Interval Analysis For Computer Graphics

John M. Snyder
California Institute of Technology
Pasadena, CA 91125

Abstract

This paper discusses how interval analysis can be used to solve a wide variety of problems in computer graphics. These problems include ray tracing, interference detection, polygonal decomposition of parametric surfaces, and CSG on solids bounded by parametric surfaces. Only two basic algorithms are required: SOLVE, which computes solutions to a system of constraints, and MINIMIZE, which computes the global minimum of a function, subject to a system of constraints.

We present algorithms for SOLVE and MINIMIZE using interval analysis as the conceptual framework. Crucial to the technique is the creation of "inclusion functions" for each constraint and function to be minimized. Inclusion functions compute a bound on the range of a function, given a similar bound on its domain, allowing a branch and bound approach to constraint solution and constrained minimization. Inclusion functions also allow the MINIMIZE algorithm to compute global rather than local minima, unlike many other numerical algorithms.

Some very recent theoretical results are presented regarding existence and uniqueness of roots of nonlinear equations, and global parameterizability of implicitly described manifolds. To illustrate the power of the approach, the basic algorithms are further developed into a new algorithm for the approximation of implicit curves.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; G.4 [Mathematical Software]: Reliability and Robustness

Additional Key Words: constraint solution, constrained minimization, interval analysis, inclusion function, approximation, implicit curve

1 Introduction

Interval analysis is a new and promising branch of applied mathematics. A general treatment can be found in [MOOR66] and [MOOR79], by R.E. Moore, the originator of this field. The main benefit of interval analysis is that it can solve problems so that the results are guaranteed to be correct, even when computed with finitely precise floating point operations. This is accomplished by using inclusion functions that compute bounds on functions relevant to the problem, thus controlling approximation errors.

Although the application of interval methods to computer graphics is not new, it has been applied only to a limited class of computer graphics problems. Mudur and Koparkar [MUDU84] have presented an algorithm for rasterizing parametric surfaces using interval arithmetic. They also suggest the utility of such methods for other operations in geometric modeling. Toth [TOTH85] has demonstrated the usefulness of interval based methods for the direct ray tracing of general parametric surfaces. Most recently, interval methods have been used for error bounding in computing topological properties of toleranced polyhedra [SEGA90], for contouring 2D functions and rendering implicit surfaces [SUFF90], and for ray tracing implicit surfaces [MITC90]. Several researchers have also used Lipschitz bounds, a special case of an interval method, in their algorithms: to approximate parametric surfaces [VONH87], to compute collisions between time-dependent parametric surfaces [VONH89,VONH90], and to ray trace implicit surfaces [KALR89].

This paper extends the work of these researchers by showing how a general set of problems in computer graphics can be solved using only two algorithms that employ interval analysis: constraint solution (SOLVE) and constrained minimization (MINIMIZE). Many of the ideas presented here are borrowed from recent work in the area of interval analysis ([RATS88, ALEF83]), but are new to computer graphics. These ideas include the actual algorithms for SOLVE (Section 3.1) and MINIMIZE (Section 3.3), and a robust test for the solution of nonlinear systems of equations (Section 3.2). Section 2 presents background information necessary for the understanding of these ideas. Using the techniques described, Section 4 presents a new, robust algorithm for the approximation of implicit curves, an important algorithm in shape modeling operations such as CSG.

1.1 Problem Definition for SOLVE and MINIMIZE

SOLVE computes solutions to a *constraint problem*, which seeks points from a domain,[1] $D \subset \mathbf{R}^n$, that satisfy a logical combination of equalities and inequalities. That is, it seeks the set given by

$$\underset{x \in D}{\mathsf{SOLVE}}\ F \equiv \{x \mid x \in D, F(x) = 1\}$$

where $F\colon \mathbf{R}^n \to \{0,1\}$ represents the constraint to be solved. For example, $F(x)$ may be given by the simultaneous satisfaction of the $r+s$ constraints

$$\begin{array}{rcll} g_i(x) & = & 0 & i = 1,\ldots,r \\ h_j(x) & \leq & 0 & j = 1,\ldots,s \end{array}$$

The scalar functions $g_i(x)$ and $h_j(x)$ are called the *constraint functionals*, and are assumed to be continuous.

Related to the constraint problem is what we call the *constrained partitioning problem*, which seeks to partition a domain D into a collection of hyper-rectangles, $\{R_i\}$, such that each partition R_i satisfies a given set constraint. A *set constraint* is a mapping from a hyper-rectangle to $\{0,1\}$. For example, let $S\colon \mathbf{R}^2 \to \mathbf{R}^3$ be a parametric surface, and let $R \subset \mathbf{R}^2$. Define the distance function of the surface, $d(R)$, as the maximum distance between two surface points, mapped from R:[2]

$$d(R) \equiv \sup\{\|S(p_1) - S(p_2)\| \mid p_1, p_2 \in R\}$$

A useful set constraint for the surface approximation problem, $G(R)$, is

$$G(R) \equiv (d(R) < \epsilon) \qquad (1)$$

which requires that no two points on S from R be farther apart than ϵ. A constrained partitioning problem can also be combined with a constraint problem, in order to partition the constraint problem's solution set. We will later show how SOLVE can be applied to the constrained partitioning problem.

[1]We will assume the domain of all problems is a hyper-rectangle, also called a vector-valued interval in Section 2.1.

[2]sup denotes supremum, the least upper bound of a set.

MINIMIZE computes solutions to a *constrained minimization problem*, which seeks the global minima of a scalar function, $f(x)$, called the *objective function*, over the points in a given domain that satisfy a system of constraints. Two possible solutions may be required, the minimum value of the objective function: [3]

$$\underset{x \in D, F(x)=1}{\text{MINIMUM}} f \equiv \inf\{f(x) \mid x \in D, F(x) = 1\}$$

or its set of global minimizers:

$$\underset{x \in D, F(x)=1}{\text{MINIMIZERS}} f \equiv \{x \mid f(x) = \underset{x \in D, F(x)=1}{\text{MINIMUM}} f\}$$

Note that the MINIMUM operator is well-defined only if the feasible set of the constraint system is nonempty. We assume the continuity of the objective function as well as the constraint functionals involved in the definition of $F(x)$. This, together with the compactness of D, guarantees that the MINIMIZERS operator is well-defined when the feasible set is nonempty.

We will use the term *global problem* for a constraint problem, constrained partitioning problem, or constrained minimization problem.

1.2 SOLVE and MINIMIZE in Computer Graphics

SOLVE and MINIMIZE can be applied to a wide variety of problems in rendering and geometric modeling. We shall describe an important, but by no means exhaustive, set of examples in the following paragraphs. Further applications of the algorithms to computer graphics include scan conversion of parametric surfaces, parametric/implicit representation conversion, selection of feasible/optimal parameters for parameterized shapes, and computation of toleranced geometric queries such as point enclosure.

Ray Tracing Ray tracing a parametric surface, $S(u,v)\colon \mathbf{R}^2 \to \mathbf{R}^3$, involves a minimization problem over (u,v) space. Let o and d be the origin and direction, respectively, of a given ray. As in [TOTH85], to find the first intersection of this ray with S, we may solve

$$\underset{\substack{(u,v) \in D_S \\ F(u,v)=1}}{\text{MINIMUM}} t(u,v)$$

where the objective function, $t(u,v)$, is given by [4]

$$t(u,v) \equiv \min\{\frac{S_x(u,v) - o_x}{d_x}, \frac{S_y(u,v) - o_y}{d_y}, \frac{S_z(u,v) - o_z}{d_z}\}$$

and the constraint function $F(u,v)$ is given by

$$(S(u,v) - o) \times d = 0 \quad \text{and} \quad t(u,v) \geq 0$$

Ray tracing of implicit surfaces can be accomplished with a similar, 1D minimization problem [MITC90].

