The Visual Computer manuscript No.
(will be inserted by the editor)

Bill Dean · Baker Kearfott

Reliable Ray Tracing by Linear Interval Estimations

Abstract Recently Bühler has introduced a new type of bounding volume for parametric surfaces. These bounding volumes, which Bühler calls Linear Interval Estimations (LIEs), are parametrized with the same parametrization as the surfaces they bound. We report on the application of LIEs to the ray tracing of Bezier surfaces. The general nonlinear ray–surface intersection problem is replaced with a linear ray–LIE intersection method. A new ray–tracing algorithm is developed which is relatively simple and compares favorably to the method of Toth, being roughly comparable in speed and yet more reliable.

Keywords computational geometry · ray tracing · affine arithmetic · interval arithmetic · robust computing

Mathematics Subject Classification (2000) 65D18 · 65G30 · 65G20

1 Introduction

The recent introduction of Linear Interval Estimations (LIEs) by Bühler [1] makes possible a new ray–tracing algorithm for Bezier surfaces which is reliable and yet relatively simple. For a given subsurface a LIE can be computed which is a bounding volume parametrized with the same parametrization as the subsurface itself. The solution in parameter space to the intersection of a ray with the LIE when computed using interval arithmetic gives bounds on the ray-surface intersection solution. Only that part of the ray-LIE intersection solution which is within the original subsurface domain needs to be considered. The remainder is pruned away to attain that portion of the original domain that may contain a ray-surface intersection. Iterating the sequence of ray-LIE intersection followed by pruning constitutes a strategy to determine the ray-surface intersection solution. The use of interval and affine arithmetics gives this process reliability. After a brief summary of interval and affine arithmetics detail on the algorithm is developed.

Mathematics Department, University of Louisiana, Box 4-1010, Lafayette Louisiana 70504-1010, USA
E-mail: wmd5332@hotmail.com, rbk@louisiana.edu

2 Notation

Unless stated otherwise, a lower case letter b represents a real number, and an upper case letter B represents an interval. Bold face lower case letters $\mathbf{b}$ represent real vectors, and bold face upper case letters $\mathbf{B}$ represent interval vectors.

3 Self-validated Computation

A computation is referred to as self-validated if, as part of the process of computation, the accuracy of the result is somehow specified. Interval analysis and affine arithmetic are two models for self-validated computation.

3.1 Interval Arithmetic

A range-based model uses ranges as approximate values. As discussed in [6] interval arithmetic is a range-based model for self-validated computation in which an interval $X = [x_1, x_2]$ represents a quantity x when x is guaranteed to lie within the range of values defined by X.

Each interval is specified by its bounds.

$$X = [x_1, x_2] = \{x \in \mathbb{R} \mid x_1 \leqq x \leqq x_2\}.$$

A midpoint and radius are also defined for each interval,

$$X = [a, b], \quad \mathrm{mid}(X) = \frac{a+b}{2}, \quad \mathrm{rad}(X) = \frac{b-a}{2}.$$

The starting point for an interval computation is a set of intervals known to contain the true values of the input variables. Interval arithmetic operations and function evaluations are defined in such a way that the result of the computation is always an interval guaranteed to contain the result of the ideal computation.

Specifically, for $\circ \in \{+, -, \times, \wedge\}$ and $\varphi \in \{\sin, \cos, \exp, \log, \mathrm{sqrt}, \mathrm{abs}\}$,

$$\begin{gathered} X \circ Y \in \Box\{\tilde{x} \circ \tilde{y} \mid \tilde{x} \in X, \tilde{y} \in Y\} \\ \varphi(X) = \Box\{\varphi(\tilde{x}) \mid \tilde{x} \in X\}, \end{gathered}$$

where $\Box S = [\inf(S), \sup(S)]$ is the interval hull of the set S.

In each case the result of $X \circ Y$ or $\varphi(X)$ is the smallest interval containing all results as $\tilde{x}$ and $\tilde{y}$ range independently over X and Y. Possible correlations between X and Y are not considered, and this leads to a difficulty with interval arithmetic. If, for example, $X = [2, 4]$, applying the definition of subtraction, we obtain $X - X = [2, 4] - [2, 4] = [-2, 2]$. Overestimation of the true range of a result can occur whenever a particular variable occurs multiple times in an expression.[1]

[1] [6, pp. 38–42]

3.2 Affine Arithmetic

As discussed in [2] affine arithmetic is another range-based model for self-validated computation. Unlike interval arithmetic, however, affine arithmetic encodes correlations between partially unknown quantities.

A partially unknown quantity x is represented by an affine form

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n.$$

The central value, x_0, is a constant. The ε_i are referred to as noise symbols, and each coefficient x_i is the corresponding i^{th} partial deviation. The values of the ε_i are unknown but are assumed to lie within the interval $[-1, 1]$.

Each ε_i stands for an independent component of the total uncertainty of x and x_i gives the magnitude of that component. The Fundamental Invariant of Affine Arithmetic states that at any point in an AA computation there is a single assignment of values from $[-1, 1]$ to each of the noise symbols, so that each affine form takes on the value of the corresponding quantity in the ideal computation.

Conversion from the interval form $X = [x_1, x_2]$ to the affine form

$$\hat{x} = \mathrm{mid}(X) + \mathrm{rad}(X)\varepsilon_1$$

carries all information about the uncertainty in x encoded in the interval representation.

However, when the affine form

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$$

is converted to the interval form

$$X = [x_0 - \mathrm{rad}(\hat{x}), x_0 + \mathrm{rad}(\hat{x})],$$

with

$$\mathrm{rad}(\hat{x}) = \sum_{i=1}^{n} |x_i|,$$

the information implicit in the dependence of the affine form on the particular noise symbols $\varepsilon_1, \ldots, \varepsilon_n$ is lost.[2]

3.3 Computing in Affine Arithmetic

As discussed in [2] we imagine that there are some precise ideal values x, y which are obtained from their affine forms

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, \quad \hat{y} = y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n,$$

by some choice of the $\varepsilon_i \in [-1, 1]$.

We consider a certain operation f carried out on x and y. We carry out some corresponding operation on $\hat{x}$ and $\hat{y}$, $\hat{f}(\hat{x}, \hat{y}) = \hat{z}$ in order to obtain an affine form $\hat{z}$ of the result z of the ideal computation, where

$$z = \hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_p\varepsilon_p$$

for some choice of the ε_j. If f is itself an affine function, $\hat{f}$ will be an affine combination of the noise symbols.

[2] [2, pp. 43–50].

Specifically, with $\alpha, \zeta \in \mathbb{R}$,

$$\begin{aligned}\hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n,\\ \alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n,\\ \hat{x} \pm \zeta &= (x_0 \pm \zeta) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n.\end{aligned}$$

We note that the range overestimation in the interval operation $X-X$ does not occur in affine arithmetic, as $\hat{x} - \hat{x} = 0$ by the formula above. In practice, to account for any roundoff error, a new noise term $\delta\varepsilon_k$ is added in each of the formulas given.

When the operation $z \leftarrow f(x, y)$ is nonaffine we have

$$\begin{aligned}z &= f(x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n)\\ &= f^*(\varepsilon_1, \ldots, \varepsilon_n)\end{aligned}$$

where f^* is a nonaffine function of the ε_i. In this case an affine function f^a of the ε_i is chosen which approximates f^*.

$$f^a(\varepsilon_1, \cdots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n$$

To f^a an additional error term is added to account for the error in the approximation plus any roundoff error. This yields

$$\begin{aligned}\hat{z} &= f^a(\varepsilon_1, \cdots, \varepsilon_n) + z_k\varepsilon_k\\ &= z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k.\end{aligned}$$

The approximation f^a is often made so as to minimize the maximum absolute error (Chebyshev approximation)[3].

4 LIEs from Affine Forms

The following is a paraphrasing from [1].
Begin with a surface patch $\mathbf{S}(u, v)$ having parameter domain $I_u \times I_v$. Let $\hat{u}, \hat{v}$ be the affine forms of the parameter intervals I_u and I_v.

