$\mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} + \Delta\mathbf{r}^{(n)} \tag{10.57}$$

  where

$$\mathbf{J}(\mathbf{r}^{(n)}) \cdot \Delta\mathbf{r}^{(n)} = -\mathbf{f}(\mathbf{r}^{(n)}) \tag{10.58}$$

  and $\mathbf{J} = [\frac{\partial f_i}{\partial r_j}(\mathbf{r}^{(n)})]$ is the Jacobian matrix [6].

- Advantages
  Quadratic convergence.
  Straightforward to program.

- Disadvantages
  Needs good initial approximations for each root, otherwise it may diverge.
  Cannot provide full assurance that all roots have been found.

Example Two intersecting circles $x^2 + y^2 = \frac{9}{16}$, $(x-1)^2 + y^2 = \frac{1}{4}$. Let

$$f(x,y) = x^2 + y^2 - \frac{9}{16} = 0 \tag{10.59}$$

$$g(x,y) = (x-1)^2 + y^2 - \frac{1}{4} = 0 \tag{10.60}$$

Then, the Jacobian matrix is evaluated as follows:

$$[J] = 2 \begin{bmatrix} x & y \\ x-1 & y \end{bmatrix} \tag{10.61}$$

10.11 Classification of Global Solution Methods

Global solution methods are designed to compute all roots in some area of interest. In recent computational geometry related research, three classes of methods for the computation of solutions of nonlinear polynomial systems have been favored:

- *Algebraic techniques* [4] [5]
  Advantages: Theoretically elegant, and well-suited for implementation in symbolic mathematical systems. They determine all roots with arbitrary precision if the coefficients of the polynomials involved are integers or rational numbers.
  Disadvantages: Inefficient in memory and processing time requirements and therefore unattractive except for very low degree or dimensionality systems.

- *Homotopy methods* [9] [28]
  They tend to be numerically ill-conditioned for high degree polynomials and similarly inefficient because they are exhaustive.

- *Subdivision methods* [18] [10] [23] [27]
  Advantages: Subdivision methods are generally efficient and stable. Therefore, they are likely to be the most successful methods in practice. They can be combined with interval methods to numerically guarantee that certain subdomains do not contain solutions.
  Disadvantages: They are not as general as algebraic methods, since they are only capable of isolating zero-dimensional solutions.
  Furthermore, although the chances, that all roots have been found, increase as the resolution tolerance is lowered, there is no certainty that each root has been extracted. Subdivision methods typically do not provide a guarantee as to how many roots there may be in the remaining subdomains. However, if these subdomains are very small the

existence of a (single) root within these subdomains is a typical assumption.
Lastly, subdivision techniques provide no explicit information about root multiplicities without additional computation.

10.12 Subdivision Method (Projected Polyhedron Method)

We want to find roots of degree m polynomial equation $f(x) = c_o + c_1 x + c_2 x^2 + \cdots + c_m x^m = 0$ over the region $a \leq x \leq b$.

- The procedure is to first make the affine parameter transformation $x = a + t(b-a)$ such that $0 \leq t \leq 1$. The transition from the interval $a \leq x \leq b$ to the interval $0 \leq x \leq 1$ is an affine map, and the polynomials are invariant under affine parameter transformation.

- Now the polynomial equation is given by the monomial form

$$f(t) = \sum_{i=0}^{m} c_i^M t^i \tag{10.62}$$

- Change the basis from monomial to Bernstein.

$$f(t) = \sum_{i=0}^{m} c_i^B B_{i,m}(t) \tag{10.63}$$

$$\text{with } c_i^B = \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{m}{j}} c_j^M$$

where $B_{i,m}(t)$ is the ith Bernstein polynomial given by

$$B_{i,m}(t) = \binom{m}{i} t^i (1-t)^{m-i} \tag{10.64}$$

- Use *linear precision property* of the Bernstein polynomial

$$t = \sum_{i=0}^{m} \frac{i}{m} B_{i,m}(t) \tag{10.65}$$

In other words, the monomial t can be expressed as the weighted sum of Bernstein polynomials with coefficients evenly spaced in the interval $[0, 1]$.

- Create a graph of function $f(t)$. Then the graph will become a Bézier curve

$$\mathbf{f}(t) = \begin{pmatrix} t \\ f(t) \end{pmatrix} = \sum_{i=0}^{M} \begin{pmatrix} \frac{i}{m} \\ c_i^B \end{pmatrix} B_{i,m}(t) \tag{10.66}$$

where $(\frac{i}{m}, c_i^B)^T$ are control points.

- Now the problem of finding roots of the univariate polynomial has been transformed into a problem of finding the intersection of the Bézier curve with the parameter axis.

- Using the convex hull property, we discard the regions which do not contain the roots by applying the de Casteljau subdivision algorithm and find a sub-region of [0,1] which contain the root.
- If the sub-region is sufficiently small, we conclude that there is a root inside and return it.
- But when there are more than one root in the sub-region, the sub-region will not be reduced. In such case we split the region evenly.
- Scale the sub-region so that it will become [0,1] using affine parameter transformation and go back to *.

Example 1: Projected Polyhedron Method in one variable

Find the roots of $f(x) = -1.1x^2 + 1.4x - 0.2 = 0$ where $0 \leq x \leq 2$. The roots are approximately, 0.164 and 1.108.

- Affine parameter transformation
  Plug $x = 0 + (2-0)t = 2t$ into $f(x)$
  $f(t) = -4.4t^2 + 2.8t - 0.2 = 0$ where $0 \leq t \leq 1$

- Monomial to Bernstein basis
  We have $c_0^M = -0.2$, $c_1^M = 2.8$ and $c_2^M = -4.4$, thus using

$$c_i^B = \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{m}{j}} c_j^M \tag{10.67}$$

  we obtain $c_0^B = -0.2$, $c_1^B = 1.2$ and $c_2^B = -1.8$

- Use *linear precision property* of the Bernstein polynomial

$$t = \sum_{i=0}^{2} \frac{i}{2} B_{i,2}(t) \tag{10.68}$$

- Create a graph of function $f(t)$. Then the graph will become a Bézier curve

$$\mathbf{f}(t) = \begin{pmatrix} t \\ f(t) \end{pmatrix} = \sum_{i=0}^{2} \begin{pmatrix} \frac{i}{2} \\ c_i^B \end{pmatrix} B_{i,2}(t) \tag{10.69}$$

- Now the control points of the Bézier curve is as follows:
  (0, -0.2), (0.5, 1.2) and (1, -1.8).
- * Construct a convex hull of the Bézier curve. In this example it is a triangle whose vertices are the control points (0, -0.2), (0.5, 1.2) and (1, -1.8).

