Three Applications of Interval Analysis in Computer Graphics

Don P. Mitchell

AT&T Bell Laboratories
Murray Hill, NJ 07974

ABSTRACT

Interval arithmetic and automatic differentiation are simple and robust techniques, which are often used together to solve difficult numerical problems. After discussing these techniques, three applications to computer graphics are examined: drawing contours of functions, ray tracing implicit surfaces, and ray tracing parametric surfaces.

1. Introduction

Interval analysis is an approach to numerical calculations using arithmetic on numbers representing ranges of values. This concept was originally applied to the analysis of numerical error; however, the usefulness of interval analysis has gone well beyond that. It is often necessary to know properties of a function within an interval, and sampling the function at individual points may fail to correctly reveal these properties. In particular, interval analysis has proven useful in finding the bounds on the value of a function, solving equations or systems of equations, optimization, differential equations and integral equations [Moore66; Moore79].

Interval algorithms sometimes require values of functions and of their derivatives. A useful technique for finding derivative values is automatic differentiation. This technique is superior in many ways to numerical differentiation (by finite differences) or symbolic differentiation. It is not unusual then that the methods of interval arithmetic and automatic differentiation are used in conjunction to solve some types of problems.

In this brief survey, the basic methods of interval arithmetic and automatic differentiation are presented. Three applications to computer graphics are discussed in some detail: finding contours of functions, intersecting rays with implicit surfaces, and intersecting rays with parametric surfaces. Readers should also look at the excellent paper by Mudur and Koparkar, which discusses some applications of interval arithmetic to graphics and geometry [Murur84].

2. Interval Arithmetic

An alternative number system can be defined, based on *interval numbers*. An interval number corresponds to a range of real values and can be represented by a pair of numbers $[a, b]$, the lower and upper bounds of the interval. In this system, an ordinary real number a can be represented by a degenerate interval $[a, a]$. Given interval numbers X and Y, we would like to compute interval values of expressions such as $X+Y$ or function values $f(X)$. Ideally, these values should be exact bounds, the range of lowest to highest values of $f(x)$ for all x *member* X. Often it is not possible to find the exact bound, but it may still be very useful to know an interval value guaranteed to contain the exact bound.

It is straightforward to define interval extensions of basic arithmetic operations:

$$[a, b] + [c, d] = [a + c, b + d] \tag{2.1a}$$

$$[a, b] - [c, d] = [a - d, b - c] \tag{2.1b}$$

$$[a, b] \; star \; [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \tag{2.1c}$$

and if $0 \; nomem \; [c, d]$

$$[a, b] \; / \; [c, d] = [a, b] \; star \; [1/d, 1/c] \tag{2.1d}$$

In many cases, interval multiplication can be done with fewer than four real multiplies. For example, if a, b, c, d are all positive, then $[a, b] \; star \; [c, d] = [ac, bd]$. Reciprocal is undefined if $0 \; member \; [c, d]$, but it is sometimes useful to simply return a special representation of the interval $[-\infty, \infty]$, rather than raising an exception. Hansen developed a more general definition of interval reciprocol in order to extend Newton's method to cases where the derivative interval contains zero [Hansen78]. His definition is:

$$1/[a, b] = \begin{cases} [1/b, 1/a] & \text{if } 0 \; nomem \; [a, b] \\ [1/b, \infty] & \text{if } a = 0 \\ [-\infty, 1/a] & \text{if } b = 0 \\ [-\infty, 1/a] \; cup \; [1/b, \infty] & otherwise \end{cases} \tag{2.2}$$

Given a rational function $r(x)$, the *natural interval extension* can be defined by evaluating the expression $r(X)$ for an interval argument $X = [x_0, x_1]$ and using the interval-arithmetic operations in (2.1). The resulting interval value of $r(X)$ is guaranteed to contain the exact bound of the real-valued $r(x)$ over the interval X. That is:

$$r(X) \; !subset \; \{r(x) \mid x \; member \; X\} \tag{2.3}$$

In fact, the interval $r(X)$ may be much larger than the exact bound of the real-valued function. The tightness of the bounds can depend on how $r(X)$ is expressed. Expressions that would be equivalent in ordinary real arithmetic can have different values when extended to interval arithmetic. For example, interval arithmetic is *subdistributive:* $X(Y + Z) \; !subset \; XY + XZ$.

One of the most important properties of the natural interval extension is *inclusion monotonicity:*

$$X' \; subset \; X, \quad \text{implies} \quad r(X') \; !subset \; r(X) \tag{2.4}$$

This means that as the interval X becomes more narrow, the interval $r(X)$ will narrow and converge toward its real restriction.