Polygonal Decomposition Approximating a shape, such as a curve or surface, as a collection of simple pieces is a fundamental operation in computer graphics. For example, we may wish to produce a collection of triangles that approximate a parametric surface, $S(u,v)$, to some error tolerance. Such an approximation can be accomplished using constrained partitioning with the set constraint, $G(R)$, from Formula (1). For each resultant (u,v) partition, a set of triangles can be formed joining the partition's four corner vertices, as well as any vertices from more highly subdivided neighbor partitions (Figure 1). The whole collection of triangles approximates the surface without deviating from it more than a distance of ϵ. Set constraints can be also be defined that bound each partition's surface area, maximum variation of surface normal, or any other function over the surface. Figure 10 compares polygonal decomposition using set constraints with simple uniform sampling.

Interference Detection Let $S(u,v)$ and $T(r,s)$ be two parametric surfaces, with D_S and D_T their respective domains. To compute whether these surfaces intersect, the following 4D constraint problem is appropriate:

$$\underset{\substack{(u,v) \in D_S \\ (r,s) \in D_T}}{\text{SOLVE}} (S(u,v) = T(r,s))$$

[3] inf denotes infimum, the greatest lower bound of a set.

[4] If any ray direction components are equal to 0, the corresponding quotient is taken to be ∞, and so is ignored in the min.

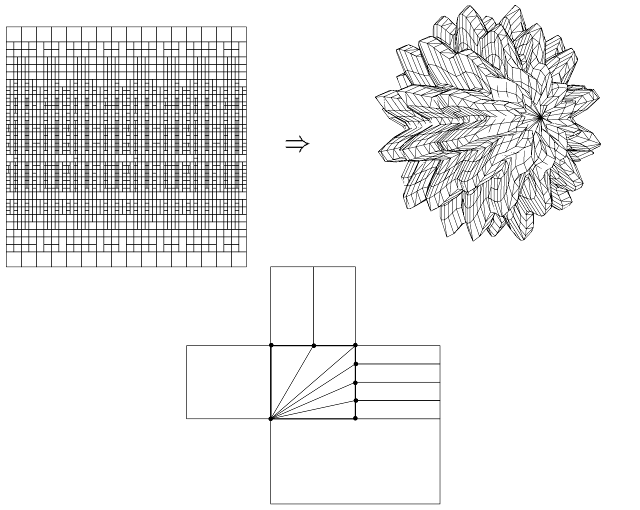


Figure 1: Approximating a surface as a triangular mesh – The surface's parameter space is first broken into rectangles each of which satisfies a set constraint, controlling the approximation quality. A collection of triangles is then generated connecting points at the corners of the rectangle or its neighbors.

In this case, evaluation of the points of intersection is unnecessary; we need only compute whether or not the feasible set is empty. Similar constraint problems can be solved to determine whether two moving surfaces intersect, by solving a 5D constraint problem given the time-dependent surfaces $S(u,v,t)$ and $T(r,s,t)$ [VONH90].

A related problem is to determine the minimum distance between two parametric surfaces, which may be expressed as the unconstrained minimization problem

$$\underset{\substack{(u,v) \in D_S \\ (r,s) \in D_T}}{\text{MINIMUM}} ||S(u,v) - T(r,s)|| \tag{2}$$

Figure 11 shows the results of the unconstrained minimization problem to compute the minimum distance between two parametric surfaces.

CSG Computing CSG operations on solids represented by their parametric surface boundaries involves computing the curve of intersection between pairs of parametric surfaces. The resulting curve can be projected into the respective parameter spaces of the two surfaces, and used to perform trimming operations. The curve of intersection is an implicit curve solving a system of three equations in four variables, of the form

$$\underset{\substack{(u,v) \in D_S \\ (r,s) \in D_T}}{\text{SOLVE}} (S(u,v) = T(r,s))$$

where S and T are the two intersecting parametric surfaces. This problem is similar to the one presented for interference detection, except that an approximation of the solution is desired rather than a mere indication of solution existence. Such an approximation can be computed using the algorithm in Section 4, which is built on the SOLVE algorithm. Figure 8 shows the results of CSG operations computed in this way.

2 Inclusion Functions

The interval analysis approach to solving global problems works by recursively subdividing an initial hyper-rectangle of the parameter space of the global problem. Inclusion functions are used to test whether a particular region satisfies the constraints (constraint and minimization problems), contains points with a small enough value of the objective function (minimization problem), or satisfies the set constraint (partitioning problem), by computing a bound on the function over the region. For example, to test whether a region X includes a solution to the equation $f(x) = 0$, an inclusion function for f is evaluated over the region X. If the resulting bound on f does not contain 0, then X may be rejected. The following section defines inclusion functions more precisely, discusses some of their properties, and explains how they may be implemented.

2.1 Terminology and Definitions

An *interval*, $A = [a, b]$, is a subset of $\mathbf{R}$ defined as

$$[a, b] \equiv \{x \mid a \leq x \leq b,\ x, a, b \in \mathbf{R}\}$$

The numbers a and b are called the *bounds* of the interval; a is called the *lower bound*, written lb $[a, b]$, and b, the *upper bound*, written ub $[a, b]$. The symbol $\mathbf{I}$ denotes the set of all intervals.

A *vector-valued interval of dimension* n, $A = (A_1, A_2, \ldots, A_n)$, is a subset of $\mathbf{R}^n$ defined as

$$A \equiv \{x \mid x_i \in A_i \text{ and } A_i \in \mathbf{I} \text{ for } i = 1, 2, \ldots, n\}$$

For example, a vector-valued interval of dimension 2 represents a rectangle in the plane, while a vector-valued interval of dimension 3 represents a "brick" in 3D space. An interval A_i that is a component of a vector-valued interval is called a *coordinate interval of* A. The symbol $\mathbf{I}^m$ denotes the set of all vector-valued intervals of dimension m. Hereafter, we will use the term interval to refer to both intervals and vector-valued intervals; the distinction will be clear from the context.

The *width* of an interval, written $w([a, b])$, is defined by

$$w([a, b]) \equiv b - a$$

Similarly, the width of a vector-valued interval, $A \in \mathbf{I}^n$, is defined as

$$w(A) \equiv \max_{i=1}^{n} w(A_i)$$

Given a subset D of $\mathbf{R}^m$, let $\mathbf{I}(D)$ be defined as the set of all intervals that are subsets of D:

$$\mathbf{I}(D) \equiv \{\, Y \mid Y \in \mathbf{I}^m \text{ and } Y \subseteq D\}$$

Let $f : D \rightarrow \mathbf{R}^n$ be a function. An *inclusion function* for f, written $\Box f$, is a function $\Box f : \mathbf{I}(D) \rightarrow \mathbf{I}^n$ such that

$$x \in Y \Rightarrow f(x) \in \Box f(Y) \quad \forall Y \in \mathbf{I}(D)$$

In other words, $\Box f$ is a vector-valued interval bound on the range of f over a vector-valued interval bound on its domain. Many possible inclusion functions may be defined for a given function f, each having different properties. For example, an inclusion function $\Box f$ is called *convergent* if

$$w(X) \rightarrow 0 \;\Rightarrow\; w(\Box f(X)) \rightarrow 0$$

Note that f must be continuous for its inclusion function to be convergent.