- The convex hull intersects the t-axis with $t = 0.0714$ and $t = 0.7$.
- Discard the regions $0 \leq t \leq 0.0714$ and $0.7 \leq t \leq 1$, which do not contain roots by appling the de Casteljau algorithm.
- Now we have a smaller convex hull which contains the roots. (Shaded triangular in Figure 10.36)
- Since there are two roots in the convex hull the sub-region will not reduce much. Therefore we split the region evenly and scale the two boxes so that it will become [0,1] using the affine parameter transformation and repeat the process until the two sub-regions become small enough.

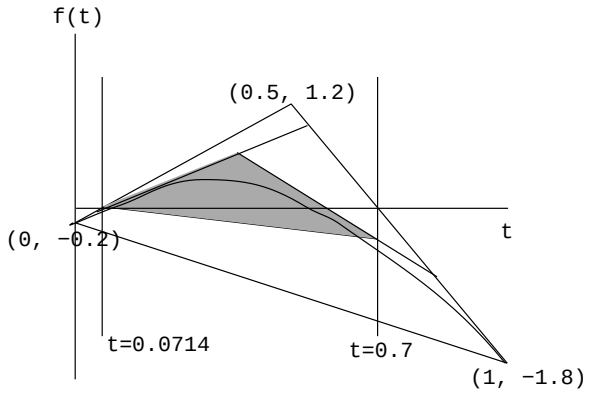


Figure 10.36: de Casteljau Algorithm applied to the quadratic Bézier curve

Example 2: Projected Polyhedron Method in two variables

Two intersecting circles $x^2 + y^2 = \frac{9}{16}$, $(x-1)^2 + y^2 = \frac{1}{4}$, see Figure 10.33. Let

$$f(x, y) = x^2 + y^2 - \frac{9}{16} = 0, \quad -1 \leq x,\ y \leq 1 \tag{10.70}$$

$$g(x, y) = (x-1)^2 + y^2 - \frac{1}{4} = 0, \quad -1 \leq x,\ y \leq 1 \tag{10.71}$$

- Affine parameter transformation

$$s = \frac{x - (-1)}{1 - (-1)} = \frac{x+1}{2} \tag{10.72}$$

$$t = \frac{y - (-1)}{1 - (-1)} = \frac{y+1}{2} \tag{10.73}$$

$$\tag{10.74}$$

Plug $x = 2s - 1$ and $y = 2t - 1$ into equations (10.70) (10.71), then we have

$$f(s, t) = 4s^2 - 4s + 4t^2 - 4t + \frac{23}{16} = 0 \tag{10.75}$$

$$g(s, t) = 4s^2 - 8s + 4t^2 - 4t + \frac{19}{4} = 0 \tag{10.76}$$

- Monomial to Bernstein basis

$$c_{ij}^B = \sum_{k=0}^{i} \sum_{l=0}^{j} \frac{\binom{i}{k}\binom{j}{l}}{\binom{m}{k}\binom{n}{l}} c_{kl}^M \tag{10.77}$$

We obtain

$$f(s,t) = \sum_{i=0}^{2} \sum_{j=0}^{2} f_{ij}^B B_{i,2}(s) B_{i,2}(t) \tag{10.78}$$

$$g(s,t) = \sum_{i=0}^{2} \sum_{j=0}^{2} g_{ij}^B B_{i,2}(s) B_{i,2}(t) \tag{10.79}$$

where

$$f_{00}^B = 1.4375, \quad f_{01}^B = -0.5625, \quad f_{02}^B = 1.4375$$

$$f_{10}^B = -0.5625, \quad f_{11}^B = -2.5625, \quad f_{12}^B = -0.5625$$

$$f_{20}^B = 1.4375, \quad f_{21}^B = -0.5625, \quad f_{22}^B = 1.4375$$

and

$$g_{00}^B = 4.75, \quad g_{01}^B = 2.75, \quad g_{02}^B = 4.75$$

$$g_{10}^B = 0.75, \quad g_{11}^B = -1.25, \quad g_{12}^B = 0.75$$

$$g_{20}^B = 0.75, \quad g_{21}^B = -1.25, \quad g_{22}^B = 0.75$$

- Use *linear precision property* of the Bernstein polynomial and create a graph.

$$\mathbf{f}(s,t) = \begin{pmatrix} s \\ t \\ f(s,t) \end{pmatrix} = \sum_{i=0}^{2} \begin{pmatrix} \frac{i}{2} \\ \frac{j}{2} \\ f_{ij}^B \end{pmatrix} B_{i,2}(s) B_{i,2}(t) \tag{10.80}$$

$$\mathbf{g}(s,t) = \begin{pmatrix} s \\ t \\ g(s,t) \end{pmatrix} = \sum_{i=0}^{2} \begin{pmatrix} \frac{i}{2} \\ \frac{j}{2} \\ g_{ij}^B \end{pmatrix} B_{i,2}(s) B_{i,2}(t) \tag{10.81}$$

The two Bézier surfaces are shown in Figure 10.37.

Now we will find the intersections of three surfaces, $\mathbf{f}(s,t)$, $\mathbf{g}(s,t)$ and xy-plane. Figure 10.38 shows the intersection between the plane and both Bézier surfaces. We can easily observe that the two intersection curves are the circles in Figure 10.33.