They are

$$\hat{u} = m_u + r_u\varepsilon_u, \quad \hat{v} = m_v + r_v\varepsilon_v,$$

where

$$m_u = \mathrm{mid}(I_u), \quad r_u = \mathrm{rad}(I_u), \quad m_v = \mathrm{mid}(I_v), \quad r_v = \mathrm{rad}(I_v), \quad \varepsilon_u, \varepsilon_v \in [-1, 1].$$

Now write an affine representation of the surface,

$$\mathbf{S}(u, v) = \begin{pmatrix} x(u, v)\\ y(u, v)\\ z(u, v)\end{pmatrix}.$$

Since, in general, $x(u, v)$ is a non–affine function of u and v, its affine form will be

$$\hat{x}(\hat{u}, \hat{v}) = x^a(\varepsilon_u, \varepsilon_v) + \sum_{i=1}^{n} x_i\varepsilon_i,$$

[3] [2, pp. 49–57].

where x^a is an affine function of the noise symbols ε_u and ε_v, and the

$$x_i\varepsilon_i(i = 1, \cdots, n)$$

are additional noise terms incorporating the error in the approximation x^a plus any roundoff error.

$$\Rightarrow \hat{x}(\hat{u}, \hat{v}) = x_0 + x_u\varepsilon_u + x_v\varepsilon_v + \sum_{i=1}^{n} x_i\varepsilon_i$$

With similar expressions for $\hat{y}$ and $\hat{z}$, we have

$$\begin{aligned}\hat{\mathbf{S}}(\hat{u}, \hat{v}) = \begin{pmatrix} \hat{x}(\hat{u}, \hat{v}) \\ \hat{y}(\hat{u}, \hat{v}) \\ \hat{z}(\hat{u}, \hat{v}) \end{pmatrix} &= \begin{pmatrix} x_o + x_u\varepsilon_u + x_v\varepsilon_v + \sum_{i=1}^{n} x_i\varepsilon_i \\ y_o + y_u\varepsilon_u + y_v\varepsilon_v + \sum_{i=1}^{n} y_i\varepsilon_i \\ z_o + z_u\varepsilon_u + z_v\varepsilon_v + \sum_{i=1}^{n} z_i\varepsilon_i \end{pmatrix} \\ &= \mathbf{f}^o + \mathbf{f}^u\varepsilon_u + \mathbf{f}^v\varepsilon_v + \sum_{i=1}^{n} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \varepsilon_i,\end{aligned}$$

with

$$\mathbf{f}^0 = \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix}, \quad \mathbf{f}^u = \begin{pmatrix} x_u \\ y_u \\ z_u \end{pmatrix}, \quad \mathbf{f}^v = \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix}.$$

By the fundamental invariant of affine arithmetic, for any given point $(u, v) \in I_u \times I_v$, there are elements of $[-1, 1]$, ε_u, ε_v, ε_i $(i = 1, \cdots, n)$ such that

$$u = m_u + r_u\varepsilon_u, \quad v = m_v + r_v\varepsilon_v,$$

and

$$\mathbf{S}(u, v) = \mathbf{f}^0 + \mathbf{f}^u\varepsilon_u + \mathbf{f}^v\varepsilon_v + \sum_{i=1}^{n} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \varepsilon_i.$$

Making the substitutions

$$\varepsilon_u = \frac{u}{r_u} - \frac{m_u}{r_u}, \quad \varepsilon_v = \frac{v}{r_v} - \frac{m_v}{r_v},$$

we have

$$\mathbf{S}(u, v) = \mathbf{f}^0 - \frac{m_u}{r_u}\mathbf{f}^u - \frac{m_v}{r_v}\mathbf{f}^v + \sum_{i=1}^{n} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \varepsilon_i + \frac{1}{r_u}\mathbf{f}^u u + \frac{1}{r_v}\mathbf{f}^v v$$

for some choice of the $\varepsilon_i \in [-1, 1]$.

Thus, for any $(u, v) \in I_u \times I_v$,

$$\mathbf{S}(u, v) \in \mathbf{f}_o + \mathbf{Q} - \frac{m_u}{r_u}\mathbf{f}^u - \frac{m_v}{r_v}\mathbf{f}^v + \frac{1}{r_u}\mathbf{f}^u u + \frac{1}{r_v}\mathbf{f}^v v$$

where

$$\mathbf{Q} = \begin{pmatrix} -\sum\limits_{i=1}^{n} |x_i|, \ \sum\limits_{i=1}^{n} |x_i| \\ -\sum\limits_{i=1}^{n} |y_i|, \ \sum\limits_{i=1}^{n} |y_i| \\ -\sum\limits_{i=1}^{n} |z_i|, \ \sum\limits_{i=1}^{n} |z_i| \end{pmatrix}.$$

Then, defining interval vector

$$\mathbf{P}_{(u,v)} = \mathbf{f}^o + \mathbf{Q} - \frac{m_u}{r_u}\mathbf{f}^u - \frac{m_v}{r_v}\mathbf{f}^v,$$

for any $(u, v) \in I_u \times I_v$ we have

$$\mathbf{S}(u, v) \in \ \mathbf{P}_{(u,v)} + \frac{1}{r_u}\mathbf{f}^u u + \frac{1}{r_v}\mathbf{f}^v v \ = \ L(u, v).$$

Bühler [1] calls $L(u, v)$ a Linear Interval Estimation (LIE) for the surface patch $\mathbf{S}(u, v)$.

As (u, v) ranges through $I_u \times I_v$, $L(u, v)$ describes an interval plane segment which is a bounding volume for $\mathbf{S}(u, v)$. Thus, the LIE is a parametized bounding volume for the surface with the same parametrization as the surface.

A note on the above calculations: In order to ensure the inclusion of $\mathbf{S}(u, v)$ in $L(u, v)$, $\mathbf{P}_{(u,v)}$ must be computed using interval arithmetic. Specifically, $\sum_{i=1}^{n} |x_i|$, $\sum_{i=1}^{n} |y_i|$, and $\sum_{i=1}^{n} |z_i|$ must be computed rounding up. Also, starting with $\mathbf{f}^o$, $\mathbf{f}^u$ and $\mathbf{f}^v$, $\mathbf{f}^o + \mathbf{Q} - \frac{m_u}{r_u}\mathbf{f}^u - \frac{m_v}{r_v}\mathbf{f}^v$ must be done as an interval computation. In addition, in $\mathbf{P}_{(u,v)} + \frac{1}{r_u}\mathbf{f}^u u + \frac{1}{r_v}\mathbf{f}^v v$, the division of $\mathbf{f}^u$ by r_u and of $\mathbf{f}^v$ by r_v must be done using interval arithmetic, and henceforth $\frac{1}{r_u}\mathbf{f}^u$ and $\frac{1}{r_v}\mathbf{f}^v$ must be treated as interval vectors, $\mathbf{Y_1}$ and $\mathbf{Y_2}$.

5 Ray-LIE Intersection

Now consider a surface patch $\mathbf{S}(u, v), (u, v) \in I_u \times I_v$ and its corresponding LIE,

$$L(u, v) = \mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v, \quad \mathbf{Y_1} \leftarrow \frac{1}{r_u}\mathbf{f}^u, \quad \mathbf{Y_2} \leftarrow \frac{1}{r_v}\mathbf{f}^v.$$

Let $\mathbf{R}(s) = E + s\mathbf{d}$ be a ray having point of origin E, direction $\mathbf{d}$, and parameter s.

The ray intersects the LIE when

$$\begin{aligned} L(u, v) = \mathbf{R}(s) &\Leftrightarrow \mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v = E + s\mathbf{d} \\ &\Leftrightarrow \mathbf{Y_1}u + \mathbf{Y_2}v - \mathbf{d}s = E - \mathbf{P}_{(u,v)}. \end{aligned}$$

Let $I_u^* \times I_v^* \times I_s^*$ be the solution of this system obtained using interval arithmetic. Then $I_u^* \times I_v^* \times I_s^*$ bounds the solution in parameter space to the ray-LIE intersection. We also note that $I_u^* \times I_v^*$ is not constrained to be contained in $I_u \times I_v$.