2.2 Inclusion Functions for Arithmetic Operators

To see how inclusion functions can be evaluated on a computer, let us first consider functions defined using arithmetic operations. Let g and h be functions from $\mathbf{R}^m$ to $\mathbf{R}$, and let $X \in \mathbf{I}^m$. Let inclusion functions for g and h be given and evaluated on the interval X

$$\begin{aligned} \Box g(X) &= [a, b] \\ \Box h(X) &= [c, d] \end{aligned}$$

Given these interval bounds on g and h, we can bound an arithmetic combination, $g \star h$, where $\star$ represents addition, subtraction, multiplication or division. This bound may be computed by bounding the set $Q_\star$, defined as

$$Q_\star \equiv \{x \star y \mid x \in [a, b], y \in [c, d]\}$$

$Q_\star$ can be bounded within an interval using the well-known technique of *interval arithmetic*, which defines the operators $+_\Box$, $-_\Box$, $*_\Box$, and $/_\Box$ according to the rules

$$\begin{array}{lclcl} [a, b] +_\Box [c, d] & \equiv & Q_+ & \equiv & [a + c, b + d] \\ [a, b] -_\Box [c, d] & \equiv & Q_- & \equiv & [a - d, b - c] \\ [a, b] *_\Box [c, d] & \equiv & Q_* & \equiv & [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\ [a, b] /_\Box [c, d] & \equiv & Q_/ & \equiv & \left[\min(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}), \max(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d})\right] \\ & & & & \quad \text{provided } 0 \notin [c, d] \end{array}$$

The inclusion functions defined above rely on an infinitely precise representation for real numbers and arithmetic operations. To perform interval analysis on a computer, an interval $A = [a, b]$ must be approximated by a *machine interval* $A_M = [a_M, b_M]$ containing A, so that a_M and b_M are members of the machine's set of floating point numbers. We can not assume that an inclusion function for $g + h$ can be constructed by producing the interval $[a +_M c, b +_M d]$, where $+_M$ denotes the hardware addition operator. Because of addition rounding errors, $a +_M c$ may not be a lower bound for $a + c$. This problem can be solved on machines that conform to the IEEE floating point standard using using round-to-$-\infty$ mode for computation of interval lower bounds, and round-to-$+\infty$ mode for interval upper bounds.

2.3 Natural Interval Extensions

It is clear that the interval bounds of the previous section can be recursively applied to yield an inclusion function for an arbitrary, nested combination of arithmetic operators on a set of functions with known inclusion functions. For example, an inclusion function for $f + (g + h)$ is given by

$$\Box(f + (g + h)) \equiv \Box f +_\Box (\Box g +_\Box \Box h) \tag{3}$$

Furthermore, this notion can be extended to non-arithmetic operators. For each operator, $P(f_1, f_2, \ldots, f_n)$, that produces a function given n simpler functions, we must define a method, $P_\Box$, that evaluates an inclusion function for P, depending only on the interval results of the inclusion functions $\Box f_i$. Let each of the functions f_i be defined on a domain D and let $X \in \mathbf{I}(D)$. Given $P_\Box$, an inclusion function for $P(f_1, \ldots, f_n)$ is then given by

$$\Box P(f_1, f_2, \ldots, f_n)(X) \equiv P_\Box(\Box f_1(X), \Box f_2(X), \ldots, \Box f_n(X))$$

In a generalization of Equation 3 , given a set of operators, $P_1, P_2, \ldots, P_N$, an inclusion function can be evaluated for any function formed by their composition (e.g., $P_1(P_2(f_1, f_2), P_3(f_3))$). Inclusion functions constructed in this way are called *natural interval extensions*.

Construction of an operator's inclusion function method may not be difficult if the operator's monotonicity intervals are known. For example, an inclusion function evaluation method can be defined for the cosine operator, based on the observation that the cosine function is monotonically decreasing in the interval $[\ \pi 2n, \pi(2n + 1)]$, and monotonically increasing in the interval $[\pi(2n + 1), \pi(2n + 2)]$, for integer n. Let f be a function from $\mathbf{R}^m$ to $\mathbf{R}$, and let $X \in \mathbf{I}^m$. Let an inclusion functions for f be given and evaluated on the interval X, yielding the interval $[a, b]$. An inclusion function for $\cos(f)$ can be evaluated on X according to the following rules:

$$\cos_\Box([a, b]) \equiv \begin{cases} [-1, 1], & \text{if } 1 + \left\lceil \frac{a}{\pi} \right\rceil \leq \frac{b}{\pi} \\ [-1, \max(\cos(a), \cos(b))], & \text{if } \left\lceil \frac{a}{\pi} \right\rceil \leq \frac{b}{\pi} \text{ and } \left\lceil \frac{a}{\pi} \right\rceil \bmod 2 = 1 \\ [\min(\cos(a), \cos(b)), 1], & \text{if } \left\lceil \frac{a}{\pi} \right\rceil \leq \frac{b}{\pi} \text{ and } \left\lceil \frac{a}{\pi} \right\rceil \bmod 2 = 0 \\ [\min(\cos(a), \cos(b)), \\ \ \max(\cos(a), \cos(b))], & \text{otherwise} \end{cases}$$

The numerical cosine evaluations implied by $\min(\cos(a), \cos(b))$, for example, must be computed so that they are a lower bound for the theoretical result. Similar inclusion functions can be constructed for operators such as sine, square root, exponential, and logarithm.

Inclusion functions for vector and matrix operations are also easy to construct. For example, an inclusion function method for the dot product operator can be defined via

$$\Box(f \cdot g) \equiv (\Box f_1 *_\Box \Box g_1) +_\Box (\Box f_2 *_\Box \Box g_2) +_\Box \ldots +_\Box (\Box f_n *_\Box \Box g_n)$$

Similarly, interval arithmetic can be used to define inclusion function methods for the matrix multiply, inverse, and determinant operators, and for vector operators like addition, subtraction, length, scaling, and cross product.

2.4 Inclusion Functions for Relational and Logical Operators

Inclusion functions can also be defined for relational and logical operators, allowing natural interval extensions for functions used as constraints.

A relational operator produces a result in the set $\{0, 1\}$, 0 for "false" and

1 for "true". The operators **equal to**, **not equal to**, **less than**, and **greater than or equal to** are all binary relational operators. An inclusion functions for a relational operator, such as **less than**, can easily be defined. Let f and g be functions from $\mathbf{R}^n$ to $\mathbf{R}$, with given inclusion functions, $\Box f$ and $\Box g$. Let $X \in \mathbf{I}^n$, and

$$\begin{aligned} \Box f(X) &= [a, b] \\ \Box g(X) &= [c, d] \end{aligned}$$

Then we have

$$\Box(f < g)(X) \equiv \begin{cases} [0,0], & \text{if } d \leq a \\ [1,1], & \text{if } b < c \\ [0,1], & \text{otherwise} \end{cases}$$

Logical operators, such as **and**, **or**, and **not**, combine results of the relational operators in Boolean expressions. Their inclusion functions are also easily defined. For example, if r_1 and r_2 are two relational functions from $\mathbf{R}^n$ to $\{0,1\}$, and $\Box r_1$ and $\Box r_2$ are their corresponding inclusion functions, then an inclusion function for the logical **and** of the relations, $r_1 \wedge r_2$, is given by

$$\Box(r_1 \wedge r_2) \equiv \begin{cases} [0,0], & \text{if } \Box r_1 = [0,0] \text{ or } \Box r_2 = [0,0] \\ [1,1], & \text{if } \Box r_1 = [1,1] \text{ and } \Box r_2 = [1,1] \\ [0,1], & \text{otherwise} \end{cases}$$

2.5 Mean Value Forms

Given a differentiable function f: $\mathbf{R}^m \rightarrow \mathbf{R}^n$, with parameters $x_1, x_2, \ldots, x_m$, an inclusion function, called the *mean value form*, can be constructed for f as follows:

$$\Box f(Y) \equiv f(c) +_{\Box} \Box f'(Y) \cdot_{\Box} (Y -_{\Box} c) \tag{4}$$

where $c \in Y$, $Y \in \mathbf{I}^m$ and $\Box f'$ is an inclusion function for the Jacobian matrix of f, i.e.,

$$\Box f'(Y) \equiv \left[\Box \frac{\partial f_i}{\partial x_j}(Y) \right]$$

That the above formula represents a valid inclusion function for f is an immediate consequence of Taylor's theorem. The mean value form has the useful property that, under certain conditions, the resulting bound on f quadratically converges to the ideally tight bound as the width of Y shrinks to 0 (Krawczyk-Nickel 1982, for a formal statement and proof, see [SNYD92b]). Note that the addition, subtraction and matrix-vector multiplication operations implied by this definition are computed using interval arithmetic. [5] The ensuing treatment of interval analysis will drop the $\Box$ subscripts for interval arithmetic operations; it should be clear by the context whether the standard operations or their interval analogs are meant.