- Project the control points of $\mathbf{f}(s,t)$ and $\mathbf{g}(s,t)$ into xz and yz planes.
- For each xz and yz plane, construct the 2D convex hulls. Solid line corresponds to $\mathbf{f}(s,t)$ and the dotted line corresponds to $\mathbf{g}(s,t)$.

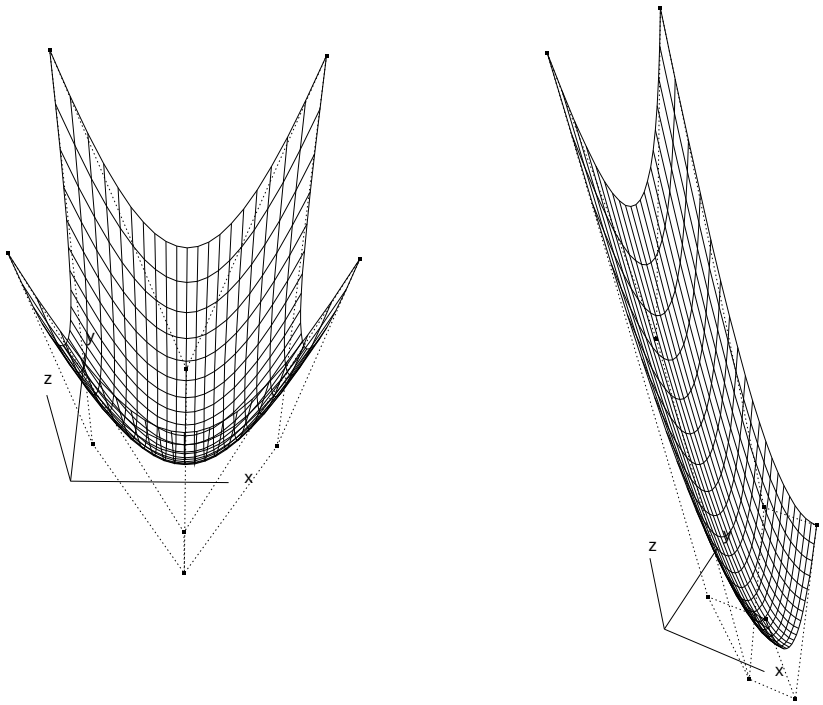


Figure 10.37: Bézier surfaces and their control points

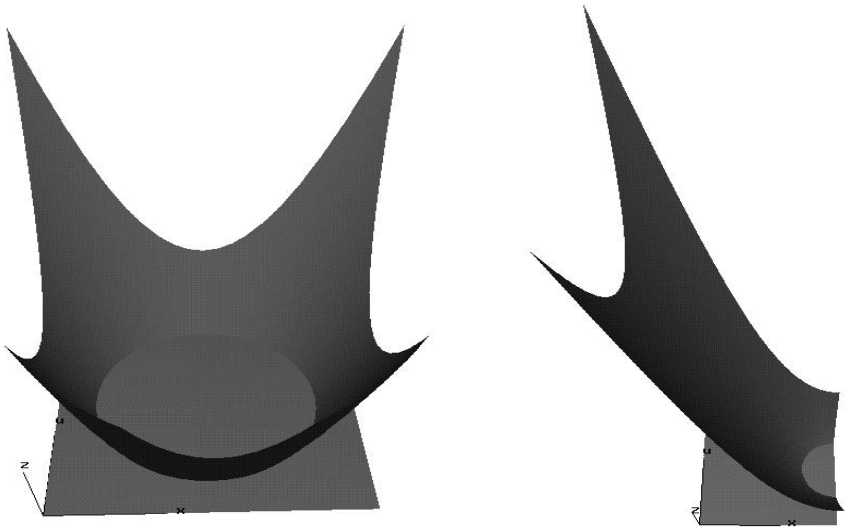


Figure 10.38: Bézier surfaces intersecting with xy-plane

Figure 10.39: Projections of Control Points

- Intersect the convex hull with horizontal axis. Because the polygon is convex, the intersection may be either a closed interval (which may degenerate to a point) or empty. If it is empty, then no root of the system exists within the given search box.
- Intersect the intervals with one another. Again, if the result is empty, no root exists within the given search box.

10.13 Interval Methods

10.13.1 Motivation

Nonlinear solvers operating in rational arithmetic are robust, but are generally time-consuming. On the other hand, nonlinear solvers operating in float point arithmetic are faster, but not robust. Interval methods solve the two problems, namely, nonlinear solvers operating in interval arithmetic are inexpensive compared to rational arithmetic, and they are robust.

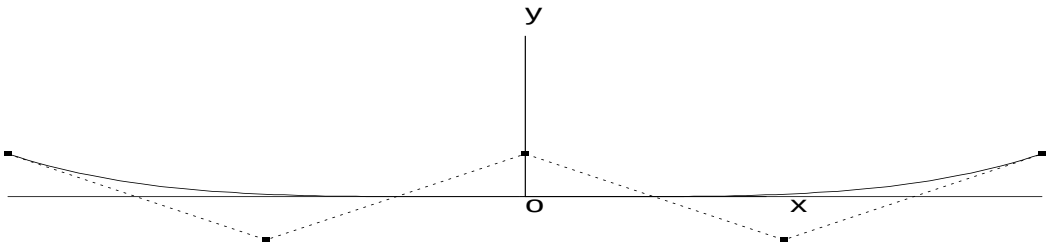


Figure 10.40: Curves $y = x^4$ and $y = 0$ contact tangentially at the origin.