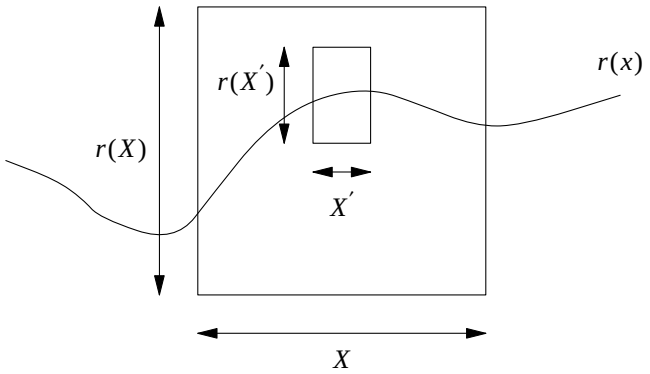

Figure 1. Inclusion Monotonicity

The natural interval extension was defined for rational functions, but this notion can be extended to include expressions containing many familiar transcendental functions. Finding the exact bound of any monotonic function is trivial, for example:

$$e^{[a, b]} = [e^a, e^b] \tag{2.5}$$

$$[a, b]^3 = [a^3, b^3] \tag{2.6}$$

Note that (2.6) is an exact bound, whereas using (2.1c) to compute $[a, b]$ *star* $[a, b]$ *star* $[a, b]$ might give a wider interval containing the exact bound (e.g., consider $[-1, 2]^3$).

Any continuous function $f(x)$ is made up of monotonic sections separated by local minima and maxima. These extrema occur at *critical points* where $f'(x) = 0$. If critical points can be found, then the exact bound of $f(x)$ on an interval can be computed. First, find all the critical points $c_1, c_2, \cdots$ contained within the interval $X = [a, b]$. Then the exact bound on the range of f in X is:

$$f([a, b]) = [\min(f(a), f(b), f(c_1), f(c_2), \cdots), \quad \max(f(a), f(b), f(c_1), f(c_2), \cdots)] \tag{2.7}$$

For many familiar transcendental functions, the locations of the extrema are known (e.g., the sine and cosine functions). In some cases, it is easy to solve $f'(x)$ (e.g., polynomials of degree three or less). Keep in mind that some, but not all, critical points where $f''(x) = 0$ are just inflection points, but there is no harm in including them.

The absolute-value function is often useful, and its interval extension is:

$$\text{abs}([a, b]) = \begin{cases} [a, b] & \text{if } a \geq 0 \\ [-b, -a] & \text{if } b \leq 0 \\ [0, \max(|a|, |b|)] & \textit{otherwise} \end{cases} \tag{2.8}$$

3. Contour Tracing

For the first application, consider the problem of drawing a contour plot of a two-dimensional function $f(x, y)$ in some region For the sake of example, let us restrict our attention to plotting the contour curve of $f(x, y) = 0$. A crude approach to this problem would be to divide the region of interest with a fine grid and sample $f(x, y)$ at every corner. Any grid cell whose corner values are not all positive or all negative must contain part of the contour. A trivial display algorithm would be to color cells with a sign change gray, and leave the rest white.

There are two serious problems with this grid-sampling approach. First, the function must be evaluated at a great many points. A high-resolution plot might require 500×500 samples or more. A more fundamental problem is that sampling can fail to detect some cells which do contain parts of the contour.

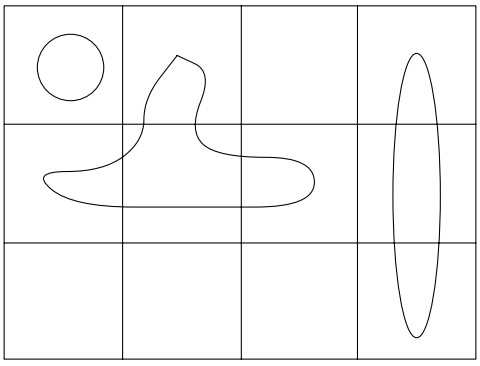

Figure 2. Contours Missed by Grid-Sampling Algorithm

Figure 2, illustrates a pathological situation where $f(x, y)$ has no sign changes at the corners of the grid.

The grid-sampling algorithm would not plot any dark cells in this case, even though a zero-level contour might exist, insinuated between all the samples. In practice, this can occur at enough places to result in a plot with missing gaps in a contour and other mistakes.

An interval extension of the function f allows us to construct a test which can quickly show that the contour is not contained in some areas. Assume we want to plot the contour in a square region of some size, starting at coordinates (i, j). Evaluate the interval extension $F = f([i, i+size], [j, j+size])$. If zero is not contained in this bound, $0 \; nomem \; F$, then the contour *definitely* cannot pass through this square region. If zero is contained in the bound, *maybe* the square contains part of the contour. We cannot be certain because the natural interval extension does not give the exact bound of the function within the square. This idea has been applied by Suffern and Fackerell [Suffern90]. We will consider a somewhat simplified version of their algorithm, with a trivial plotting operation (coloring cells gray or white).