The idea of a mean value form can be generalized to produce inclusion functions that incorporate more terms of a function's Taylor expansion, called *Taylor forms*. A related inclusion function, called the *monotonicity-test inclusion function*, is also defined using inclusion functions on the partial derivatives of f [MOOR79]). By testing whether these derivatives exclude 0, (i.e., the function is monotonic with respect to a given parameter), very tight bounds can be produced. Mean value forms can also be defined for functions which are only piecewise differentiable (see [RATS88]).

3 Solving Global Problems

3.1 Constraint Solution Algorithm

A system of constraints can be represented as a function, F: $\mathbf{R}^n \rightarrow \mathbf{R}$, that returns a 1 if the constraints are satisfied and a 0 if they are not. Such a function can incorporate both equality and inequality constraints, and can be represented with the relational and logical operators whose inclusion functions were examined in Section 2.4. As discussed in Section 2.4, an inclusion function for F, $\Box F$, over a region $X \subset \mathbf{I}^n$ can take on three possible values:

$$\begin{aligned} \Box F(X) = [0,0] &\quad \Rightarrow \quad X \text{ is an infeasible region} \\ \Box F(X) = [0,1] &\quad \Rightarrow \quad X \text{ is an indeterminate region} \\ \Box F(X) = [1,1] &\quad \Rightarrow \quad X \text{ is a feasible region} \end{aligned}$$

An *infeasible region* is a region in which no point solves the constraint system. A *feasible region* is a region in which every point solves the constraint system. An *indeterminate region* is a region in which the constraint system may or may not have solutions. We now present an algorithm to find solutions to this constraint system.

Algorithm 3.1 (SOLVE) We are given a constraint inclusion function $\Box F$, an initial region, [6] X, in which to find solutions to the constraint problem $F(x) = 1$, and the solution acceptance set constraint, $\Box A$, specifying when an indeterminate region should be accepted as a solution.

```
place $X$ on list $L$
while $L$ is nonempty
    remove next region $Y$ from $L$
    evaluate $\Box F$ on $Y$
    if $\Box F(Y) = [1,1]$ add $Y$ to solution
    else if $\Box F(Y) = [0,0]$ discard $Y$
    else if $\Box A(Y) = [1,1]$ add $Y$ to solution
    else subdivide $Y$ into regions $Y_1$ and $Y_2$,
        and insert into $L$
endwhile
```

Subdivision in Algorithm 3.1 can be achieved by dividing each candidate interval in half along the midpoint of a single dimension. By storing the index of the last subdivided dimension with each region, the algorithm can cyclically subdivide all dimensions of the initial region, ensuring that the width of candidate regions tends to 0 as the number of iterations increases. On the other hand, by knowing properties of the constraint system whose solutions are sought, we can often deduce smaller regions that bound the solutions, especially through the use of interval Newton methods [TOTH85,RATS88,SNYD92b]. The Hansen-Greenberg algorithm is an efficient method for finding zeroes of a function [RATS88] and uses exhaustive subdivision, interval Newton methods, and local Newton methods.

3.1.1 The Problem of Indeterminacy

Algorithm 3.1 finds a set of intervals bounding the solutions to the constraint system. In particular, by the property of inclusion functions, if this algorithm finds no solutions, then the constraint system has no solutions, because a region Y is rejected only when $\Box F(Y)$ shows that it is infeasible. It can also be proved that the constraint solution algorithm converges to the actual solution set, when the inclusion functions used in the equality and inequality constraints are convergent (see, for example, [SNYD92b]).

Unfortunately, a computer implementation of the constraint solution algorithm can not iterate forever; it must terminate at some iteration n and accept the remaining regions as solutions. Especially when equality constraints are used, the algorithm may accept some indeterminate regions, which may contain zero, one, or more solutions, when these regions satisfy the solution acceptance set constraint. This problem is mitigated by several factors.

First, it may be enough to distinguish between the case that the constraint problem possibly has solutions (to some tolerance), and the case that it has no solutions. For example, to compute interference detection between two parametric surfaces, S_1, S_2: $\mathbf{R}^2 \rightarrow \mathbf{R}^3$, a constraint system of three equations in four variables can be solved of the form

$$S_1(u_1, v_1) = S_2(u_2, v_2)$$

If we instead solve the relaxed constraint problem,

$$||S_1(u_1, v_1) - S_2(u_2, v_2)|| < \epsilon$$

the algorithm can hope to produce feasible solution regions, for which the constraints are satisfied for every point in the region. [7] Such relaxed con-

[5] It should also be noted that to implement this mean value form on a computer, the interval $\Box f([c, c])$ should replace $f(c)$. This is because the computer can not *exactly* compute $f(c)$ and must instead bound the result.

[6] The initial region X can be infinite if the technique of *infinite interval arithmetic* is used (see [RATS88]).

[7] Note that the solution to the unrelaxed system is typically a curve of intersection between the two parametric surfaces. Any neighborhood of a point on this curve will also contain points for which the two surfaces do not intersect and hence do not solve the system of equations. The relaxed problem, on the other hand, has solutions for which a neighborhood of small enough size is completely contained within the solution space.

straint problems are called ϵ-collisions in [VONH89]. If any feasible regions are found, the surfaces interfere, within the tolerance. If all regions are eventually found to be infeasible, the surfaces do not interfere within the tolerance, and in fact come no closer than ϵ. It is also possible that only indeterminate regions are accepted as solutions. In this case, we may consider the two surfaces to interfere to the extent that our limited floating point precision is able to ascertain.

Second, we may know a priori that the system has a single solution. Let the solution acceptance set constraint have the simple form

$$\Box A(Y) \equiv \left(w(Y) < \epsilon \right)$$

If the inclusion functions bounding the constraint equality and inequality functions are convergent, then the solution approximation produced by Algorithm 3.1 achieves any degree of accuracy as ϵ goes to 0.

Third, we may be able to compute information about solutions to the constraint system as the algorithm progresses. Section 3.2 presents a theorem specifying conditions computable with interval techniques under which a region contains exactly one zero of a system of equations.

Finally, we can relax the constraints of a constraint system and/or accept indeterminate results of the algorithm. In practice, although we can not guarantee the validity of such results, they are nevertheless useful.

3.1.2 Termination and Acceptance Criteria for Constraint Solution

SOLVE can be applied to five specific problems:

1. find a bound on the set of solutions
2. determine whether a solution exists
3. find one solution
4. find all solutions
5. solve a constrained partitioning problem

The following discussion analyzes the application of Algorithm 3.1 to these specific problems, making the distinction between *heuristic* approaches, in which the results are not guaranteed to be correct, and *robust* approaches, in which the results are guaranteed to be correct.

Algorithm 3.1 never rejects a region unless it contains no solutions to the constraint problem. Therefore, an unmodified Algorithm 3.1 can be used to robustly find a set of regions bounding the solutions to the constraint system. Such a solution superset is often useful in higher-level algorithms, such as the implicit curve approximation algorithm of Section 4. The solution superset can also be visualized to obtain a rough idea of the nature of the solutions, even if the solutions form a multidimensional manifold rather than a finite set of points.

To determine whether a solution exists, if the algorithm terminates with an empty list of solutions, then the algorithm should return the answer "no". If at any point the algorithm finds a feasible region, then the algorithm can immediately terminate with the answer "yes". If the algorithm finds only indeterminate regions, then nothing can be concluded with certainty. A heuristic solution is to return "yes" anyway. This heuristic approach can be made more robust through the choice of an appropriate solution acceptance set constraint. For example, in solving the system $f(x) = 0$ for a continuous function f, it is reasonable that a region, Y, before being accepted as a solution, should satisfy

$$w(\Box f(Y)) < \delta$$

for some small δ. The algorithm should report an error when none of the indeterminate regions satisfy the acceptance criteria, before the machine precision limit is reached during subdivision. A robust solution to the problem can be achieved by testing indeterminate regions for the existence of solutions, using the test of Section 3.2.