Example
Suppose we have a degree four planar Bézier curve whose control points are given by

$$(-0.5, 0.0625), \quad (-0.25, -0.0625), \quad (0, 0.0625), \quad (0.25, -0.0625), \quad (0.5, 0.0625) \quad (10.82)$$

as shown in Figure 10.40. This Bézier curve is equivalent to the explicit curve $y = x^4$ ($-0.5 \leq x \leq 0.5$). Apparently the curve intersects with x-axis tangentially at $(x, y) = (0, 0)$. However, if the curve has been translated by $+1$ in the y direction and translated back to the original position by moving by $-\frac{1}{3}$ three times during a geometric processing session, the curve will generally not be the same as the original curve if floating point arithmetic (FPA) is used for the computation. For illustration, let us assume a decimal computer with a four-digit normalized mantissa, and the computer rounds off intelligently rather than truncating. Then the rational number $-\frac{1}{3}$ will be stored in the decimal computer as -0.3333×10^0 and after the processing the new control points will be

$$(-0.5, 0.0631), \quad (-0.25, -0.0624), \quad (0, 0.0631), \quad (0.25, -0.0624), \quad (0.5, 0.0631) \quad (10.83)$$

If we evaluate the curve at parameter value $t = 0.5$, we obtain (0, 0.00035) instead of (0,0). Therefore there exists a numerical gap which could later lead to inconsistency between topological structures and geometric equations. For example, if these new control points are used for computing intersections with the x-axis, the computer will return no solutions when the tolerance is smaller than 0.00035. The above problem illustrates the case when the error is created during the formulation of the governing equations by various algebraic transformations.

10.13.2 Definitions

An *interval* is a set of real numbers defined below [21]:

$$[a, b] = \{x | a \leq x \leq b\} \tag{10.84}$$

Two intervals $[a, b]$ and $[c, d]$ are said to be *equal* if

$$a = c \quad and \quad b = d \tag{10.85}$$

The *intersection* of two intervals is *empty* or $[a, b] \cap [c, d] = \emptyset$, if either

$$a > d \quad or \quad c > b \tag{10.86}$$

Otherwise,

$$[a, b] \cap [c, d] = [max(a, c), min(b, d)] \tag{10.87}$$

The *union* of the two intersecting intervals is

$$[a, b] \cup [c, d] = [min(a, c), max(b, d)]. \tag{10.88}$$

An *order* of intervals is defined by

$$[a, b] < [c, d] \;\; if \;\; and \;\; only \;\; if \;\; b < c. \tag{10.89}$$

The width of an interval $[a, b]$ is $b - a$.

The *absolute value* is

$$|[a, b]| = max(|a|, |b|). \tag{10.90}$$

Examples

$$[2, 4] \cap [3, 5] = [max(2, 3), min(4, 5)] = [3, 4]$$
$$[2, 4] \cup [3, 5] = [min(2, 3), max(4, 5)] = [2, 5]$$
$$|[-7, -2]| = max(|-7|, |-2|) = 7$$

10.13.3 Interval Arithmetic

$$[a,b] \circ [c,d] = \{x \circ y \mid x \in [a,b] \; and \; y \in [c,d]\}. \tag{10.91}$$

where $\circ$ represents an arithmetic operation $\circ \in \{+,-,\cdot,/\}$. Using the end points of the two intervals, we can rewrite equation (10.91) as follows

$$\begin{aligned}
&[a,b] + [c,d] = [a+c, b+d] \\
&[a,b] - [c,d] = [a-d, b-c] \\
&[a,b] \cdot [c,d] = [min(ac,ad,bc,bd), max(ac,ad,bc,bd)] \\
&[a,b]/[c,d] = [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)]
\end{aligned} \tag{10.92}$$

provided $0 \notin [c,d]$ in the division relation.

Examples

$$\begin{aligned}
&[2,4] + [3,5] = [2+3, 4+5] = [5,9] \\
&[2,4] - [3,5] = [2-5, 4-3] = [-3,1] \\
&[2,4] \cdot [3,5] = [min(2\cdot 3, 2\cdot 5, 4\cdot 3, 4\cdot 5), max(2\cdot 3, 2\cdot 5, 4\cdot 3, 4\cdot 5)] = [6,20] \\
&[2,4]/[3,5] = [min(2/3, 2/5, 4/3, 4/5), max(2/3, 2/5, 4/3, 4/5)] = [2/5, 4/3]
\end{aligned}$$

10.13.4 Algebraic Properties

Interval arithmetic is *commutative* and *associative*.

$$\begin{aligned}
&[a,b] + [c,d] = [c,d] + [a,b] \\
&[a,b] \cdot [c,d] = [c,d] \cdot [a,b] \\
&[a,b] + ([c,d] + [e,f]) = ([a,b] + [c,d]) + [e,f] \\
&[a,b] \cdot ([c,d] \cdot [e,f]) = ([a,b] \cdot [c,d]) \cdot [e,f]
\end{aligned}$$

But it is not *distributive*, however, it is *subdistributive*.

Examples

$$\begin{aligned}
&[1,2]([1,2] - [1,2]) = [1,2]([-1,1]) = [-2,2] \\
&[1,2][1,2] - [1,2][1,2] = [1,4] - [1,4] = [-3,3]
\end{aligned}$$

10.13.5 Rounded Interval Arithmetic and its Implementation

One has to keep in mind that any sequence of operations on a digital computer is essentially equivalent to a finite sequence of manipulations on a discrete grid of points. For example, a floating point number in the general form is given by [11]

$$(\pm).\mathbf{d_1 d_2} \cdots \mathbf{d_p} \cdot \mathbf{2^{exp}},$$

where $d_1 d_2 \cdots d_p$ is mantissa with $d_1 \neq 0$, and p is the number of significant digits, and exp is the integer exponent. If $p = 2$ and $-2 \leq exp \leq 3$ then a list of positive numbers in this system is

$$\begin{aligned}
.10 * 2^{-2} &= \frac{1}{8} & .11 * 2^{-2} &= \frac{3}{16} \\
.10 * 2^{-1} &= \frac{1}{4} & .11 * 2^{-1} &= \frac{3}{8} \\
.10 * 2^{0} &= \frac{1}{4} & .11 * 2^{0} &= \frac{3}{4} \\
.10 * 2^{1} &= 1 & .11 * 2^{1} &= \frac{3}{2} \\
.10 * 2^{2} &= 2 & .11 * 2^{2} &= 3 \\
.10 * 2^{3} &= 4 & .11 * 2^{3} &= 6
\end{aligned}$$