Let $I_u \times I_v$ be the parameter domain of the surface patch $\mathbf{S}(u, v)$.

For any $(u, v) \in I_u \times I_v$, we have

$$\mathbf{S}(u, v) \in \mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v = L(u, v).$$

Suppose the ray $\mathbf{R}(s)$ intersects the surface $\mathbf{S}$ at

$$\begin{aligned}(u^*,v^*)\in I_u\times I_v &\Rightarrow \mathbf{S}(u^*,v^*)=\mathbf{R}(s^*)\\ &\Rightarrow \mathbf{R}(s^*)\in \mathbf{P}_{(u,v)}+\mathbf{Y_1}u^*+\mathbf{Y_2}v^*\\ &\Rightarrow \exists \mathbf{p}\in\mathbf{P}_{(u,v)},\exists\mathbf{y_1}\in\mathbf{Y_1},\exists\mathbf{y_2}\in\mathbf{Y_2}\quad\text{such that}\quad \mathbf{R}(s^*)=\mathbf{p}+\mathbf{y_1}u^*+\mathbf{y_2}v^*\\ &\Rightarrow (u^*,v^*,s^*)\end{aligned}$$

is in the solution set of

$$\mathbf{P}_{(u,v)}+\mathbf{Y_1}u+\mathbf{Y_2}v=\mathbf{R}(s),$$

that is, $L(u,v)=\mathbf{R}(s)$.

So any parameter solution to the ray-surface intersection is contained within the set of parameter solutions to the ray-LIE intersection. So we have

Proposition 1 *Let $I_u^{'}\times I_v^{'}$ be the solution set to $L(u,v)=\mathbf{R}(s)$ and let $I_u\times I_v$ be the domain of the patch. Then, for any parameter solution (u^*,v^*) to the ray-surface intersection, we have $(u^*,v^*)\in(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)$.*

This implies the following practical consequences.

(I) $(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)$, which we refer to as the pruned domain, contains all solutions to the ray-surface intersection.

(II) If $(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)=\emptyset$, the ray does not intersect the surface.

Now suppose $(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)\neq\emptyset$.

Then $\exists(u^*,v^*)\in I_u\times I_v$, such that $\mathbf{R}(s^*)=\mathbf{p}^{'}+\mathbf{y}_1^{'}u^*+\mathbf{y}_2^{'}v^*$, where $\mathbf{p}^{'}\in\mathbf{P}_{(u,v)}$, $\mathbf{y}_1^{'}\in\mathbf{Y}_1$, and $\mathbf{y}_2^{'}\in\mathbf{Y}_2$. But $\mathbf{p}^{'}+\mathbf{y}_1^{'}u^*+\mathbf{y}_2^{'}v^*$, where $(u^*,v^*)\in(I_u\times I_v)$, is a point within the LIE restricted to $I_u\times I_v$. Therefore, the ray intersects a LIE which contains the surface $\mathbf{S}(u,v),(u,v)\in I_u\times I_v$. This implies

(III) If $(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)\neq\emptyset$, the ray intersects a LIE which contains the surface. In this case the ray may or may not intersect the surface, but it does pass within a distance of the patch on the order of the dimensions of the LIE restricted to $(u,v)\in I_u\times I_v$.

Observations I, II, and III provide the basis for an algorithm to determine ray-surface intersection. The basic process will be an iteration of the ray-LIE intersection and domain pruning. To this is added a subdivision process to ensure reduction of the parameter space being searched.

6 Search of the Parameter Space $(I_u\times I_v)$ for a Solution to Ray–Surface Intersection

Begin with a surface patch $\mathbf{S}$(u,v), $(u,v)\in(I_u\times I_v)$ and its corresponding LIE, $L(u,v)$, $(u,v)\in(I_u\times I_v)$. Compute the solution $(I_u^{'}\times I_v^{'})$ to the ray-LIE intersection $\mathbf{R}(s)=L(u,v)$. Call $(I_u^{'}\times I_v^{'})\cap(I_u\times I_v)=(I_u^1\times I_v^1)$ the pruned domain. Since $(I_u^1\times I_v^1)$ contains all solutions to the ray-surface intersection, it is the only part of $(I_u\times I_v)$ which is relevant to the search. If $(I_u^1\times I_v^1)=\emptyset$, there is no solution. If $(I_u^1\times I_v^1)\neq\emptyset$. the LIE containing the surface is intersected by the ray. In this case the ray passes within a distance of the surface on the order of the dimensions of the LIE. Therefore, if the LIE is small enough we record a solution at the LIE's center. If $(I_u^1\times I_v^1)\neq\emptyset$, but the LIE is not small, we iterate ray-LIE intersection and domain pruning.

There are several possible outcomes to the iteration.

1. **No solution case**.
   As has been shown, any solution to the ray-surface intersection is within the corresponding pruned domain. Thus, if the process of ray-LIE intersection and domain pruning results in an empty pruned domain $(I_{u,i+1} \times I_{v,i+1})$,
   $$\begin{aligned} &(I_{u,i} \times I_{v,i}) \leftarrow \text{domain} \\ &(I'_{u,i} \times I'_{v,i}) \leftarrow \text{solution to } L(u,v) = \mathbf{R}(s) \\ &(I_{u,i} \times I_{v,i}) \cap (I'_{u,i} \times I'_{v,i}) = (I_{u,i+1} \times I_{v,i+1}) = \emptyset \end{aligned}$$
   this verifies that there is no solution to the ray-surface intersection within $(I_{u,i} \times I_{v,i})$.
2. **An in tolerance solution**.
   When the iteration
   $$\begin{aligned} &(I_{u,i} \times I_{v,i}) \leftarrow \text{domain} \\ &(I'_{u,i} \times I'_{v,i}) \leftarrow \text{solution to } L(u,v) = \mathbf{R}(s) \\ &(I_{u,i} \times I_{v,i}) \cap (I'_{u,i} \times I'_{v,i}) = (I_{u,i+1} \times I_{v,i+1}) \neq \emptyset \end{aligned}$$
   results in a nonempty pruned domain, a LIE containing part of the surface is intersected by the ray. Therefore the ray passes within a distance of the surface on the order of the dimensions of the LIE. So if the LIE is small enough we have a solution. It can be shown that the width of a LIE decreases at least linearly with the width of its parameter domain. A tolerance value on the width of the parameter domain can thus be chosen which produces a LIE of near or below pixel dimensions. If such a LIE contains a portion of the surface and is intersected by the ray (i.e., the LIE has a nonempty pruned domain) the ray passes within pixel dimensions of the surface. In this case the pixel is given surface color. We refer to this situation as an in tolerance solution.
3. **A stalled iteration**.
   It may happen in the iteration that the pruned domain is not a proper subset of the starting domain.
   $$(I_{u,i} \times I_{v,i}) \cap (I'_{u,i} \times I'_{v,i}) = (I_{u,i} \times I_{v,i})$$
   In this case the iteration would stall without the subdivision process.
4. **The pruned domain is nonempty but the LIE is not within solution tolerance**.
   $$\begin{aligned} &(I_{u,i} \times I_{v,i}) \leftarrow \text{domain} \\ &(I_{u,i} \times I_{v,i}) \cap (I'_{u,i} \times I'_{v,i}) = (I_{u,i+1} \times I_{v,i+1}) \neq \emptyset \\ &(I_{u,i+1} \times I_{v,i+1}) \leftarrow \text{new starting domain} \end{aligned}$$
   In this case we wish to continue the iteration on $(I_{u,i+1} \times I_{v,i+1})$.
   In view of (3) we subdivide $(I_{u,i+1} \times I_{v,i+1})$ along the parameter interval of greatest width before continuing with the iteration. This guarantees that the size of the parameter domains being searched decreases.