To find any single solution to a constraint system, Algorithm 3.1 may conclude that the entire starting region is infeasible, or find a feasible region. In the latter case, any point in the feasible region is chosen as a representative solution and the algorithm is halted. Indeterminate regions are heuristically accepted when they satisfy the solution acceptance set constraint. They may also be tested for the existence of solutions, again using the test of Section 3.2.

Algorithm 3.1 can also be applied to the problem of finding all solutions to a constraint system, when a finite set of solutions is expected. Again, if the algorithm terminates with an empty solution list, there are no solutions. If a feasible region is found, an infinite number of solutions exist. If only indeterminate regions are found, then a useful heuristic approach is to union all contiguous solution regions into a set of mutually disjoint aggregate regions, as shown in Figure 2. A point inside each aggregate region is picked as a solution. If the number of solutions is known beforehand, then the algorithm can be terminated with an error condition if the machine precision is reached in subdivision with a number of aggregated regions unequal to the number of solutions. Of course, we note that although this approach almost always works correctly, it is still heuristic, since, for example, one region may contain no solutions while another contains two. A robust approach is to test for solution existence in each aggregated region. In this case, reaching the machine precision limit during subdivision without being able to verify solution existence should result in an error termination.

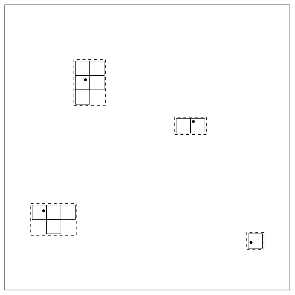


Figure 2: Solution aggregation – The solution regions returned by Algorithm 3.1 are the collection of nondashed squares. The actual solutions are marked by dots. An adequate level of subdivision has been achieved so that sets of contiguous regions encompass each of the four solutions, and each contiguous region may be bounded in an interval (dashed boxes) that is disjoint from other such regions.

Finally, to solve a constrained partitioning problem, Algorithm 3.1 must be slightly modified so that it adds a region to the solution only when $\Box A$ is true, regardless of the value of $\Box F$. Alternatively, the constraint inclusion can be set so that it returns true (i.e., the constant $[1, 1]$) for all regions. The solution acceptance set constraint then becomes the set constraint of the constrained partitioning problem.

3.2 Interval Tests for Solution Existence and Uniqueness

An interesting and useful result can be proved that guarantees the existence of a unique zero of the function $f: \mathbf{R}^n \rightarrow \mathbf{R}^n$ in an interval domain X.

Theorem 3.1 (Bao-Rokne 1987) Let $f: \mathbf{R}^n \rightarrow \mathbf{R}^n$ be continuously differentiable in an interval domain X, and let $c \in X$. Let $\Box J$ be the interval Jacobian matrix of f over X, i.e.,

$$\Box J \equiv \left\{ J \mid J_{ij} \in \Box \frac{\partial f_i}{\partial x_j}(X) \right\}$$

Let Q be the solution set of the linear interval equation in x

$$f(c) + \Box J(x - c) = 0$$

That is,

$$Q \equiv \{ x \mid \exists J \in \Box J \text{ such that } f(c) + J(x - c) = 0 \}$$

If $Q \neq \emptyset$ and $Q \subseteq X$, then f has a unique zero in X.

A proof of this theorem can be found in [SNYD92b]. The hypothesis of the theorem can be verified using practical computations in several ways. First, if the interval determinant of $\Box J$ is not 0, then

$$Q \subset c - \Box J^{-1} f(c)$$

where $\Box J^{-1}$ is the interval matrix inverse of $\Box J$. We can therefore compute the interval inverse of the Jacobian matrix $\square J$, compute

$$Q^* \equiv c - \square J^{-1} f(c)$$

and verify that $X \subset Q^*$, in order to show the existence of a unique solution in X. Other methods involve Gauss-Sidel iteration on the linear equation [RATS88], or use of linear optimization [SNYD92b].

We note that the theorem is not useful in every case, since if there is a zero of f in X at p, and the determinant of the Jacobian of f at p is 0, then we can never verify solution uniqueness using this theorem. We also note that an interval test for solution existence (but not necessarily uniqueness) can be found in [MOOR80]. The appendix discusses a test that indicates when a region has **at most** one zero.

3.3 Minimization Algorithm

The constrained minimization problem involves finding the global minimum (or global minimizers) of a function $f\colon \mathbf{R}^n \to \mathbf{R}$ for all points that satisfy a constraint function $F\colon \mathbf{R}^n \to \{0, 1\}$. This constraint function is defined exactly as in Section 3.1.

Algorithm 3.2 (MINIMIZE) We are given a constraint inclusion function $\square F$, a solution acceptance set constraint, $\square A$, an inclusion function for the objective function, $\square f$, and an initial region, X. The variable u is a progressively refined least upper bound for the value of the objective function f evaluated at a feasible point. Regions are inserted into the priority queue L so that regions with a smaller lower bound on the objective function f have priority.

```
place $X$ on priority queue $L$
initialize upper bound $u$ to $+\infty$
while $L$ is nonempty
    get next region $Y$ from $L$
    if $\square A(Y) = [1, 1]$ add $Y$ to solution
    else
        subdivide $Y$ into regions $Y_1$ and $Y_2$
        evaluate $\square F$ on $Y_1$ and $Y_2$
        if $\square F(Y_i) = [0, 0]$ discard $Y_i$
        evaluate $\square f$ on $Y_1$ and $Y_2$
        if lb $\square f(Y_i) > u$ discard $Y_i$
        insert $Y_i$ into $L$ according to lb $\square f(Y_i)$
        if $Y_i$ contains an identified feasible point $q$
            $u = \min(u, f(q))$
        else if $Y_i$ contains an unidentified feasible point
            $u = \min(u, \text{ub}\,\square f(Y_i))$
        endif
    endif
endwhile
```

Let the region U_n^i be the i-th region on the priority queue L after n while loop iterations of the algorithm. Let u_n be the value of u at iteration n, and let l_n be given by

$$l_n \equiv \text{lb}\,\square f(U_n^1)$$

The interval U_n^1 is called the *leading candidate interval*, and has the smallest lower bound for the value of f. Let f^* be the minimum value of the objective function subject to the constraints. We note that if a region X contains feasible points for the constraint function F, then f^* exists. Given existence of a feasible point, an important property of Algorithm 3.2 is

$$l_n \leq f^* \leq u_n \quad \forall n$$

Algorithm 3.2 suffers the same problems that Algorithm 3.1 does, in that an indeterminate region (i.e., a region Y for which $\square F(Y) = [0, 1]$), may or may not include feasible points of the system of constraints. This implies that the algorithm may accept indeterminate regions as solutions that are, in fact, infeasible. Moreover, if the constraints can never be satisfied exactly, (e.g., they are represented using equality constraints), then all candidate regions are indeterminate, so that u is never updated. In this case, the algorithm is unable to reject any of the candidate regions on the basis of the objective function bound and accepts all indeterminate regions as solutions.

A robust solution to this problem is to use an existence test, such as the one presented in Section 3.2, to verify that a region contains at least one feasible point. A heuristic approach is to consider indeterminate regions of small enough width as if they contained a feasible point. These indeterminate regions may be subjected to an appropriate acceptance test that provides more confidence that the region contains a feasible point.

Algorithm 3.2 can be enhanced with techniques that find feasible points, feasible points with a smaller value of the objective function, or feasible regions in which the objective function is monotonic with respect to any input variable [RATS88].

3.3.1 Termination and Acceptance Criteria for Minimization

A constrained minimization problem can be "solved" in three ways:

1. find the minimum value of the objective function
2. find one feasible point that minimizes the objective function
3. find all feasible points that minimize the objective function

Slight modifications to Algorithm 3.2 regarding when the algorithm is halted and when indeterminate regions are accepted as solutions can make it applicable to each of these specific subproblems.