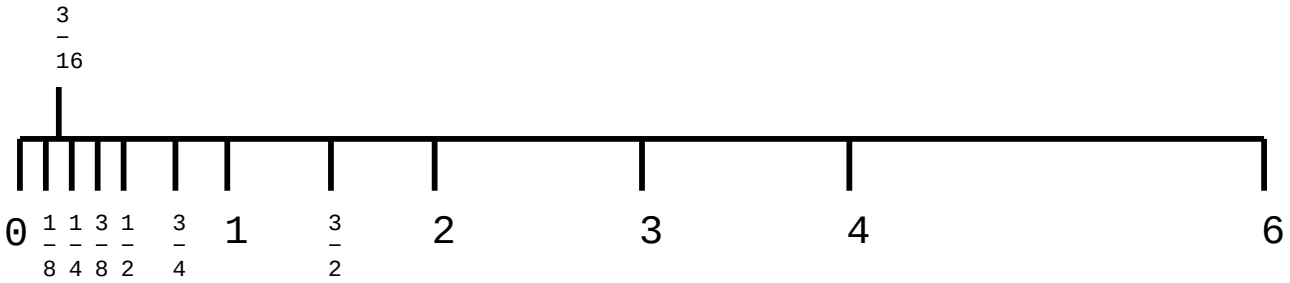


Figure 10.41: Nonnegative floating-point numbers on the interval [0,6], adapted from [11]

If floating–point arithmetic is used to evaluate these interval arithmetic equations there is no guarantee that the roundings of the bounds are performed conservatively. [1].

Most commercial processors implement floating–point arithmetic using the representation defined by ANSI/IEEE Std 754–1985, *Standard for Binary Floating–Point Arithmetic* [2]. This standard defines the binary representation of the floating–point number X in terms of a sign bit s, an integer exponent E, for $E_{min} \leq E \leq E_{max}$, and a p–bit significand (or mantissa) B, where:

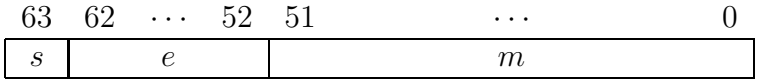
$$X = (-1)^s 2^E B \tag{10.93}$$

The significand B is a sequence of p bits $b_0 b_1 \cdots b_{p-1}$, where $b_i = 0$ or 1, with an implied binary point (analogous to a decimal point) between bits b_0 and b_1. Thus, the value of B is calculated as:

[1]This statement is true only for the default IEEE-754 rounding mode of *round towards nearest*

$$B \;=\; b_0.b_1b_2\cdots b_{p-1} \;=\; b_0 2^0 + \sum_{1}^{p-1} b_i 2^{-i} \tag{10.94}$$

For double precision arithmetic, the standard defines $p = 53$, $E_{min} = -1022$, and $E_{max} = 1023$. The number X is represented as a 64–bit quantity with sign bit s, an 11–bit biased exponent $e = E+1023$, and a 52–bit fractional mantissa m composed of the bit string $b_1b_2\cdots b_{52}$. Since the exponent can always be selected such that $b_0 = 1$ (and thus, $1 \le B < 2$), the value of b_0 is constant and it does not need to be stored in the binary representation.



The integer value of the 11-bit biased exponent e is calculated as:

$$e = e_0e_1\cdots e_{10} = \sum_{0}^{10} e_i 2^{10-i} \tag{10.95}$$

ULP (One Unit in the Last Place)

If x and x' are consecutive positive double-precision numbers, they differ by an amount ϵ called *ulp*. To calculate *ulp* it is necessary to extract the integer value of the exponent from the binary representation. Recall that the value of the significand B of a double precision number X is:

$$B = 1 + b_1 2^{-1} + b_2 2^{-2} + \cdots + b_{52} 2^{-52} \tag{10.96}$$

and that the double precision value $X = (-1)^s 2^E B$. The value of the least significant bit b_{52} is 2^{-52}. Thus, the value of *ulp* is $2^E 2^{-52} = 2^{E-52}$.

Rounded interval arithmetic [19, 20] ensures that the computed end points always contain the exact interval as follows:

$$\begin{array}{lcl}
[a,b] + [c,d] & \equiv & [a + c - \epsilon_\ell, b + d + \epsilon_u] \\
[a,b] - [c,d] & \equiv & [a - d - \epsilon_\ell, b - c + \epsilon_u] \\
[a,b] \;\cdot\; [c,d] & \equiv & [\min(a\cdot c, a\cdot d, b\cdot c, b\cdot d) - \epsilon_\ell, \max(a\cdot c, a\cdot d, b\cdot c, b\cdot d) + \epsilon_u] \\
[a,b] \;/\; [c,d] & \equiv & [\min(a/c, a/d, b/c, b/d) - \epsilon_\ell, \max(a/c, a/d, b/c, b/d) + \epsilon_u]
\end{array} \tag{10.97}$$

where ϵ_ℓ and ϵ_u are the units–in–last–place and are denoted by ulp_ℓ and ulp_u. When performing standard operations for interval numbers using RIA, the lower bound is extended to include its previous consecutive floating–point number, which is smaller than the lower bound by ulp_ℓ. Similarly, the upper bound is extended by ulp_u to include its next consecutive number. Thus, the width of the result is enlarged by $ulp_\ell + ulp_u$ and the result will be reliable in subsequent operations.