An initial square region can be recursively subdivided in a quadtree fashion, down to some limiting square size. The routine below is a naive application of this test. If the interval value of f shows that there is no contour in a subsquare, it is colored white. If, at the lowest level of subdivision, the subsquare might contain the contour, it is colored gray. The interval test is applied by the function **noContour**.

```
plotSquare(i, j, size)
int i, j, size;
{

    if (noContour(i, j, size)) {
        whiteSquare(i, j, size);
    } else {
        if (size > 1) {
            plotSquare(i, j, size/2);
            plotSquare(i+size/2, j, size/2);
            plotSquare(i+size/2, j+size/2, size/2);
            plotSquare(i, j+size/2, size/2);
        } else
            graySquare(i, j, size);
    }
}
```

Program 1. Naive algorithm for plotting contours

Figure 3 shows the result of this algorithm applied to a sum of Gaussians:

$$e^{-(x-1)^2-y^2} + e^{-(x+1)^2-(y+0.5)^2} \tag{3.1}$$

A clear advantage of this algorithm is the quick elimination of large squares, known to not contain the contour. A clear disadvantage is that the "don't know" regions, colored in gray, do not necessarily define the contour very well. In Figure 3, the topology of the contour is ambiguous; it could be one loop, two separate loops, or a figure "8".

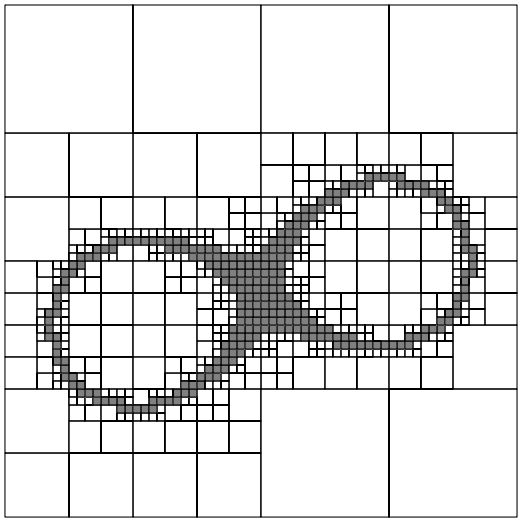

Figure 3. Output of Naive Interval Contouring Algorithm

Suffern and Fackerell suggest applying the grid-sampling algorithm within the ambiguous (gray-colored) subsquares. This should give the same result as applying grid sampling over the whole region, but skipping all the white-colored subsquares means greatly improved performance. However, there may still be ambiguous subsquares, where there are no sign changes at the corners, but still 0 *member* $f(X,Y)$.

The inclusion-monotonicity property described in (2.4) guarantees that tighter bounds on the range of a function can be found by narrowing the range of the arguments. That is, instead of computing $f(X)$, we can subdivide the interval X and evaluate f on these subintervals to find a tighter bound of the function's range. This concept can be applied to the naive interval contouring algorithm, subdividing the region below the display-grid resolution. Before coloring a square, it will be subdivided further, and at least one subsquare must still be ambiguous before the square is colored gray. Figure 4 illustrates this process, subdividing 16 levels below that displayed.

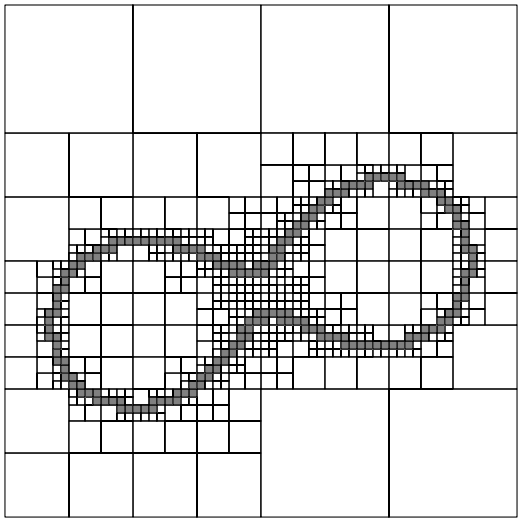

Figure 4. Using Subdivision Below the Display Resolution.

This reveals the correct topology of the contour. However, the gray squares are still ambiguous regions, some of which might not actually contain part of the contour. Testing the sign changes on the corners of cells also leaves some cells ambiguous, so a completely satisfying solution to the problem has not been presented. However, interval analysis has made a positive contribution by restricting the ambiguities to a much smaller total area. Finer subdivision increases the probability that an ambiguous region actually does contain part of the contour. This method has also been applied to the three dimensional problem of polygonizing implicit surfaces [Suffern90].