To find the minimum value of the objective function, f^*, Algorithm 3.2 should be terminated when a leading candidate interval, U_n^1, is encountered with $w(\square f(U_n^1))$ sufficiently small, given that U_n^1 contains at least one feasible point. [8] In this case, the value $f(q)$ should be returned for some $q \in U_n^1$. This approach is justified because if U_n^1 contains a feasible point then

$$\text{lb}\,\square f(U_n^1) \leq f^* \leq \text{ub}\,\square f(U_n^1)$$

This approach presumes that we can verify the presence of a feasible point in an indeterminate region before the machine precision is reached in subdivision. Lack of this verification should result in some form of error termination. A heuristic approach is to accept indeterminate regions of small enough width (and, possibly, satisfying other criteria) as though they contained a feasible point.

Finding one or all minimizers of the objective function is a difficult problem that is currently not amenable to completely robust solution. Under certain conditions [9], Algorithm 3.2 converges, in a theoretical sense, to the set of global minimizers of the minimization problem. In practice however, we obtain a bound on the set of global minimizers after a finite number of iterations. Although techniques exist to verify whether a given interval in this bound contains a local minimizer of the minimization problem, we will not know, in general, if these local minimizers are also global minimizers.

If we know, a priori, that a single global minimizer exists, then the technique of solution aggregation (Section 3.1.2) can be used to collect candidate solutions into a single interval. We can then verify that the width of this interval tends to zero as the algorithm iterates. If we expect a finite set of global minimizers, then a reasonable heuristic approach is to aggregate solutions, and pick a point in each aggregated region as a global minimizer. Such an aggregated region should be small enough in width and satisfy other acceptance criteria that increase confidence that it contains a global minimizer.

4 Example: Approximating Implicit Curves

An implicit curve is the solution to a constraint system $F(x) = 1$, $x \in X \subset \mathbf{R}^n$, such that the solution forms a 1D manifold. Implicit curves are extremely useful in geometric modeling, especially for CSG and trimming operations on parametrically described shapes. They can represent, for example, the intersection of two parametric surfaces in $\mathbf{R}^3$, or the silhouette edges of a parametric surface in $\mathbf{R}^3$ with respect to a given view.

The robustness of the algorithm presented here is superior to local methods such as [TIMM77,BAJA88]. Timmer's method, for example, separates implicit curve approximation into a hunting phase, where intersections of the implicit curve with a preselected grid are computed, and a tracing phase, where the curve inside each grid cell is traced to determine how to connect the intersections.

The new algorithm computes points on the implicit curve using Algorithm 3.1, guaranteeing a bound on the result. This method is superior to

[8] If all candidate intervals are rejected, then no feasible points exist in the original region, so f^* does not exist.

[9] A sufficient condition is the existence of a sequence of points in the interior of the feasible domain that converges to a global minimizer [RATS88].

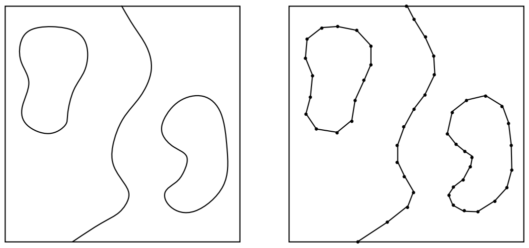


Figure 3: Implicit curve approximation – The figure on the left shows an implicit curve satisfying the algorithm's assumptions. It consists of three segments: two closed segments, and one segment intersecting the boundary of the interval of consideration. The figure on the right shows an approximation of the implicit curve. In this case, the algorithm produces three linked lists of points as output, one for each segment of the implicit curve.

local methods, such as Newton iteration, which are not guaranteed to converge. Timmer's method also fails to find a disjoint segment of the curve if it lies completely within one grid cell, while the proposed algorithm uses a global parameterizability criterion that subdivides parameter space until no curve segment can be lost. The algorithm is similar to the one described in [SUFF90], but differs in three respects: it uses this global parameterizability criterion, it handles multivariate implicit curves, and it incorporates an approximation quality metric.

4.1 An Implicit Curve Approximation Algorithm

The following are inputs to the approximation algorithm:

1. an interval $X \in \mathbf{I}^n$, called the *interval of consideration*, in which to approximate the implicit curve.
2. an inclusion function $\Box F(Y)$, $Y \in \mathbf{I}(X)$ for the constraint system defining the implicit curve.
3. an inclusion function $\Box A(Y)$, $Y \in \mathbf{I}(X)$, called the approximation acceptance inclusion function. This inclusion function tells when an interval Y is small enough that each segment of the implicit curve it contains can be approximated by a single interpolation segment between a pair of solution points.

The algorithm works by subdividing the region X into subregions, called *proximate intervals*, that contain the implicit curve, satisfy the approximation acceptance inclusion function, and allow simple computation of the local topology of the curve. The algorithm makes the following assumptions:

1. The solution to the constraint system $F(x) = 1$ is a continuous, 1D manifold. This implies that the solution contains no self-intersections, isolated singularities, or solution regions of dimensionality greater than 1. It further implies that each disjoint curve segment of the solution is either closed or has endpoints at the boundary of the region X.
2. The intersection of the solution curve with a proximate interval's boundaries is either empty or a finite collection of points, not a 1D manifold. This assumption is unimportant for implicit curves with no segments entirely along the parametric axes. When the implicit curve does have such segments, the constraint system must be reposed (often simply by a linear transformation of the parametric coordinates) as discussed in [SNYD92b].

Under these assumptions, each point on the implicit curve is linked to two neighbors, or possibly a single neighbor if the point is on the boundary of X. The output of the approximation algorithm is a list of "curves", where each curve is a linked list of points on a single, disjoint segment of the implicit curve, as shown in Figure 3.

Algorithm 4.1 (Implicit Curve Approximation)

1. **Subdivide X into a collection of proximate intervals bounding the implicit curve and satisfying the approximation acceptance inclusion function.** This can be accomplished using Algorithm 3.1. Figure 4 shows an example of a collection of proximate intervals.
2. **Check each proximate interval for global parameterizability.** The implicit curve contained in a proximate interval Y is called *globally parameterizable in a parameter i* if there is at most one point in Y on the curve for any value of the i-th parameter (see Figure 5). If the implicit curve is not globally parameterizable in Y for any parameter, then Y is recursively subdivided and tested again.
3. **Find the intersections of the implicit curve with the boundaries of each proximate interval, using Algorithm 3.1.** Assumption 2 implies that this intersection will be empty or a finite collection of points.
4. **Ensure that the boundary intersections are disjoint in the global parameterizability parameter.** Let i be the global parameterizability parameter for a proximate interval Y, computed from Step 2. This step checks that intersections of the implicit curve with Y's boundary are non-overlapping in coordinate i, as shown in Figure 6, so that they can be unambiguously sorted in increasing order of coordinate i.

   If Y's boundary intersections are not disjoint in parameter i, Y is recursively subdivided and retested.
5. **Compute the connection of boundary intersections in each proximate interval.** If an interval Y contains no boundary intersections, it can be discarded, because the global parameterizability condition implies that the solution cannot be a closed curve entirely contained in Y. Nor can the solution be a curve segment that does not intersect Y's boundary, by Assumption 1. If Y contains a single boundary intersection, then the solution is either tangent to a boundary of Y or passes through a corner of Y, but does not intersect the interior of Y.

   If Y contains more than one boundary intersection, the boundary intersections are sorted in order of the global parameterizability parameter i. For each pair of boundary intersections adjacent in parameter i, Algorithm 3.1 is used to see if the solution curve intersects the i-th parameter hyperplane midway between the two boundary intersections, as shown in Figure 7. If so, the boundary intersections are connected in the local curve topology linked list.
6. **Find the set of disjoint curve segments comprising the implicit curve.** After the implicit curve has been traced inside of each proximate interval, the list of connected boundary intersections is traversed, using the following algorithm

```
let $S$ be the set of boundary intersections
while $S$ is nonempty
        remove an intersection point $P$ from $S$
        find and remove all points $Q$ in $S$ that are
                (indirectly) connected to $P$
        associate $P$ and the set $Q$ with a new curve
endwhile
```

   We note that in accumulating the set of points on a particular curve using this algorithm, if a point $P' \in Q$ is eventually found such that $P = P'$ then the curve is closed. Otherwise, the curve has two endpoints on the boundary of X by Assumption 1.