The resulting algorithm is outlined in the following psuedocode. A linked list, $\mathfrak{L}$, of LIE objects is created and the search proceeds through $\mathfrak{L}$. Each LIE object has associated with it the LIE's parameter domain $(I_{u,i} \times I_{v,i})$ and the corresponding pruned domain $(I_{u,i} \times I_{v,i}) \cap (I'_{u,i} \times I'_{v,i})$. Only LIEs with nonempty pruned domains are put on the list, and the list is maintained in order according to the minimum time (ray parameter) at which each LIE is hit by the ray. LIEs with smaller hit times are put on the list ahead of LIEs with larger hit times.

The Algorithm

```
Input: A surface $\mathbf{S}$(u,v) and a ray $\mathbf{R}$(s).
HaveSolution = False
    Compute LIE of full surface.
    Compute solution to ray–LIE intersection and
    the corresponding pruned domain.
    if(pruned domain $\neq \emptyset$)
        {Initialize list $\mathfrak{L}$ to contain the full surface LIE.
        While {$\mathfrak{L} \neq \emptyset$}
            {min s = time at which the ray first hits the LIE at top of $\mathfrak{L}$.
            if(HaveSolution = True and solution hit time < min s)[4]
            {break from while loop}
            if(LIE at top of $\mathfrak{L}$ is small enough)
                { Set HaveSolution = True
                if(this solution is the 1st solution or the solution closest to the ray origin)
                {record the solution}
                Delete the LIE at the top of $\mathfrak{L}$.
                Continue at the beginning of the while loop.
                }
            else if (the LIE at the top of $\mathfrak{L}$ is not small)
                {Divide the LIE's pruned domain.
                Compute LIEs for the two pruned domain halves.
                Intersect these LIEs with the ray and
                compute the corresponding pruned domains.
                Delete the LIE at the top of $\mathfrak{L}$.
                Insert each LIE with nonempty pruned domain
                into $\mathfrak{L}$ in order of their hit times.
                Continue at beginning of while loop.
                } end else if
            } end while loop
        } end if

    if(HaveSolution = True)
        {return recorded solution}
    if (HaveSolution = False)
        {return "no solution"}
```

The algorithm produces a binary tree structure whose root is the LIE of the original surface. Non-leaf nodes correspond to LIEs whose pruned domains have been divided. Leaf nodes correspond to either LIEs with empty pruned domains or LIEs corresponding to in tolerance solutions.

[4] Explanation of the first if clause in the while loop: When a solution is found, its hit time is the corresponding value of the ray parameter. A LIE's hit time is the smallest value of the ray parameter corresponding to ray-LIE intersection. That is, the LIE's hit time is the time the LIE is first hit by the ray. Since the LIEs are on the list in order of their hit times, if the solution hit time is less than the hit time of the LIE at the top of the list, it is less than the hit times of all the LIEs on the list. Therefore, since the LIEs on the list cover that part of the surface still to be searched, the solution is necessarily closer to the ray's origin than any other solution. In this case the search can be ended. Hence the while loop is exited when the stated condition is met.

The subdivision process guarantees that the tree ends in leaves after a finite number of levels because tolerance dimensions are eventually reached, and then the pruned domain is either

1. empty $\Rightarrow$ empty pruned domain leaf, or
2. nonempty $\Rightarrow$ in tolerance solution leaf.

Any solution (u^*, v^*) in a subdomain is in the corresponding pruned domain. Therefore any solution in the original domain must be in some leaf node at the end of the search. Thus if all leaves correspond to LIEs with empty pruned domains, there can be no solution and this result is returned by the algorithm. Furthermore:

– If an in tolerance solution leaf occurs, that solution is returned by the algorithm.
– If more than one solution leaf occurs, the solution which is closest to the ray origin is returned as the solution.

7 Domain Inflation

As the algorithm proceeds it may happen that a pruned domain is encountered which has zero width in one of the parameter intervals, e.g., $I_u \times I_v = [0.179411, 0.187177] \times [1, 1]$ occurs as a pruned domain in one of our test surfaces. In the next iteration the computation of the LIE for this parameter domain involves division by $r_v = rad(I_v)$. This causes the program to terminate in a division by zero.

This difficulty can be overcome by testing the diameter of the pruned domain and inflating the pruned domain slightly if the diameter is below a certain tolerance. We have found that the following code works well.

if (Diam$(I_u) < 0.0000001$)

$I_u = \text{Hull}(\text{Inf}(I_u) - 0.000001, \text{Sup}(I_u) + 0.000001)$ (same for I_v).

8 Normal Vectors

Once an in tolerance solution is found it is necessary to find a normal vector to the surface at the location of the solution. A lighting model can then be invoked to assign an appropriate color to the pixel.

For $\mathbf{S}(u, v)$ with $(u, v) \in I_u \times I_v = [a, b] \times [c, d]$ we have

$$\mathbf{S}(u, v) \in L(u, v) = \mathbf{P}_{(u,v)} + \frac{\mathbf{f}^u}{r_u} u + \frac{\mathbf{f}^v}{r_v} v.$$

When the LIE L(u,v) is computed using affine arithmetic it can be shown by direct calculation that in the limit as $b \to a$ and $d \to c$, $\frac{\mathbf{f}^u}{r_u} \to \frac{\partial \mathbf{S}(a,c)}{\partial u}$ and $\frac{\mathbf{f}^v}{r_v} \to \frac{\partial \mathbf{S}(a,c)}{\partial v}$.

Therefore, as the widths of the domains of the LIEs shrink toward zero, the coefficient vectors $\frac{\mathbf{f}^u}{r_u}$ and $\frac{\mathbf{f}^v}{r_v}$ become good approximations to the tangent vectors $\frac{\partial \mathbf{S}(a,c)}{\partial u}$ and $\frac{\partial \mathbf{S}(a,c)}{\partial v}$. So once an in tolerance solution is found, the vectors $\frac{\mathbf{f}^u}{r_u}$ and $\frac{\mathbf{f}^v}{r_v}$, which have already been calculated, can be used in place of $\frac{\partial \mathbf{S}}{\partial u}$ and $\frac{\partial \mathbf{S}}{\partial v}$ to compute a normal vector $\frac{\mathbf{f}^u}{r_u} \times \frac{\mathbf{f}^v}{r_v}$ to the surface. We have found that this method of computing a normal vector gives good results.

9 Computation Details

We now present some of the detail in the computations.

9.1 Solution of the ray-LIE Intersection

With $L(u,v) = \mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v$ and $R(s) = E + s\mathbf{d}$, we have

$$\begin{aligned}\mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v = E + s\mathbf{d} \Leftrightarrow u\mathbf{Y_1} + v\mathbf{Y_2} - s\mathbf{d} &= E - \mathbf{P}_{(u,v)} \\ \Leftrightarrow uY_{1x} + vY_{2x} - sd_x &= E_x - P_x \\ uY_{1y} + vY_{2y} - sd_y &= E_y - P_y \\ uY_{1z} + vY_{2z} - sd_z &= E_z - P_z,\end{aligned}$$

where the x, y, and z subscripts denote the x, y, and z vector components.