Example

```
main()
{
        double a = 1.5;

        Interval b = 1.5;


        double dresult = pow(a, 20.);

        Interval iresult = pow(b, 20.);
}
```

dresult 3325.2567300796509

iresult [3325.2567300796404,3325.2567300796613]

10.14 Interval Projected Polyhedron Algorithm

We extend the de Casteljau subdivision method to operate in rounded interval arithmetic in order to find all the roots of a polynomial system **robustly**. We illustrate the concept for a single polynomial equation.

$$\mathbf{(x - 0.1)(x - 0.6)(x - 0.7) = 0}$$

Floating Point Arithmetic

Iter	*Bounding Box (FPA)*	*Message*
1	[0,1]	
2	[0.0763636363636364, 0.856]	
3	[0.098187732239346, 0.770083868323999]	
4	[0.0999880766853688, 0.72387404781026]	Binary Sub.
5	[0.402239977003124, 0.704479954527487]	
6	[0.550441290533288, 0.700214508664293]	
7	[0.591018492648952, 0.700000534482207]	
8	[0.599458794784619, 0.700000000003332]	Binary Sub.
9	[0.649998841568898, 0.699999999999999]	No Root
10	[0.599997683137796, 0.649998841568898]	Root Found
11	[0.099999999478761, 0.402239977003124]	Root Found

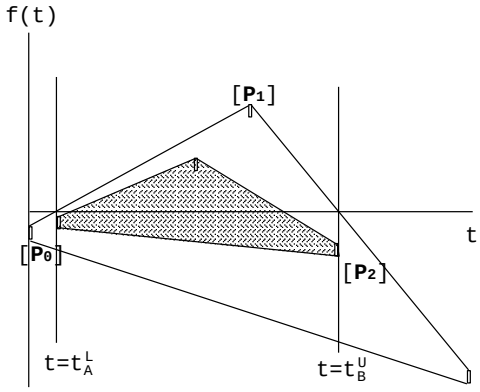


Figure 10.42: Interval Projected Polyhedron Method

Rounded Interval Arithmetic

Iter	*Bounding Box (RIA)*	*Message*
1	[0, 1]	
2	[0.076363636363635, 0.856000000000001]	
3	[0.0981877322393447, 0.770083868324001]	
4	[0.0999880766853675, 0.723874047810262]	Binary Sub.
5	[0.402239977003124, 0.704479954527489]	
6	[0.550441290533286, 0.700214508664294]	
7	[0.591018492648947, 0.700000534482208]	
8	[0.599458794784611, 0.700000000003333]	Binary Sub.
9	[0.649998841568894, 0.7]	Root Found
10	[0.599997683137788, 0.649998841568895]	Root Found
11	[0.0999999994787598, 0.402239977003124]	Root Found

Most applications mentioned above result in n nonlinear polynomial equations with n unknowns, referred to as balanced systems. However, there exist some problems such as

tangential and overlapping intersection or implicit curve/surface rendering consisting of n nonlinear polynomial equations with m unknowns, where n could be larger or smaller than m. When $n > m$ the system is called *overconstrained* and when $n < m$ it is called *underconstrained*. Here is an algorithm (the interval n-dimensional Projected-Polyhedron algorithm) such that it can effectively handle overconstrained problems [15].

Suppose we solve a system of nonlinear interval polynomial equations $[\mathbf{f}] = ([f_1], [f_2], \ldots, [f_n]) = \mathbf{0}$ over the box $S \in \mathbf{R}^m$ ($n > m$, $n = m$, $n < m$) where S is defined by

$$S = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_m, b_m]. \tag{10.98}$$

and each $[f_i]$ is an interval polynomial in m variables. That is, we wish to find all $\mathbf{u} \in S$ such that

$$[f_1](\mathbf{u}) = [f_2](\mathbf{u}) = \ldots = [f_n](\mathbf{u}) = \mathbf{0}. \tag{10.99}$$

By making the *affine parameter transformation* [13] $u_i = a_i + x_i(b_i - a_i)$ for $i = 1, \cdots m$, we simplify the problem to the problem of determining all $\mathbf{x} \in [0, 1]^m$ such that

$$[f_1](\mathbf{x}) = [f_2](\mathbf{x}) = \ldots = [f_n](\mathbf{x}) = \mathbf{0}. \tag{10.100}$$

Now furthermore suppose that each of the $[f_k]$ is polynomial in the independent parameters $x_1, x_2, \ldots, x_m$. Let $d_i^{(k)}$ denote the degree of $[f_k]$ in the variable x_i; then $[f_k]$ can be written in the multivariate Bernstein basis:

$$[f_k](\mathbf{x}) = \sum_{i_1=0}^{d_1^{(k)}} \sum_{i_2=0}^{d_2^{(k)}} \ldots \sum_{i_m=0}^{d_m^{(k)}} [w_{i_1 i_2 \ldots i_m}^{(k)}] B_{i_1, d_1^{(k)}}(x_1) B_{i_2, d_2^{(k)}}(x_2) \ldots B_{i_n, d_m^{(k)}}(x_m). \tag{10.101}$$

where $B_{i,m}$ is the ith Bernstein polynomial. The notation in (10.101) may be simplified by letting $I = (i_1, i_2, \ldots i_m)$, $D^{(k)} = (d_1^{(k)}, d_2^{(k)}, \ldots, d_m^{(k)})$ and writing (10.101) in the equivalent form [27]

$$[f_k](\mathbf{x}) = \sum_{I}^{D^{(k)}} [w_I^{(k)}] B_{I, D^{(k)}}(\mathbf{x}). \tag{10.102}$$

Here we have merely rewritten the product of Bernstein polynomials as a single *Bernstein multinomial* $B_{I,D^{(k)}}(\mathbf{x})$. Bernstein polynomials have a useful identity called *linear precision property* [13], in other words, the monomial t can be expressed as the weighted sum of Bernstein polynomials with coefficients evenly spaced in the interval $[0, 1]$. Using this property, we can rewrite equations (10.102) as follows:

$$[\mathbf{F}_k](\mathbf{x}) = \sum_{I}^{D^{(k)}} [\mathbf{v}_I^{(k)}] B_{I, D^{(k)}}(\mathbf{x}) \tag{10.103}$$

where

$$[\mathbf{v}_I^{(k)}] = ([\frac{i_1}{d_1^{(k)}}], [\frac{i_2}{d_2^{(k)}}], \ldots, [\frac{i_m}{d_m^{(k)}}], [w_I^{(k)}]). \tag{10.104}$$

These $[\mathbf{v}_I^{(k)}]$ are called the *interval control points* of $[\mathbf{F}_k]$. Now the algebraic problem of finding roots of systems of polynomials has been transformed to the geometric problem involving the

intersection of the hypersurfaces. Because the problem is now phrased geometrically, we can use the *convex hull property* of the multivariate Bernstein basis to bound the set of roots. We can structure a root-finding algorithm as follows:

1. Start with an initial box of search.

2. Scale the box and, as we did in converting between equations (10.99) and (10.100), perform an appropriate affine parameter transformation to the functions f_k, so that the box becomes $[0,1]^m$. However, keep track of the scaling relationship between this box and the initial box of search. This transformation can be performed with multivariate De Casteljau subdivision.