Step 1 of the algorithm combines the constraint inclusion with the approximation acceptance inclusion to create an initial collection of proximate intervals bounding the implicit curve (subproblem 1 in Section 3.1.2). Step 2 ensures that each proximate interval satisfy a global parameterizability criterion. The appendix presents a theorem identifying conditions for global parameterizability, computable with interval techniques already discussed. This theorem pertains to the special case of a system of $n - 1$ continuously differentiable equality constraints in n parameters. We have also developed a more general but heuristic test for global parameterizability, discussed in [SNYD92b].

Step 3 of the algorithm computes the intersections of the implicit curve with the boundary of each proximate interval. Algorithm 3.1 is used with the original constraint inclusion, $\Box F$, and an initial region formed by one of the $2n$ $(n - 1)$-dimensional hyperplanes bounding the proximate interval. For each boundary hyperplane, Algorithm 3.1 searches for all the constraint system's solutions, producing a set of intervals bounding the solutions, called *boundary intersection intervals*. Boundary intersection intervals that are shared along edges or corners of contiguous proximate intervals should be merged, as discussed in [SNYD92b].

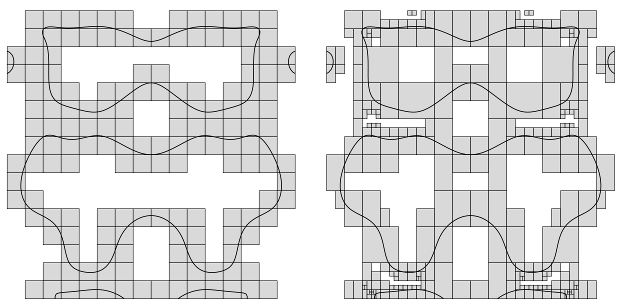
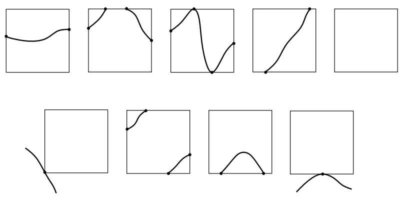


Figure 4: Collection of proximate intervals bounding an implicit curve – In these examples, the constraint system is given by the equation

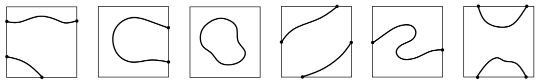
$$x^2 + y^2 + \cos(2\pi x) + \sin(2\pi y) + \sin(2\pi x^2)\cos(2\pi y^2) = 1$$

The interval of consideration is $[-1.1, 1.1] \times [-1.1, 1.1]$. The approximation acceptance inclusion function for the left example simply requires that the width of the parameter space interval should be less than 0.2, while that on the right guarantees the global parameterizability of the solution in each interval.

I. Examples Globally Parameterizable in x



II. Examples Not Globally Parameterizable in x



III. Examples Not Allowed by Assumptions

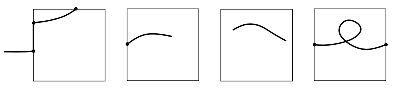


Figure 5: Global parameterizability – The figure illustrates some of the possible behaviors of an implicit curve in an interval.

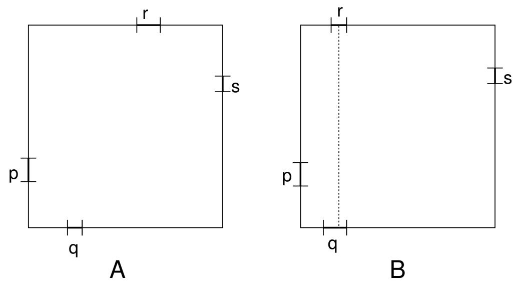
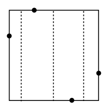
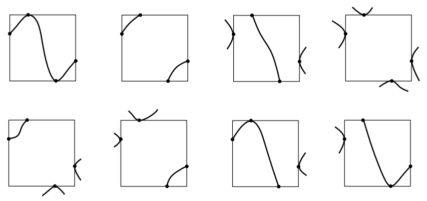


Figure 6: Boundary intersection sortability – Figure A illustrates a 2D interval containing four boundary intersections that are disjoint in the x parameter (horizontal axis). They can therefore be sorted in x, yielding the ordering p, q, r, s. In figure B, boundary intersections q and r are not disjoint in x (the dashed line shows a common x coordinate).

I. Boundary Intersections of an Implicit Curve



II. Eight Cases of Implicit Curve Behavior



III. Not Allowed by Global Parameterizability

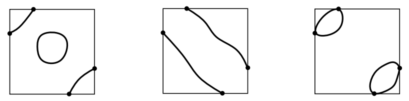


Figure 7: Global parameterizability and the linking of boundary intersections – In this figure, we assume an implicit curve defined in $\mathbf{R}^2$ is globally parameterizable in x in an interval. The implicit curve has four intersections with the interval's boundary, as shown in I. Because of global parameterizability and the curve approximation algorithm's assumptions, there are only eight possible ways the implicit curve can connect the boundary intersections, as shown in II. The possibilities shown in III are not globally parameterizable in x, and are therefore excluded. To disambiguate between these eight cases, we need only see if the implicit curve intersects the x hyperplane (dashed vertical line in I) between each pair of adjacent boundary intersections.

Steps 4 and 5 link boundary intersection intervals that are connected by the same segment of the implicit curve. Boundary intersection intervals are sorted in the global parameterizability parameter, and each pair of adjacent intersections is tested. The test uses Algorithm 3.1 to discover whether the implicit curve intersects a hyperplane midway between the pair of intersections. This application of the constraint algorithm need only ascertain whether a solution exists; the location of the intersection point is not required. On the other hand, the intersection point can be used to better approximate the implicit curve's behavior between the boundary intersections, at little extra computational cost.

Finally, after all proximate intervals have been examined, Step 6 associates each of the boundary intersection intervals with a disjoint segment of the implicit curve. A point inside each of the boundary intersection intervals should be chosen to represent the actual point of intersection of the proximate interval's boundary with the implicit curve. This point can be chosen arbitrarily (e.g., midpoint of the interval) or computed using a local iterative technique such as Newton's method.

We note that an algorithm similar to Algorithm 4.1 can be used to generate approximations of implicit surfaces [SNYD92a]. This algorithm also uses the global parameterizability criterion described in the appendix, for the case of a 2D manifold rather than a 1D manifold.

5 Results

Figures 8 through 11 illustrate the results of the interval analysis algorithms. Running times for the examples ranged from about 5 seconds for the computation of the minimum distance between two parametric surfaces (Figure 11) to several minutes for the CSG example (Figure 8) on a HP9000 Series 835 Workstation.

6 Conclusions

We have shown how a variety of important problems in computer graphics can be solved using the technique of interval analysis. These problems include ray tracing, computation of toleranced polygonal decompositions, detection of collisions, computation of CSG operations, approximation of silhouette curves, and many others. We have described two general algorithms, constraint solution and constrained minimization, which can solve these problems either directly, or when used in a higher level algorithm such as the implicit curve approximation algorithm of Section 4.

The advantage of the approach advocated here is twofold. Robust solution of computer graphics problems is achieved because interval analysis controls numerical error. A simple implementation is achieved because only two basic algorithms are necessary, which require inclusion functions for functions relevant to the problem. Definition of inclusion functions is not difficult; natural interval extensions, a particular type of inclusion function, can be defined by implementing an inclusion function method for each operator used in the relevant functions (e.g., the arithmetic operators and the cosine operator of Section 2.3). Mean value forms, another type of inclusion function, can be defined using natural interval extensions and a derivative operator. An entire, very powerful geometric modeling system can be built upon a set of operators each having an inclusion method, such as the system described in [SNYD92a,SNYD92b].

Acknowledgments

I would like to thank Al Barr for his support and encouragement of the publication of this research. Al Barr and Ronen Barzel have provided many helpful comments and suggestions. This work was funded, in part, by IBM, Hewlett-Packard, and the National Science Foundation.