The parameter s can be eliminated to obtain

$$\begin{pmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix},$$

where

$$\begin{aligned} G_{1,1} &= d_xY_{1y} - d_yY_{1x}, & G_{1,2} &= d_xY_{2y} - d_yY_{2x}, \\ G_{2,1} &= d_xY_{1z} - d_zY_{1x}, & G_{2,2} &= d_xY_{2z} - d_zY_{2x}, \\ B_1 &= d_x(E_y - P_y) - d_y(E_x - P_x) & B_2 &= d_x(E_z - P_z) - d_z(E_x - P_x), \end{aligned}$$

And by Cramer's rule we obtain

$$U = \frac{\begin{vmatrix} B_1 & G_{1,2} \\ B_2 & G_{2,2} \end{vmatrix}}{\begin{vmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{vmatrix}}, \quad V = \frac{\begin{vmatrix} G_{1,1} & B_1 \\ G_{2,1} & B_2 \end{vmatrix}}{\begin{vmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{vmatrix}}.$$

When using interval arithmetic to do this calculation, the solution $U \times V$ will bound the solution to the ray-LIE intersection. So:

Proposition 2 *The computed ray-LIE intersection solution is reliable in the sense that it is guaranteed to contain all solutions to the ray-LIE intersection.*

We note that E is a constant for the ray-tracing process and so may be assumed to be exact. The components of $\mathbf{d}$ are calculated for a given ray and so are subject to rounding error. However, $\mathbf{d}$ is not calculated by an iterative process, so that roundoff error does not build up. Any error should be minimal and not likely to effect the final results. We therefore feel justified in ignoring it and in assuming that the components of $\mathbf{d}$ are exact.

9.2 Computation of the LIE

In order to carry out the ray-LIE intersections involving the LIE, $L(u,v) = \mathbf{P}_{(u,v)} + \mathbf{Y_1}u + \mathbf{Y_2}v$, we require $\mathbf{P}_{(u,v)}, \mathbf{Y_1}$, and $\mathbf{Y_2}$. These forms need to be calculated for the general LIE with parameter domain $[a,b] \times [c,d]$.

We have

$$\mathbf{P}_{(u,v)} = \mathbf{f}^0 + \mathbf{Q} - \frac{m_u}{r_u}\mathbf{f}^u - \frac{m_v}{r_v}\mathbf{f}^v, \quad \mathbf{Y_1} = \frac{1}{r_u}\mathbf{f}^u, \quad \mathbf{Y_2} = \frac{1}{r_v}\mathbf{f}^v,$$

where

$$\mathbf{f}^0 = \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix}, \mathbf{Q} = \begin{pmatrix} -\sum\limits_{i=1}^{n} |x_i|, & \sum\limits_{i=1}^{n} |x_i| \\ -\sum\limits_{i=1}^{n} |y_i|, & \sum\limits_{i=1}^{n} |y_i| \\ -\sum\limits_{i=1}^{n} |z_i|, & \sum\limits_{i=1}^{n} |z_i| \end{pmatrix}, \mathbf{f}^u = \begin{pmatrix} x_u \\ y_u \\ z_u \end{pmatrix}, \quad \mathbf{f}^v = \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix},$$

$$m_u = \text{mid}[a,b], \quad m_v = \text{mid}[c,d], \quad r_u = \text{rad}[a,b], \quad r_v = \text{rad}[c,d],$$

and m_u, m_v, r_u, r_v, are calculated from the domain. In order to calculate the other forms, needed are the central value x_0 and the partial deviations $x_u, x_v, x_1, \dots, x_n$ of the affine representation of $x(u,v)$, $\hat{x}(\hat{u},\hat{v}) = x_0 + x_u\varepsilon_u + x_v\varepsilon_v + \sum_{i=1}^{n} x_i\varepsilon_i$. It is similar for $\hat{y}(\hat{u},\hat{v})$ and $\hat{z}(\hat{u},\hat{v})$.
Letting $x_1(u,v) = x(u,v)$, $x_2(u,v) = y(u,v)$, $x_3(u,v) = z(u,v)$, we have for a Bezier surface patch,

$$x_i(u,v) = [u^3\, u^2\, u\, 1] M P_i M \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix}, \quad \text{with} \quad M = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$

and P_i is the 4×4 matrix of the i^{th} components of the control points. Writing $A_i = MP_iM$,

$$\begin{aligned} x_i(u,v) = {} & (u^3A_{i11} + u^2A_{i21} + uA_{i31} + A_{i41})v^3 \\ & + (u^3A_{i12} + u^2A_{i22} + uA_{i32} + A_{i42})v^2 \\ & + (u^3A_{i13} + u^2A_{i23} + uA_{i33} + A_{i43})v \\ & + (u^3A_{i14} + u^2A_{i24} + uA_{i34} + A_{i44}). \end{aligned}$$

With the LIE domain $[a,b] \times [c,d] = I_u \times I_v$ we must build the affine form of $x_i(u,v)$. Code for doing this is described in de Figueiredo and Stolfi [2]. In our implementation we wrote the code based on the discussion in [2]. An affine arithmetic package is also available from [7].

10 Theorem on Reliability

Theorem 1 (Reliability) *If the ray through a pixel hits the surface, the pixel is colored.*

Proof Let $I_u \times I_v$ be the parameter domain of the surface. Assume the ray hits the surface at (u^*, v^*). By Proposition 1, $(u^*, v^*) \in$ pruned domain[5]. Then,

1. $(u^*, v^*) \in$ pruned domain $\Rightarrow$ pruned domain $\neq \emptyset$.
2. If the pruned domain is small enough, we color the pixel if this is the first solution or the solution closest to the ray origin. If it is not the solution closest to the origin, the pixel is colored due to the closer solution. In any case, if the pruned domain is small enough, the pixel is colored.
3. If the pruned domain is not small enough, we divide the pruned domain (The division is done reliably using PROFIL.) We then know that:
   (a) (u^*, v^*) is in one of the divisions;

[5] By "pruned domain," we mean $I_u' \times I_v' \cap I_u \times I_v$, where $I_u' \times I_v'$ is the solution to the ray-LIE intersection. By Proposition 2, $I_u' \times I_v'$ is computed reliably, so we are guaranteed that $(u^*, v^*) \in I_u' \times I_v'$. The intersection $I_u' \times I_v' \cap I_u \times I_v$ is also computed reliably by the PROFIL interval software, so we are guaranteed $(u^*, v^*) \in I_u' \times I_v' \cap I_u \times I_v$.

(b) the pruned domain for that division is not empty;
(c) the corresponding LIE will be put on the list for further processing[6].

11 Numerical Results and Comparisons

11.1 Toth's algorithm

For purposes of comparison we also implemented Toth's method [8], an early ray-tracing algorithm. Toth begins with a parametric surface $\mathbf{H}(s,t) = (x(s,t), y(s,t), z(s,t))$ and a parametric ray $\mathbf{R}(u) = E + u\mathbf{d}$. Then the ray-surface intersection is the solution to the nonlinear system $\mathbf{H}(s,t) - \mathbf{R}(u) = \mathbf{0}$, which can be written $\mathbf{f}(v) = \mathbf{0}$, $\mathbf{v} = (s,t,u)$. Toth uses the Krawczyk method to bound solutions; Toth also uses traditional modified Newton iteration heuristically in certain places in his algorithm. Traditional Newton method iteration has the form $\mathbf{v}_{k+1} = \mathbf{v}_k - Yf(\mathbf{v}_k)$, where Y is the inverse Jacobian of $\mathbf{f}$ at $\mathbf{v}_k$. Letting interval vector $\mathbf{X}$ be a subset of the domain of $\mathbf{f}$, with $\mathbf{y}$ any real vector in $\mathbf{X}$, and $F'(\mathbf{X})$ an interval extension of the Jacobian of $\mathbf{f}$, Toth considers the Krawczyk operator

$$K(\mathbf{X}, \mathbf{y}, Y) = \mathbf{y} - Y\mathbf{f}(\mathbf{y}) + \{I - YF'(\mathbf{X})\}(\mathbf{X} - \mathbf{y})$$

K has the following properties:

- $K \subset \mathbf{X}$ implies that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ has a solution in $\mathbf{X}$.
- All solutions to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ are in K.
- $K \cap \mathbf{X} = \emptyset$ implies there is no solution to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in X.
- If $K \subset \mathbf{X}$ and $r = ||I - YF'(\mathbf{X})|| < 1$, then the simple Newton sequence $\mathbf{x}_{n+1} = \mathbf{x}_n - Y\mathbf{f}(\mathbf{x}_n)$ (Y held fixed) will converge to a unique solution $\mathbf{x}^*$ in $\mathbf{X}$ to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ from any starting point $\mathbf{x}_0$ in $\mathbf{X}$. (In this case $\mathbf{X}$ is called a safe-starting region.)