3. Using the convex hull property, find a sub-box of $[0,1]^m$ which contains all the roots. The essential idea behind the box generation scheme in this algorithm is to transform a complicated $m+1$-dimensional problem into a series of m two-dimensional problems. Suppose $\mathbf{R}^{m+1}$ can be coordinatized with the $x_1, x_2, \ldots, x_{m+1}$ axes; we can then employ these steps:

   (a) Project the $[\mathbf{v}_I^{(k)}]$ of all of the $[\mathbf{F}_k]$ into m different coordinate planes; specifically, the (x_1, x_{m+1})-plane, the (x_2, x_{m+1})-plane, and so on, up to the (x_m, x_{m+1}) plane.

   (b) In each one of these planes,

      i. Construct n two-dimensional convex hulls. The first is the convex hull of the projected control points of $[\mathbf{F}_1]$, the second is from $[\mathbf{F}_2]$ and so on.

      ii. Intersect each convex hull with the horizontal axis (that is, $x_{m+1} = 0$). Because the polygon is convex, the intersection may be either a closed interval (which may degenerate to a point) or empty. If it is empty, then no root of the system exists within the given search box.

      iii. Intersect the intervals with one another. Again, if the result is empty, no root exists within the given search box.

   (c) Construct an m-dimensional box by taking the Cartesian product of each one of these intervals in order. In other words, the x_1 side of the box is the interval resulting from the intersection in the (x_1, x_{m+1})-plane, and so forth.

4. Using the scaling relationship between our current box and the initial box of search, see if the new sub-box represents a sufficiently small box in $\mathbf{R}^m$. If it does not, then go to step 5. If it does, then check the convex hulls of the hypersurface in the new box. If the convex hulls cross each variable axis, conclude that there is a root or at least an approximate root in the new box, and put the new box into a root list. Otherwise the new box is discarded (see Appendix for elaboration of this point).

5. If any dimension of this sub-box are not much smaller than 1 unit in length (i.e., the box has not decreased much in size along one or more sides), split the box evenly along each dimension which is causing trouble. Continue on to the next iteration with several independent sub-problems.

6. If none of the boxes is left, then the root-finding process is over. Otherwise, go back to step 2, and perform it once for each new box.

If we assume that each equation 10.99 is degree m in each variable and the system is n-dimensional, then the total asymptotic time per step is of $O(n^2 m^{n+1})$. The number of steps depends primarily on the accuracy required [27].

The Projected Polyhedron Algorithm achieves quadratic convergence in one dimension, while for higher dimensions, it exhibits at best linear convergence.

10.15 Interval Newton method

Interval Newton methods [2] have been the focus of significant attention in numerical analysis. A thorough review of various types of interval Newton methods is presented in [3]. In the sequel we briefly review the interval Newton method. The interval Newton method solves a system of nonlinear equations in a numerically verifiable manner.

$$f_i(x_1, x_2, \cdots, x_n) = 0, \quad 1 \leq i \leq n \tag{10.105}$$

within boxes

$$a_i \leq x_i \leq b_i, \quad 1 \leq i \leq n \tag{10.106}$$

If we denote the n-vector whose ith component is x_i by $\mathbf{X}$ and the n-vector whose ith component is f_i by $\mathbf{F}(\mathbf{X})$, the interval Newton methods can be described as finding a box $\bar{\mathbf{X}}_k$ that contains all the solutions of the interval linear system

$$\mathbf{J}(\mathbf{X}_k)(\bar{\mathbf{X}}_k - \mathbf{X}_k^c) = -\mathbf{F}(\mathbf{X}_k^c) \tag{10.107}$$

where the subscript k denotes the kth iteration, $\mathbf{J}(\mathbf{X}_k)$ is the Jacobian matrix of $\mathbf{F}$ over the box $\mathbf{X}_k$, and $\mathbf{X}_k^c$ is some point in $\mathbf{X}_k$. There is a theorem about *unique solution* in the box given by [16]:

Theorem

If $\bar{\mathbf{X}}_k$ is *strictly* contained in $\mathbf{X}_k$, then the system of equations in (10.105) has a unique solution in $\mathbf{X}_k$, and Newton's method starting from any point in $\mathbf{X}_k$ will converge to that solution. Conversely, if $\mathbf{X}_k \cap \bar{\mathbf{X}}_k$ is empty, then there are no solutions of the system in (10.105).

The next iteration $\mathbf{X}_{k+1}$ is evaluated by

$$\mathbf{X}_{k+1} = \mathbf{X}_k \cap \bar{\mathbf{X}}_k \tag{10.108}$$

According to the mean value theorem, the solutions in the $\mathbf{X}_k$ must be in $\mathbf{X}_{k+1}$. If the coordinate intervals of $\mathbf{X}_{k+1}$ are smaller than those of $\mathbf{X}_k$, equations (10.107) and (10.108)

[2] When we use the term "interval Newton method", we assume that bisection is included in the process when interval reduction is not substantial by the pure interval Newton step.