In the two applications described later, we will be locating isolated roots, single points where some function is zero. In many ways, that is an easier problem than locating and representing a one or two-dimensional zero set.

4. Automatic Differentiation

The previous section described a useful application of interval analysis. In addition to evaluating an interval extension of a function f, some numerical methods may require values of the derivative f'. This section presents a method for computing values of derivatives which can be implemented with ordinary or interval arithmetic.

The simplest approaches is to compute the derivative *numerically* with a finite-difference approximation:

$$f'(t) \approx \frac{f(t+\Delta t) - f(t)}{\Delta t} \tag{4.1}$$

Unfortunately, this method is too inaccurate for many applications. Another approach is to compute the derivative *symbolically,* by operating on some formula representation:

$$\frac{d}{dt}(3t^2 \sin(t)) = 3t^2 \cos(t) + 6t \sin(t) \tag{4.2}$$

However, symbolic manipulation of expressions can become complicated and inefficient. Even with simplification heuristics, a symbolic derivative can become a much larger expression than the original function.

A very practical solution to this problem is a technique called *automatic differentiation,* which can compute a value of f and its derivative concurrently with only a little more effort than computing $f(t)$ alone. This technique, like interval analysis, was introduced by Ramon E. Moore, and an excellent survey of the subject

can be found in [Rall81].

Automatic differentiation is accomplished by evaluating the expression for $f(t)$, using an alternative set of rules for *derivative-arithmetic*. These operations are applied to pairs of *numbers* representing values of $f(t)$ and $f'(t)$ for some value of t. For example, a constant would be represented by the pair $(c, 0)$. The independent variable would be represented by $(t, 1)$. Starting with these definitions, we can evaluate arbitrary expressions by applying primitive operations such as:

$$\begin{aligned} (f, f') + (g, g') &= (f + g, f' + g') \\ (f, f') \; star \; (g, g') &= (fg, fg' + f'g) \\ \sin(f, f') &= (\sin(f), f' \cos(f)) \end{aligned} \tag{4.3}$$

These rules are based on the laws of calculus, addition of derivatives, the product rule, the chain rule, etc. Beginning with the definition of constants and the independent variable, t, rules like these can be applied to the expression for f and will result in the pair of numbers $(f(t), f'(t))$.

When interval values (f, f') are computed from (2.1) and (4.3), the information from the derivative can sometimes be used to improve the bounds on f. If $0 \; nomem \; f'([a, , b])$, then we know the function must be monotonic in the interval $[a, b]$. In that case, the *exact bound* on the range of f is known to be $[f(a), f(b)]$ or $[f(b), f(a)]$, depending on whether f is increasing or decreasing. If not monotonic, one can still compute a bound on f from the interval value of f' by:

$$f((a+b)/2) + f'([a, b])(b - a)[-\tfrac{1}{2}, \tfrac{1}{2}]. \tag{4.4}$$

This is an example (the midpoint form) of the very important *mean-value extension* of a function. This is an alternative to the natural extension, and may or may not give a better bound on the range of a function. However, as an interval T becomes more narrow, the mean-value extension $f_{MV}(T)$ converges to the exact bound more rapidly than the natural extension. The mean-value extension is illustrated in Figure 5. The dotted line and the dashed line represent the lower and upper bound on the range of slopes given by the interval extension $f'(T)$:

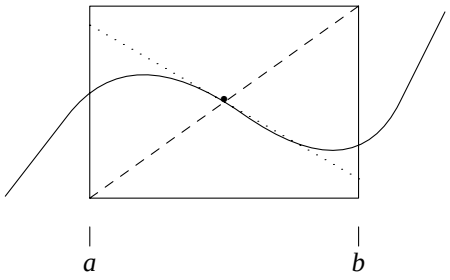

Figure 5. The Mean-Value Interval Extension (Midpoint Form)

5. Ray Tracing Implicit Surfaces

The second application uses both interval arithmetic and automatic differentiation. A common representation of surfaces in graphics is the *implicit surface,* defined by the zero set of a three-dimensional scalar-valued function, $g(\vec{x}) = 0$, where $\vec{x}$ is the coordinate vector of a point (x, y, z). Implicit surfaces are particularly useful in ray tracing, because the problem of finding the intersection with a ray is easy to understand. Assume the *parametric* representation of a ray, $r(t) = \vec{o} + t\vec{d}$, where the parameter t can range from zero to infinity. To find the intersection of the ray with the surface, find points on the ray satisfying $g(\vec{x}) = 0$. By a simple substitution, this problem reduces to the solution of a single equation in one variable, $g(r(t)) \equiv f(t) = 0$.