The interval Newton iterartion is

$$\mathbf{X}_{n+1} = K(\mathbf{X}_n, \mathrm{mid}(\mathbf{X}_n), Y_n) \cap \mathbf{X}_n, \tag{1}$$

where

$$Y_n = \begin{cases} \mathrm{mid}(F'(\mathbf{X}_n))^{-1} & \text{when } r_n \leq r_{n-1}, \\ Y_{n-1} & \text{otherwise,} \end{cases}$$

with $r_n = ||I - Y_n F'(\mathbf{X}_n)||$.

Following developments by Moore[4], [5], Toth[8] proves the following theorem on (1).

Theorem 2 *Suppose that $||I - YF'(\mathbf{X})|| = r < 1$. Set $Y_0 = Y$ and $\mathbf{X}_0 = \mathbf{X}$ in (1). Then either the iterative sequence (1) will converge to a unique solution $\mathbf{x}^*$ in $\mathbf{X}$ to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, or (1) will break down due to empty intersection, in which case no solutions to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ are in $\mathbf{X}$.*

Toth uses these properties of K to construct a search algorithm for the solution to the ray-surface intersection.

The initial surface domain is divided into quarters and a list of subregions is initialized to contain these quarters. The list is maintained in order of the ray parameter (time) when surface extents of the subregions are first intersected by the ray. Subregions with smaller extent hit times are put ahead of those with larger extent hit times.

[6] In this case, we return to Step 2, to either record a solution or subdivide further. Since we always divide along the parameter interval of longest width, repeated calls to Steps 2 and 3 eventually lead to a nonempty pruned domain which is small enough.

Starting with the first subregion $\mathbf{X}_0$ on the list, $K(\mathbf{X}_0, \text{mid}(\mathbf{X}_0), Y)$ and $r = ||I - YF'(\mathbf{X}_0)||$ with $Y = \text{mid}(F'(\mathbf{X}_0))^{-1}$ are calculated. $\mathbf{X}_0$ is excluded from the search if $K(\mathbf{X}_0, \text{mid}(\mathbf{X}_0), Y) \cap \mathbf{X}_0 = \emptyset$ or if the ray does not intersect an extent for the surface on $\mathbf{X}_0$. If $\mathbf{X}_0$ is safe-starting, simple Newton iteration is performed to a solution. If $\mathbf{X}_0$ is not safe-starting but cannot be excluded and $r = ||I - YF'(\mathbf{X}_0)|| < 1$, the interval Newton iteration is performed with each iterate being tested for safe-starting or exclusion. If the safe-starting condition is met, simple Newton iteration is run to a solution. If r is not less than one but $\mathbf{X}_0$ cannot be excluded $K(X_0, \text{mid}(\mathbf{X}_0), Y) \cap \mathbf{X}_0$ is subdivided along the parameter direction of longest width. The two halves resulting from the division are put on the list in order of their extent hit times. Once a subregion has been processed the next list element is chosen and the steps are repeated. For $i > 0$, a subregion $\mathbf{X}_i$ is excluded from the search if a solution has already been found with a ray parameter less than the time the surface extent over $\mathbf{X}_i$ is first hit by the ray. When a solution is found it is recorded if it is the first solution or the solution closest to the ray origin of the solutions thus far identified. In this way when multiple solutions are found, it is the one closest to the ray origin that is returned at the end of the search.

Toth notes that incorrect results may occur due to floating point round-off if either the width of the parameter interval or the widths of the interval extensions to the partial derivatives become too small. Therefore, tolerances are placed on these widths and a subregion below them is considered too small for further processing. If a region becomes too small one final Newton step is taken. If the final step is within the subregion, this is considered a solution. If the final step is outside the region, the region is excluded from further search.

Deciding on whether or not there is a solution based on the heuristic described in the last paragraph does not guarantee a correct result. The analog to the two-dimensional case depicted in figure 1 would fail to identify the solution x^*. Such a failure would seem more likely when a solution is near a subregion boundary.

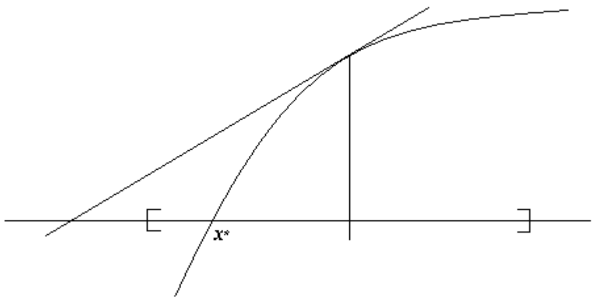


Fig. 1 Failure of the final Newton step heuristic.

11.2 Implementation

Both Toth's method and the LIE-based algorithm were implemented in C++ using Microsoft's Visual C++ environment. The interval library PROFIL (version 1) by Knüppel[3] was also used. The programs were done in single precision. Several versions of Toth's method were implemented. One version, which we will call version A, subdivided the intersection of the Krawczyk operator and the parameter domain $K(X, m(X), Y) \cap X$ in the subdivision step. Another version, which we will call version B, subdivided the parameter domain X in the subdivision step. Version A is consistent with the method described by Toth in his 1985 paper, while version B is a variant of that method.

Another way to design different versions of the Toth algorithm is based on the character of the Y matrix of the Newton iteration. Recall that we are solving the ray-surface intersection $H(s,t) - (E + uD) = 0$ which is the system

$$\begin{pmatrix} x(s,t) - E_x - uD_x \\ y(s,t) - E_y - uD_y \\ z(s,t) - E_z - uD_z \end{pmatrix} = \mathbf{0}$$

where E_x, E_y, E_z are the x, y, and z components of the starting point of the ray and D_x, D_y, D_z are the x, y, and z components of the direction vector of the ray. The parameter u can be eliminated using $u = \frac{x(s,t)-E_x}{D_x}$ to yield the equivalent system

$$\mathbf{g}(s,t) = \begin{pmatrix} D_x(y - E_y) - D_y(x - E_x) \\ D_x(z - E_z) - D_z(x - E_x) \end{pmatrix} = \mathbf{0}.$$

To solve the system $\mathbf{g}(v) = \mathbf{0}, v = (s,t)$ we perform the Newton iteration $v_{k+1} = v_k - Y\mathbf{g}(v_k)$. There are different choices for the matrix Y. Y may be the inverse of the matrix of midpoints of an interval extension of the Jacobian of $\mathbf{g}$ over the parameter domain. We will call this version 1. Y may be the inverse of the Jacobian of $\mathbf{g}$ at the midpoint of the parameter domain $I_u \times I_v$. We will call this version 2. We implemented both version 1 and 2.

11.3 Robustness Comparisons

We tested the five programs (LIE, A1, A2, B1, B2) on six different surfaces (figure 3 through figure 8). On the first five of these all the programs gave identical results, meaning that the renderings were indistinguishable. The sixth surface was self-intersecting and we rendered two views, one oblique view and one more edge-on view. We will refer to the more edge-on view as surface 7. The five programs gave identical renderings of the oblique view (figure 6) but gave different results on surface 7 (figure 7 and figure 8). On surface 7 all five programs failed to color a number of pixels in the lower left side of the surface where the image is very thin. Rays are cast through the lower left corners of the pixels and we speculate that the surface is so thin that rays from a succession of pixels miss it (figure 2). The fact that all the programs show the same effect suggests that the difficulty results from the discrete nature of the tracing process and not from a fault in the program methods.