References

[ALEF83] Alefeld, G., and J. Herzberger, *Introduction to Interval Computations,* Academic Press, New York, 1983.

[BAJA88] Bajaj, C., C. Hoffman, J. Hopcroft, and R. Lynch, "Tracing Surface Intersections," *Computer Aided Geometric Design,* 5, 1988, pp. 285-307.

[KALR89] Kalra, Devendra, and Alan H. Barr, "Guaranteed Ray Intersections with Implicit Surfaces," *Computer Graphics,* 23(3), July 1989, pp. 297-304.

[MITC90] Mitchell, Don, "Robust Ray Intersections with Interval Arithmetic," Proceedings Graphics Interface '90, May 1990, pp. 68-74.

[MITC91] Mitchell, Don, "Three Applications of Interval Analysis in Computer Graphics," Course Notes for Frontiers in Rendering, Siggraph '91.

[MOOR66] Moore, R.E., *Interval Analysis,* Prentice Hall, Englewood Cliffs, New Jersey, 1966.

[MOOR79] Moore, R.E., *Methods and Applications of Interval Analysis,* SIAM, Philadelphia.

[MOOR80] Moore, R.E., "New Results on Nonlinear Systems," in *Interval Mathematics 1980*, Karl Nickel, ed., Academic Press, New York, 1980, pp. 165-180.

[MUDU84] Mudur, S.P., and P.A. Koparkar, "Interval Methods for Processing Geometric Objects," *IEEE Computer Graphics and Applications,* 4(2), Feb, 1984, pp. 7-17.

[RATS88] Ratschek, H. and J. Rokne, *New Computer Methods for Global Optimization,* Ellis Horwood Limited, Chichester, England, 1988.

[SEGA90] Segal, Mark, "Using Tolerances to Guarantee Valid Polyhedral Modeling Results," *Computer Graphics,* 24(4), August 1990, pp. 105-114.

[SNYD91] Snyder, John, *Generative Modeling: An Approach to High Level Shape Design for Computer Graphics and CAD,* Ph.D. Thesis, California Institute of Technology, 1991.

[SNYD92a] Snyder, John, "Generative Modeling: A Symbolic System for Geometric Modeling," to be published in Siggraph '92.

[SNYD92b] Snyder, John, *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design Using Interval Analysis*, to be published by Academic Press, summer 1992.

[SUFF90] Suffern, Kevin G., and Edward Fackerell, "Interval Methods in Computer Graphics," Proceedings of Ausgraph '90, Melbourne, Australia, 1990, pp. 35-44.

[TIMM77] Timmer, H.G., *Analytic Background for Computation of Surface Intersections,* Douglas Aircraft Company Technical Memorandum CI-250-CAT-77-036, April 1977.

[TOTH85] Toth, Daniel L., "On Ray Tracing Parametric Surfaces," *Computer Graphics,* 19(3), July 1985, pp. 171-179.

[VONH87] Von Herzen, Brian P. and Alan H. Barr,"Accurate Sampling of Deformed, Intersecting Surfaces with Quadtrees," *Computer Graphics*, 21(4), July 1987, pp. 103-110.

[VONH89] Von Herzen, Brian P., *Applications of Surface Networks to Sampling Problems in Computer Graphics,* Ph.D. Thesis, California Institute of Technology, 1989.

[VONH90] Von Herzen, B., A.H. Barr, and H.R. Zatz, "Geometric Collisions for Time-Dependent Parametric Surfaces," *Computer Graphics,* 24(4), August 1990, pp. 39-48.

Appendix – A Robust Test for Global Parameterizability

Consider an r-dimensional manifold defined as the solution to a system of $n - r$ equations in n parameters ($r \in \{0, 1, \ldots, n-1\}$):

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_{n-r}(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

Given a set of r parameter indices, $A = \{k_1, k_2, \ldots, k_r\}$, and an interval $X \in \mathbf{I}^n$, we define a *subinterval* of X over A as a set depending on r parameters $(y_1, y_2, \ldots, y_r)$, $y_i \in X_{k_i}$, defined by

$$\left\{ x \in X \,\middle|\, \begin{array}{ll} x_i = y_j & \text{if } i = k_j \in A \\ x_i \in X_i & \text{otherwise} \end{array} \right\}$$

Thus, a subinterval is an interval subset of X, r of whose coordinates are a specified constant, and the rest of whose coordinates are the same as in X.

The solution to a system of $n - r$ equations in n parameters is called *globally parameterizable* in the r parameters indexed by A over an interval X if there is at most one solution to the system in any subinterval of X over A. Put more simply, the system of equations is globally parameterizable if r parameters can be found such that there is at most one solution to the system for any particular value of the r parameters in the interval.

We define $\Box J_{\{k_1,k_2,\ldots,k_r\}}(X)$, called the *interval Jacobian submatrix*, as an $(n-r) \times (n-r)$ interval matrix given by

$$\Box J_{\{k_1,k_2,\ldots,k_r\}}(X) \equiv \left[\Box \frac{\partial f_i}{\partial x_j}(X) \right]_{j \notin \{k_1,k_2,\ldots,k_r\}}$$

For an $n \times n$ interval matrix $\Box M$, we write $\det \Box M \neq 0$ if there exists no matrix $M \in \Box M$ such that $\det M = 0$. The following theorem guarantees the global parameterizability of the solution in an interval X (for a proof, see [SNYD92b]).

Theorem A.1 (Interval Implicit Function Theorem) Let the constraint functions $f_i(x)$, $i = 1, 2, \ldots, n-r$ be continuously differentiable. Let a region $X \in \mathbf{I}^n$ exist such that

$$\det \Box J_{\{k_1,k_2,\ldots,k_r\}}(X) \neq 0$$

Then the solution to the system of equations $f_i(x) = 0$ is globally parameterizable in the r parameters indexed by $\{k_1, k_2, \ldots, k_r\}$ over X.

In the case of approximation of a 1D solution manifold, $r = 1$; i.e., a system of $n - 1$ equations in n variables is to be solved. The theorem guarantees that if, in an interval X, we can find $n - 1$ parameters such that

$$\det \Box J_{\{k\}}(X) = \det \left[\Box \frac{\partial f_i}{\partial x_j} \right]_{j \neq k} \neq 0$$

then the solution manifold is globally parameterizable in X over the parameter x_k, and thus satisfies the constraint of Step 2. We can verify that

$$\det \Box J_{\{k\}}(X) \neq 0$$

by forming an inclusion function for the determinant of any of the n interval Jacobian submatrices using the interval arithmetic presented in Section 2.2.

Figure 8: CSG Example – Algorithm 4.1 was used to find the curve of intersection between a bumpy sphere surface and a cylinder surface. The output of the algorithm was used in a parametric trimming operation, resulting in the subtraction of the cylinder from the bumpy sphere on the left, and the subtraction of the bumpy sphere from the cylinder on the right.

Figure 9: Silhouette Edge Detection Example – The figures show the results of the implicit curve approximation algorithm to approximate the silhouette curve of a parametric surface, $S(u, v)$, with respect to a given (in this case, orthographic) view. The implicit curve is the solution in two variables, u and v, of the equation $E \cdot (\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}) = 0$ where $S(u, v)$ is the parametric surface and E is the viewing direction.

Figure 10: Polygonal Decomposition Example – The figure on the left shows polygonal decomposition based on uniform sampling in parameter space. On the right, the same surface has been decomposed using a slightly smaller number of triangles, using the constrained partitioning algorithm, which subdivides the parameter space (shown below the two surfaces) until the maximum variation in the surface normal is below a threshold. Polygonal artifacts on the highly curved projection are much reduced.

Figure 11: Minimum Distance Computation Example – The results of the minimization algorithm to find the minimum distance between two parametric surfaces is displayed. The yellow line connects the points on the two surfaces closest to each other. In this case, a single global minimizer was found for the unconstrained minimization problem of Formula 2 in Section 1.2.