are iterated until the bounding boxes are smaller than a specified tolerance. If the coordinate intervals of $\mathbf{X}_{k+1}$ are not smaller than those of $\mathbf{X}_k$, then one of these intervals is bisected to form two new boxes. The boxes are pushed into a stack and iteration is continued until the stack becomes empty. The first interval Newton method introduced by Moore [21] involves computing the inverse of the interval matrix $\mathbf{J}(\mathbf{X}_k)$. Hansen [12] introduced the Gaussian elimination procedure to solve the linear equation system in an interval Newton method. Krawczyk [12] introduced a variation of the interval Newton method which avoids the Gaussian elimination of an interval matrix by not attempting to obtain a sharp solution of (10.107). He computes the new box $\bar{\mathbf{X}}_k$ as follows

$$\bar{\mathbf{X}}_k = \mathbf{K}(\mathbf{X}_k) = \mathbf{X}_k^c - \mathbf{Y}_k\mathbf{F}(\mathbf{X}_k^c) + (\mathbf{I} - \mathbf{Y}_k\mathbf{J}(\mathbf{X}_k))(\mathbf{X}_k - \mathbf{X}_k^c) \tag{10.109}$$

where $\mathbf{Y}_k$ is a preconditioned matrix of midpoints of the elements of the interval Jocobian matrix. Hansen and Sengupta [12] introduced a box which is generally smaller than $\mathbf{K}(\mathbf{X}_k)$. They simply solve the ith equation for the ith variable and replace the others by bounding intervals, which is the non-linear version of the Gauss-Seidel operator for linear systems. Let $\mathbf{x}_i^c$ be the ith component of $\mathbf{X}_k^c$ and $\mathbf{k}_i$ be the ith component of $\mathbf{Y}_k\mathbf{F}(\mathbf{X}_k)$ and $\mathbf{G}_{ij}$ be the entry in the ith row and jth column of $\mathbf{Y}_k\mathbf{J}(\mathbf{X}_k)$ then, the step for the ith row of the Hansen-Sengupta operator becomes

$$\begin{aligned} \bar{\mathbf{x}}_i &= \mathbf{x}_i - [\mathbf{G}_{ii}]^{-1}[\mathbf{k}_i + \sum_{j=1}^{i-1} \mathbf{G}_{ij}(\hat{\mathbf{x}}_j - \mathbf{x}_j^c) + \sum_{j=i+1}^{n} \mathbf{G}_{ij}(\mathbf{x}_j - \mathbf{x}_j^c)] \\ \hat{\mathbf{x}}_i &= \mathbf{x}_i \cap \bar{\mathbf{x}}_i \end{aligned} \tag{10.110}$$

for $i = 1, \cdots, n$.

Bibliography

[1] S. L. Abrams, W. Cho, C.-Y. Hu, T. Maekawa, N. M. Patrikalakis, E. C. Sherbrooke, and X. Ye. Efficient and reliable methods for rounded-interval arithmetic. *Computer-Aided Design*, 30(8):657–665, July 1998.

[2] ANSI/IEEE Std 754–1985. *IEEE Standard for Binary Floating–Point Arithmetic*. IEEE, New York, 1985. Reprinted in *ACM SIGPLAN Notices*, 22(2):9-25, February 1987.

[3] C. Bliek. *Computer Methods for Design Automation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, July 1992.

[4] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory: Progress, Directions and Open Problems in Multidimensional Systems*, pages 184–232. Dordrecht, Holland: D. Reidel Publishing Company, 1985.

[5] J. F. Canny. Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9:241–250, 1990.

[6] G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[7] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, June 1988.

[8] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, Chichester, England, 1987.

[9] C. B. Garcia and W. I. Zangwill. Global continuation methods for finding all solutions to polynomial systems of equations in n variables. In A. V. Fiacco and K. O. Kortanek, editors, *Extremal Methods and Systems Analysis*, pages 481–497. Springer-Verlag, New York, NY, 1980.

[10] A. Geisow. *Surface Interrogations*. PhD thesis, School of Computing Studies and Accountancy, University of East Anglia, Norwich NR47TJ, U. K., July 1983.

[11] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis*. Addison-Wesley, Reading, MA, 4th edition, 1990.

[12] E. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:201–211, 1981.

[13] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A. K. Peters, Wellesley, MA, 1993. Translated by L. L. Schumaker.

[14] C. Y. Hu, T. Maekawa, N. M. Patrikalakis, and X. Ye. Robust interval algorithm for surface intersections. *Computer-Aided Design*, 29(9):617–627, September 1997.

[15] C. Y. Hu, T. Maekawa, E. C. Sherbrooke, and N. M. Patrikalakis. Robust interval algorithm for curve intersections. *Computer-Aided Design*, 28(6/7):495–506, June/July 1996.

[16] R. B. Kearfott and M. Novoa. INTBIS, a portable interval Newton/bisection package (algorithm 681). *ACM Transactions on Mathematical Software*, 16(2):152–157, June 1990.

[17] G. A. Kriezis, P. V. Prakash, and N. M. Patrikalakis. Method for intersecting algebraic surfaces with rational polynomial patches. *Computer-Aided Design*, 22(10):645–654, December 1990.

[18] J. M. Lane and R. F. Riesenfeld. Bounds on a polynomial. *BIT: Nordisk Tidskrift for Informations-Behandling*, 21(1):112–117, 1981.

[19] T. Maekawa and N. M. Patrikalakis. Computation of singularities and intersections of offsets of planar curves. *Computer Aided Geometric Design*, 10(5):407–429, October 1993.

[20] T. Maekawa and N. M. Patrikalakis. Interrogation of differential geometry properties for design and manufacture. *The Visual Computer*, 10(4):216–237, March 1994.

[21] R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[22] M. E. Mortenson. *Geometric Modeling*. John Wiley and Sons, New York, 1985.

[23] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *ACM Computer Graphics*, 24(4):337–345, August 1990.

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

[25] N. M. Patrikalakis and P. V. Prakash. Surface intersections for geometric modeling. *Journal of Mechanical Design, Transactions of the ASME*, 112(1):100–107, March 1990.

[26] H. Pottmann. General offset surfaces. *Neural, Parallel and Scientific Computations*, 5:55–80, 1997.

[27] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10(5):379–405, October 1993.

[28] W. I. Zangwill and C. B. Garcia. *Pathways to solutions, fixed points, and equilibria*. Prentice-Hall, Englewood Cliffs, NJ, 1981.