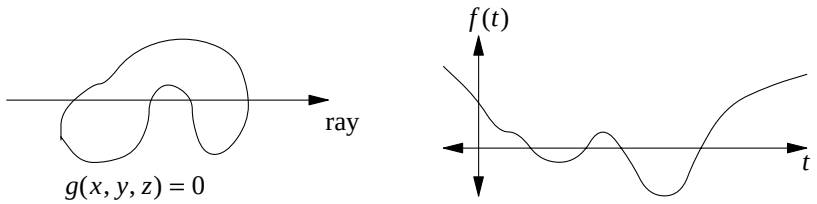

Figure 5. Ray Intersections Correspond to Solutions of $f(t) = 0$.

The problem is to find solutions of $f(t) = 0$. Linear and quadratic equations have closed-form solutions, but the solution of general nonlinear equations, $f(t) = 0$, will require numerical methods. This problem can be divided into two tasks. The first is *root isolation,* finding intervals $(t_i, t_{i+1}]$ which are known to contain a single root of $f(t) = 0$, as illustrated below:

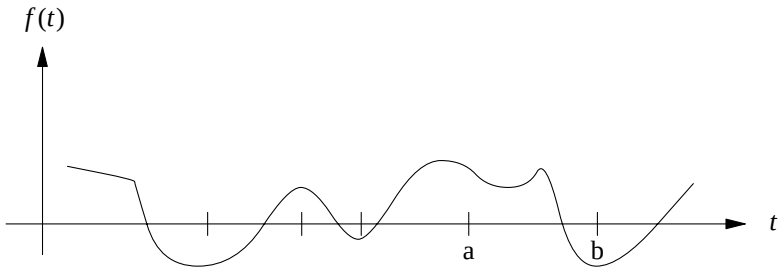

Figure 6. Root Isolation

The second task is *root refinement,* where each isolated root is located as accurately as possible. This involves a process of closing in on the root with ever-smaller bounds, as shown below:

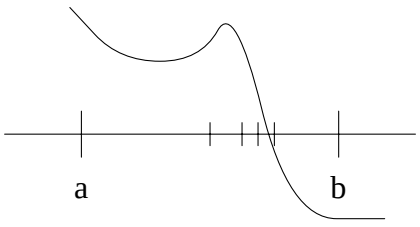

Figure 7. Root Refinement

Root refinement is well understood, and familiar algorithms such as bisection or *regula falsi* or the highly efficient Brent-Dekker method [Brent73] could be used. *The hard problem is root isolation.* This fact is not always sufficiently stressed in introductions to numerical methods. If *g* is algebraic, then $f(t)$ will be a polynomial. Reliable root isolation methods for the real roots of polynomials have been developed based on Descartes' rule of signs or Sturm's theorem, and these methods have been applied in ray tracing [Hanrahan83; van Wijk84].

If an implicit solid is defined by a function $g(\vec{x})$ which is not polynomial, the problem of finding roots in $f(t)$ is much more difficult. Only a few examples exist in the graphics literature. Blinn ray traced "blobby"

surfaces defined by sums of three-dimensional Gaussians [Blinn82]. He used an heuristic method for solving $f(t)$. Kalra and Barr report a robust algorithm for ray tracing a class of non-algebraic surfaces for which Lipschitz conditions are known to hold (i.e., when a bound on Δf is proportional to Δt) [Kalra89].

Moore gives a simple and general algorithm for root isolation which can be applied to rational functions and also to expressions involving familiar transcendental functions [Moore66]. This methods is based on the use of interval arithmetic, and has been applied to ray tracing non-algebraic surfaces [Mitchell90]. With the ability to bound the range of a function, a simple recursive algorithm can be defined to isolate the roots of a function of one variable. We start with an initial interval $[a, b]$ in t, perhaps obtained from a bounding box.

Step 1. Evaluate $f([a, b])$. If $0 \; nomem \; f([a, b])$, there are no roots in this interval.

Step 2. Evaluate the derivative $f'([a, b])$. If $0 \; nomem \; f'([a, b])$, then f is *monotonic* in the interval. In that case, if $f(a)\, f(b) \le 0$, there is one root in the interval which can be found by root refinement. If $f(a)\, f(b) > 0$, there are no roots in the interval.

Step 3. If $f([a, b])$ and $f'([a, b])$ both contain zero, then subdivide the interval at its midpoint and recursively process $[a, (a + b)/2]$ and $[(a + b)/2, b]$.

Step 4. The process of subdivision should be stopped when the width of an interval approaches the limits of machine precision.