Aside from this the LIE-based program traced the full surface 7 (figure 7). The Toth version B1 program traced the full surface except for two pixels on the right side. The Toth version B2 program traced the full surface except for one pixel on the right side (figure 8). The Toth version A1 program missed multiple pixels in a band along the right side and some additional pixels toward the middle of the surface (figure 7). Toth version A2 missed fewer pixels than version A1, but still missed multiple pixels in a band along the right side and a pixel near the middle of the surface (figure 8). The Toth version A and version B programs thus differ in a way that significantly effects their performance.

Considering version A, when $y =$ midpoint of $\mathbf{X}$, as is the case in our implementation, $K(\mathbf{X}, \mathbf{y}, Y)$ is symmetric about the Newton step. Therefore, $K(\mathbf{X}, \mathbf{y}, Y)$ tends to be symmetric about a solution as the step nears a solution. Then the bisection of $K(\mathbf{X}, \mathbf{y}, Y) \cap \mathbf{X}$, as in version A, will tend to place the solution near an edge of the resulting subdivisions. This makes it less likely the $K \subset \mathbf{X}$ in the following iteration, so that a safe-starting region is less likely to be found. It also makes it more likely that a final Newton step will be outside the parameter box even when there is a solution within the box.

Consistent with this analysis, in looking at the data for the pixels missed by version A1, we found a substantial number of cases of the following form. Version B1 reached a safe-starting region and returned a solution by the point Newton iteration. For the same ray version A1 did not reach a safe-starting region but did do a final Newton step on a parameter box containing the version B1 solution near one of the box edges. The final Newton step done by version A1 was just outside the parameter box, and so the box was rejected. Since the version B1 program finds solutions from a non-interval Newton iteration, the results are not guaranteed. Therefore, our results do not prove but they do strongly suggest that many of the missed pixels result from failure of the final Newton step to identify solutions.

The version A2 data did not follow the same pattern as version A1. For many of the missed pixels two final Newton steps were taken, both of which failed to find solutions. Unlike the version A1 case, however, the parameter boxes over which these final steps were taken did not contain version B solutions. So we do not have evidence here that the final Newton steps are missing solutions. Also, we observed that in many cases the midpoint values returned by the PROFIL software for small diameter parameter domains were outside the domains. So this is a possible reason why solutions were missed.

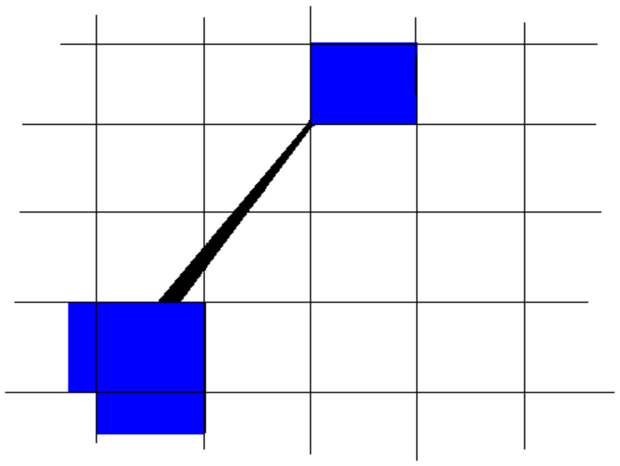


Fig. 2

11.4 Timing Comparisons

Table 1 lists the run times of the five programs on the various surfaces, all normalized to the run time of the Toth version A2 program (the version most consistent with the method described by Toth).

Except for surface 5, where version B is slightly faster and surface 7, where version B is slightly slower, the run times of the different versions of the Toth program are the same. The relative speeds of the Toth and LIE-based programs depend on the surface considered. Relative speeds range from the case of surface 4, where the LIE program takes more than twice the time that the Toth programs take, to the case of surface 7, where the LIE program takes 74 percent of the time that Toth version A takes and 45 percent of the time that Toth version B takes. As seen from the totals, the LIE program takes on average about 155 percent of the time that the Toth programs take. So the relative speeds are roughly comparable.

Table 1 Run times.

	LIE	Toth A1	Toth A2	Toth B1	Toth B2
surface 1	1.49	1.00	1.00	1.00	1.00
surface 2	1.89	1.00	1.00	1.03	1.06
surface 3	1.30	1.00	1.00	0.98	0.98
surface 4	2.38	1.00	1.00	1.00	1.01
surface 5	2.05	0.99	1.00	0.82	0.82
surface 6	0.99	1.01	1.00	1.07	1.08
surface 7	0.74	1.00	1.00	1.65	1.67
Totals	10.84	7.00	7.00	7.54	7.61

12 Conclusions

We have implemented a new method for ray-tracing Bezier surfaces based on Linear Interval Estimations (LIEs). The method is robust. We have proven that if a ray passing through a pixel hits the surface, the pixel will be given surface color. Because of its dependence on a non-rigorous heuristic, a similar theorem does not hold for Toth's method. We have seen an example where this method does not color pixels which should be colored. We have seen evidence to suggest that in some cases the missed pixels are not colored due to the failure of the final Newton step heuristic to identify solutions. A version (B) of the Toth method is closer to the LIE program in robustness, but still fails to match its reliability. The use of LIE's in ray-tracing may thus be advantageous when a high degree of reliability is required.

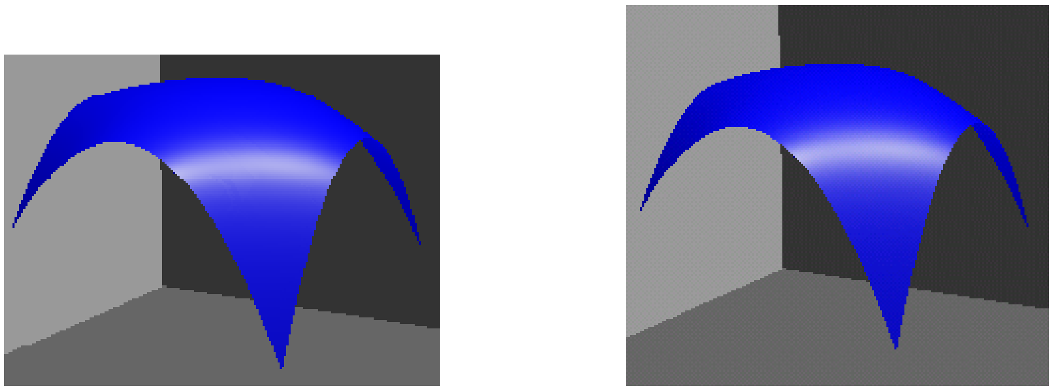


Fig. 3 On the left, Surface 1 by the LIE method. On the right, Surface 1 by Toth version A2.

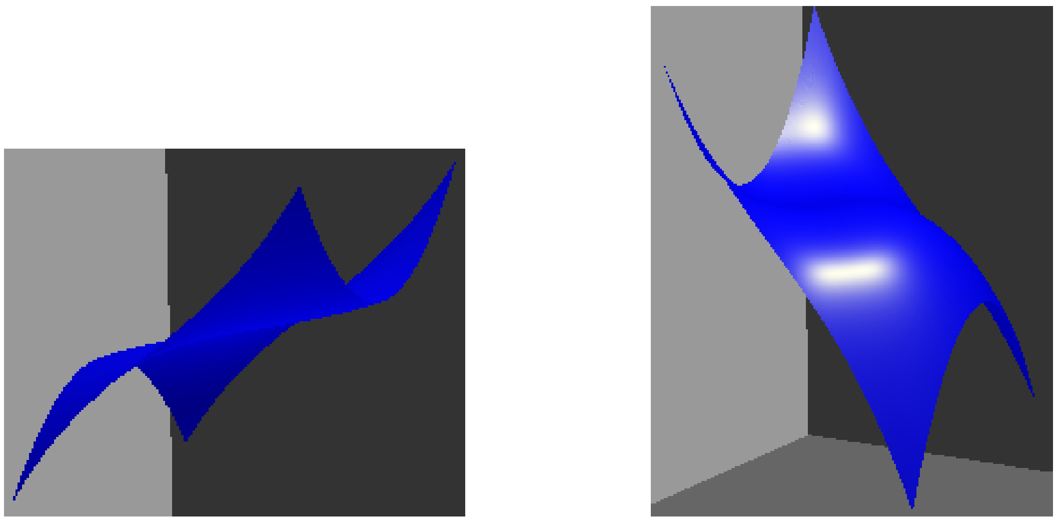


Fig. 4 On the left, Surface 2 by the LIE method. On the right, Surface 3 by the LIE method.