In step 3, the closest interval is processed first. If only the closest ray intersection is desired, the algorithm can stop the first time it finds a root in Step 2 or 4. In a CSG model, it may be necessary to find all roots. For a shadow-probe ray, it is sometimes possible to skip the root refinement step. Proof of the existence of a root may be enough to complete a shadow calculation.

Automatic differentiation (using interval arithmetic) can be used to efficiently compute the values of $f(t)$ and $f'(t)$ concurrently. However, there is the problem that we do not have an explicit expression for $f(t)$, and it is not always practical to derive it symbolically from the ray and $g(\vec{x})$. Fortunately, it is not necessary to derive $f(t)$. All we need are $g(\vec{x})$ and derivative-number values for the variables x, y, z:

$$\begin{aligned} (x, \partial x/\partial t) &= (o_x + td_x, d_x) \\ (y, \partial y/\partial t) &= (o_y + td_y, d_y) \\ (z, \partial z/\partial t) &= (o_z + td_z, d_z) \end{aligned} \tag{5.1}$$

Using these, we evaluate $g(x, y, z)$ and automatically end up with the derivative number, $(f(t), f'(t))$. Automatic differentiation can also be used to compute ∇g, the surface normal used in shading.

6. Ray Tracing Parametric Surfaces

The last application we will look at is Toth's algorithm for intersecting a ray with a parametric surface [Toth85]. Ray tracing parametric surfaces is a difficult problem, and Toth's method is general, robust and correct. This is a sophisticated application of interval analysis, and in this tutorial, I will focus more on describing the algorithm than on proving the mathematics.

The points on a parametric surface are given explicitly by a function (often a bicubic function) of two parameters, $s(u, v)$, in which each coordinate is a function of the parameters $(s_x(u, v), s_y(u, v), s_z(u, v))$. Often, we are interested in a parametric *patch* where the parameters are bounded (e.g., $0 \le u \le 1, 0 \le v \le 1$). Likewise, the parametric form of a ray is a set of points given as a function of one parameter, $r(t) = \vec{o} + t\vec{d}$. Let $\vec{u}$ refer to the parameter vector (u, v, t), and the difference of the surface and ray vectors is given by

$f(\vec{u}) = s(u,v) - r(t)$. Points of intersection between the surface and ray correspond to solutions of $f(\vec{u}) = 0$, which is a system of three equations in three unknowns, u, v, t:

$$\begin{aligned} s_x(u,v) - o_x - td_x &= 0 \\ s_y(u,v) - o_y - td_y &= 0 \\ s_z(u,v) - o_z - td_z &= 0 \end{aligned} \tag{6.1}$$

An alternative formulation of the problem uses an implicit form of the ray (of a line, actually), the intersection of two orthogonal planes given by $Ax + By + Cz + D = 0$ and $Ex + Fy + Gz + H = 0$. The coordinates of the parametric surface can be substituted into the two plane equations giving a system of two equations in two unknowns, u, v:

$$\begin{aligned} As_x(u,v) + Bs_y(u,v) + Cs_z(u,v) + D &= 0 \\ Es_x(u,v) + Fs_y(u,v) + Gs_z(u,v) + H &= 0 \end{aligned} \tag{6.2}$$

In either case, these are generally systems of nonlinear equations, and fast, reliable algorithms for solving them are not easy to come by. A variety of iterative methods of root refinement can be extended to multidimensional problems; in particular, Newton's method:

$$\vec{u}_{k+1} = \vec{u}_k - [f'(\vec{u}_k)]^{-1} f(\vec{u}_k) \tag{6.3}$$

Because, f is a mapping from $\mathbf{R}^n$ to $\mathbf{R}^n$, its derivative f' is a linear map represented by an $n \times n$ matrix (the Jacobian). In the case of (6.1), for example:

$$f' = \begin{bmatrix} \frac{\partial f_x}{\partial u} & \frac{\partial f_x}{\partial v} & \frac{\partial f_x}{\partial t} \\ \frac{\partial f_y}{\partial u} & \frac{\partial f_y}{\partial v} & \frac{\partial f_y}{\partial t} \\ \frac{\partial f_z}{\partial u} & \frac{\partial f_z}{\partial v} & \frac{\partial f_z}{\partial t} \end{bmatrix} \tag{6.4}$$

Newton's method requires the matrix inversion of f' at each step. In fact, we can forgo that expense and just use the initial $[f'(\vec{u}_0)]^{-1}$ matrix for every step of the iteration. This gives the *simple Newton's method*, but it is at best only linearly convergent, while (6.3) can converge quadratically.

Newton's method by itself is not a solution to our problem. It is a root-refinement algorithm which sometimes behaves erratically. We want to find all the intersections of the ray with the parametric patch, or at least the closest intersection. In order to do that reliably, we will have to methodically search a region of the parameter space (of two or three dimensions), find regions known to contain one root, and use Newton's method only when it is certain to converge properly. Interval analysis is one of the surest ways to do that.

Interval Newton's method has been studied by Moore and others [Moore66; Moore79]. An interval vector $\vec{U}$ is made up of interval values (U, V, T) and represents a rectangular box in parameter space. Given an initial vector $\vec{U}_0$, a straightforward interval extension of (6.3) is given by:

$$\vec{U}_{k+1} = N(\vec{U}_k) \; cap \; \vec{U}_k \tag{6.5}$$

where

$$N(\vec{U}) = m(\vec{U}) - [F'(\vec{U})]^{-1} f(m(\vec{U}))$$

Here, $m(\vec{U})$ is an ordinary vector whose components are the midpoints of the interval components of $\vec{U}$. The intersection with $\vec{U}$ in $N(\vec{U}) \; cap \; \vec{U}$ is a precaution against effects of finite-precision machine arithmetic. A problem with the $N(\vec{U})$ operator is that it involves the inversion of an interval matrix. Krawczyk developed an alternative Newton-like method using an operator that is cheaper to compute:

$$\begin{aligned} K(\vec{U}) = m(\vec{U}) &- [m(F'(\vec{U}))]^{-1} f(m(\vec{U})) \\ &+ \{I - [m(F'(\vec{U}))]^{-1} F'(\vec{U})\} \, (\vec{U} - m(\vec{U})) \end{aligned} \tag{6.6}$$

In this operator, the the inversion of an ordinary matrix $m(F'(\vec{U}))$ is computed. The upper portion of (6.6)

involves ordinary vectors and is a Newton step taken from the midpoint of $\vec{U}$. The lower portion is an interval vector which will add an interval on either side of the midpoint Newton step. I denotes the $n \times n$ unit matrix. Moore and others have proven a number of very important theorems about when iterative root-finding methods of Krawczyk's form will converge [Moore79].

Theorem 1 (Existence). *If $K(\vec{U})$!subset $\vec{U}$, at least one root exists in $\vec{U}$.*

Theorem 2 (NonExistence). *If $K(\vec{U})$ cap $\vec{U}$ = empty, there are no roots in $\vec{U}$.*

Theorem 3 (Convergence). *If $K(\vec{U})$!subset $\vec{U}$ and $\|I - [m(F'(\vec{U}))]^{-1} F'(\vec{U})\| < 1$, there is a unique root in $\vec{U}$ and a number of iterative methods will converge to it.*

The interval vector norm is define as $\|\vec{X}\| = \max(|X_1|, \cdots |X_n|)$, and the absolute value of an interval number is defined by $|[a, b]| = \max(|a|, |b|)$. The absolute value of an interval number $|X_1|$ is a real number, and should not be confused with the natural extension of the absolute value function in (2.8).

If the conditions of Theorem 3 are met, several different iterative methods can be shown to converge. Simple Newton's method is one:

$$\vec{u}_{k+1} = \vec{u}_k - [m(F'(\vec{U}_0))]^{-1} f(\vec{u}_k) \tag{6.7}$$

Another iterative method that will converge is a type of interval Newton method:

$$\vec{U}_{k+1} = K(\vec{U}_k) \; cap \; \vec{U}_k \tag{6.8}$$

where

$$K(\vec{U}_k) = m(\vec{U}_k) - Y_k f(m(\vec{U}_k)) + \{I - Y_k F'(\vec{U}_k)\}(\vec{U}_k - m(\vec{U}_k))$$

and

$$Y_k = \begin{cases} Y = [m(F'(\vec{U}_k))]^{-1} & \text{if } \|I - YF'(\vec{U}_k)\| \le \|I - Y_{k-1} F'(\vec{U}_k)\| \\ Y_{k-1} & \text{otherwise} \end{cases}$$

Toth proves an additional theorem about this interval Newton's method:

Theorem 4 (Toth). *If $\|I - [m(F'(\vec{U}))]^{-1} F'(\vec{U})\| < 1$, then either the interval Newton's method in (6.8) will converge to a unique root in $\vec{U}$ or it will terminate with $K(\vec{U})$ cap $\vec{U}$ = empty, in which case there is no root.*

Toth's algorithm is based on the above theorems. Assume we are beginning with a parametric patch with $U = [0, 1]$ and $V = [0, 1]$. By plugging interval values for U, V into an interval extension of the parametric surface formula, we can compute a spatial *extent*, $\vec{X} = s(U, V)$, which will be an axis-aligned box. The ray can be intersected with the box $\vec{X}$ (i.e., clipped) to find an interval value of the ray parameter T. Given an initial U, V, T, roots can be found by a recursive search procedure of bisection and testing:

Step 1. If $K(\vec{U}_k)$!subset $\vec{U}_k$ and $\|I - m(F'(\vec{U}_k))^{-1} F'(\vec{U}_k)\| < 1$, then use simple Newton's method (6.7) to find the root.