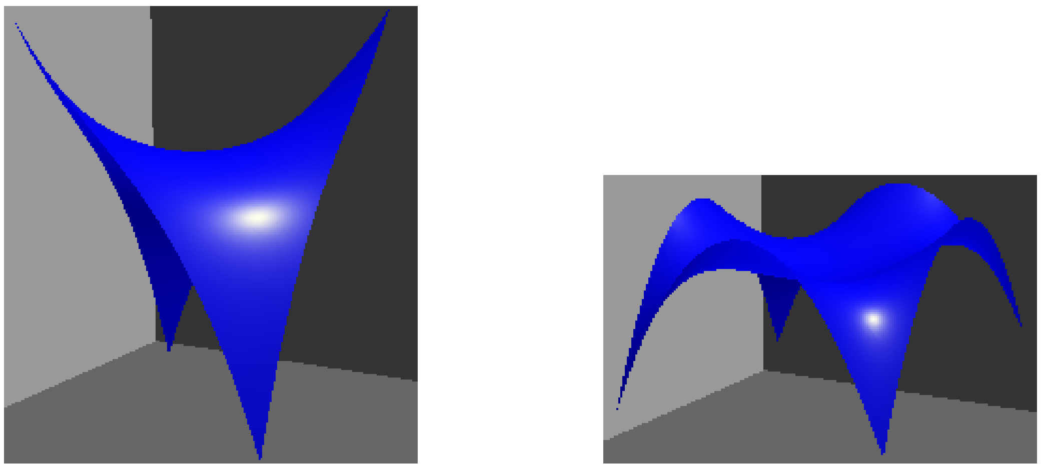


Fig. 5 On the left, Surface 4 by the LIE method. On the right, Surface 5 by the LIE method.

References

1. Katja Bühler. Taylor Models and Affine Arithmetics - Towards a More Sophisticated Use of Reliable Methods in Computer Graphics. 17th Spring Conference on Computer Graphics (SCCG '01).
2. Luiz de Figueiredo and Jorge Stolfi. Self-Validated Numerical Methods and Applications. Brazilian Mathematics Colloquium monograph, IMPA, Rio de Janeiro, Brazil, July 1997.
3. Olaf Knüppel. PROFIL - Programmer's Runtime Optimized Fast Interval Library. 1993.
4. R.E. Moore. A Test for Existence of Solutions to Nonlinear Systems. Siam Journal on Numerical Analysis. Vol.14, No.4 (Sep.,1977),611-615.
5. R.E. Moore and S.T. Jones. Safe Starting Regions for Iterative Methods. Siam Journal on Numerical Analysis. Vol.14, No.6 (Dec.,1977),1051-1065.
6. Arnold Neumaier. Introduction to Numerical Analysis. Cambridge University Press. 2001.
7. Jorge Stolfi. Free C Code. libaa - An Implementation of Affine Arithmetic. http://www.dcc.unicamp.br/ stolfi/EXPORT/software/c/Index.html

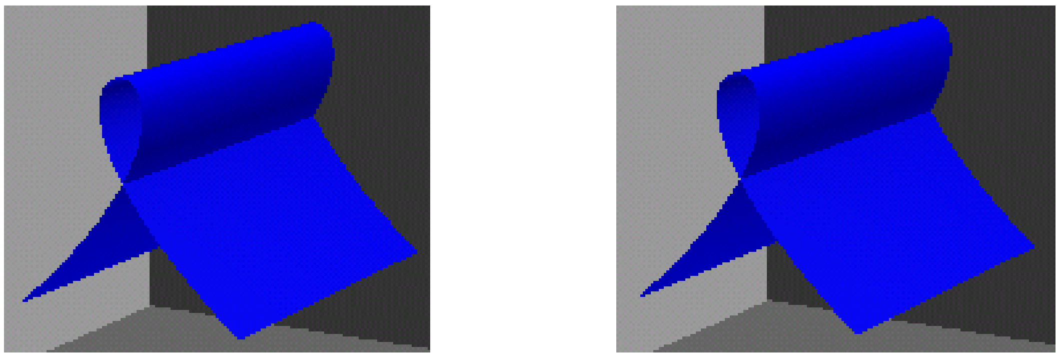


Fig. 6 On the left, Surface 6 by the LIE method. On the right, Surface 6 by Toth version A1.

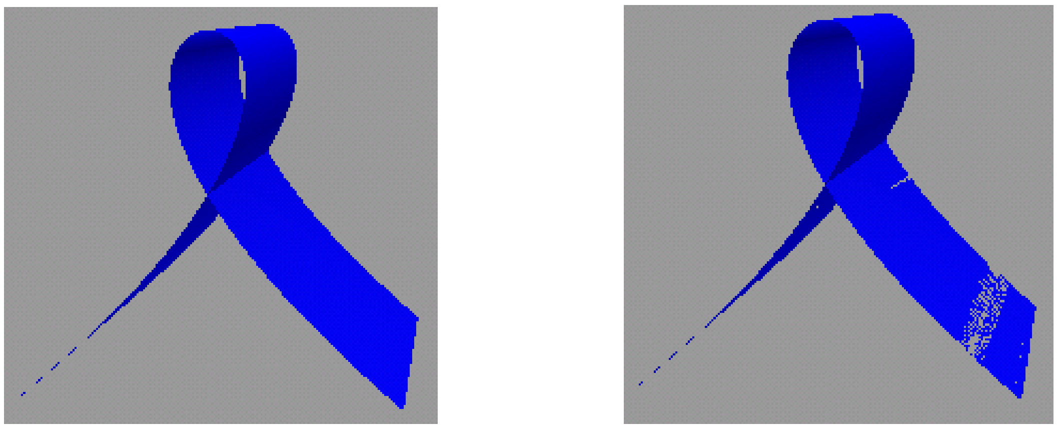
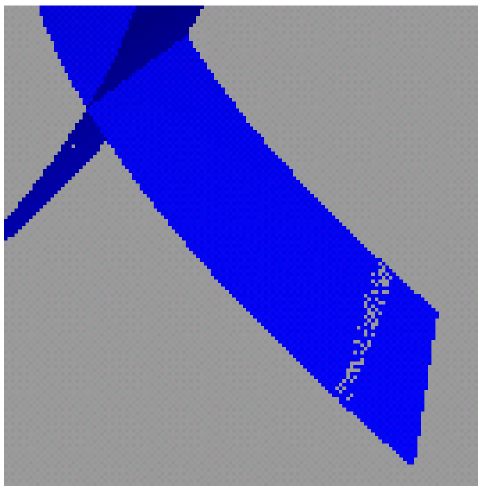


Fig. 7 On the left, Surface 7 by the LIE method. On the right, Surface 7 by Toth version A1.

8. Daniel L. Toth. On Ray Tracing Parametric Surfaces. Computer Graphics (SIGGRAPH '85 Proceedings), vol.19, no.3, July 1985, pp.171-179.



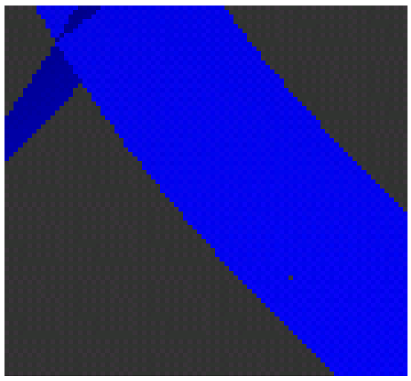


Fig. 8 On the left, Surface 7 by the Toth version A2. On the right, Surface 7 by Toth version B2.