Step 2. If the ray misses the interval extent, *ray cap* $\vec{X}_k$ = *empty*, reject $\vec{U}_k$.

If a point of intersection with the surface has been found which is closer than *ray cap* $\vec{X}_k$, reject this interval (unless we will need all the intersection points for CSG).

If $K(\vec{U}_k)\ cap\ \vec{U}_k = empty$, reject U_k.

Step 3. If $\|I - Y_k F'(\vec{U}_k)\| < 1$, narrow the interval by taking steps of the interval Newton's method (6.8). If at any time, $K(\vec{U}_k)\ cap\ \vec{U}_k = empty$, reject this interval. If $K(\vec{U}_k)\ !subset\ \vec{U}_k$, go to Step 1.

Step 4. Subdivide $\vec{U}_k$ along its widest dimension. Recursively analyze the two halves, starting with the one whose extent $\vec{X}$ is closest to the ray origin.

Step 5. If $\vec{U}_k$ or F' become smaller than some tolerance, stop bisecting. If one step of Newton's method stays inside $\vec{U}_k$, assume a root is found; otherwise, reject the interval.

This is very similar to Moore's algorithm [Moore79], but Step 4 makes use of Toth's theorem 4 to narrow intervals without bisection. Toth also presents another test for convergence in addition to Theorem 3. Notice that simple Newton's method (using real numbers) is invoked as soon as it is possible. More detail can be found in Toth's paper. The notation used by Toth is consistent with Moore's, but may confuse some readers. For example, the symbol X is sometimes used to refer to a vector (U, V, T) rather than the Cartesian coordinate. The following table may help:

quantity	this tutorial	Moore & Toth
coordinates	x, y, z	x, y, z
surface params.	u, v	s, t
ray param.	t	u
param. vector	$\vec{u}$	v or x
interval param. vector	$\vec{U}$	X
interval coordinate vector	$\vec{X}$	$([X_{\min}, X_{\max}], \cdots)$

Table 1. Notation: A Guide to the Perplexed

An implementor of this algorithm could experiment with many variations. Use automatic differentiation to compute F'. Try working in the two-dimensional space of (6.2) instead of three dimensions (Toth suggests something along those lines). Try using true Newton's method in Step 1 instead of simple Newton's method (see chapter 5 of [Moore79] for exactly how to do that).

Often parametric patches are used in a mesh, to describe a large object. A ray must be intersected with a number of patches in the mesh, and there is a danger that it will slip through a numerical "crack" between patches. An interesting thing to try would be to treat the entire mesh as a single parametric surface. Because Toth's algorithm is so general, this could be done. A one-dimensional analog can be found in Grandine's interval-analysis algorithm for finding zeros of splines [Grandine89].

7. References

[Blinn82] Blinn, J. F. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1, 3 (July 1982), 235-256.

[Brent73] Brent, R. P. *Algorithms for Minimization without Derivatives.* Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Grandine89] Grandine, T. A. Computing zeros of spline functions. *Computer Aided Geometric Design*, 6 (1989), 129-136.

[Hanrahan83] Hanrahan, P. Ray tracing algebraic surfaces. *Computer Graphics*, 17, 3 (July 1983), 83-90.

[Hansen78] Hansen, E. A globally convergent interval method for computing and bounding real roots. *BIT*, 18, 4 (1978), 415-424.

[Kalra89] Kalra, D. and Barr, A. H. Guaranteed ray intersection with implicit surfaces. *Computer Graphics* 23, 3 (July 1989), 297-306.

[Mitchell90] Mitchell, D. P. Robust ray intersection with interval arithmetic. *Graphics Interface 90*, (May 1990) 68-74.

[Moore66] Moore, R. E., *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Moore79] Moore, R. E., *Methods and applications of interval analysis.* SIAM, 1979.

[Mudur84] Mudur, S. P. and Koparkar, P. A. Interval methods for processing geometric objects. *IEEE Computer Graphics and Applications*, (February 1984) 7-17.

[Suffern90] Suffern, K. G., Fackerell, E. D. Interval methods in computer graphics. *AUSGRAPH 90*, (1990), 35-44.

[Toth85] Toth, D. L. On ray tracing parametric surfaces. *Computer Graphics*, 19, 3 (July 1985), 171-179.

[Wijk84] van Wijk, J. J., Ray tracing objects defined by sweeping a sphere. *Eurographics 84*, (September 1984), 73-82.