The Visual Computer

Ray tracing general parametric surfaces using interval arithmetic

Wilhelm Barth[1], Roland Lieger, and Michael Schindler

Technical University Vienna,
Department of Computer Graphics,
Karlsplatz 13/186, A-1040 Vienna, Austria.
[1]e-mail: barth@eichow.tuwien.ac.at

This paper describes an algorithm for ray tracing general parametric surfaces. After dividing the surface adaptively into small parts, a binary tree of these parts is built. For each part a bounding volume is calculated with interval arithmetic . From linear approximations and intervals for the partial derivatives it is possible to construct parallelepipeds that adapt the orientation and shape of the surface parts very well and form very tight enclosures. Therefore we can develop an algorithm for rendering that is similar to that used with Bèzier and B-spline surfaces, where the bounding volumes are derived from the convex hull property. The tree of enclosures (generated once in a preprocessing step) guarantees that each ray that hits the surface leads to an iteration on a very small surface part; this iteration can be robustly (and very quickly) performed in real arithmetic.

Key words: Ray tracing – Parametric surfaces – Interval arithmetic – Bounding volumes

Correspondence to: W. Barth

1 Introduction

Ray tracing is a very nice method for generating colored pictures of high quality. Its disadvantage, however, is the enormous amount of computing time that is necessary. Most of this time is spent searching for the point where a ray hits a part of the scene to be rendered.

Many methods have been developed to speed up the calculations. The most popular approach is the use of bounding volumes. For scenes built up from geometric primitives like spheres, boxes, pyramids, etc., in a CSG tree, many authors construct bounding volumes, boxes, or spheres, for parts of the scene with the intention of enabling a faster search for those objects that might be hit (Whitted 1980; Weghorst et al. 1984; Kay and Kajiya 1986). When only approximations for the surface parts and no enclosures (e.g., Lane et al. 1980) are used, it is possible that some rays that hit the surface do not hit the approximation, and therefore the algorithm cannot find an intersection.

Similar methods can be developed for ray tracing Bézier and B-spline surfaces. The fact that the surface is enclosed by the convex hull of its control points makes constructing a bounding volume very easy. Some authors use rectangular boxes for this purpose (Sweeney and Bartels 1986), the boxes may be oriented (Yen et al. 1991) to match the direction of the surface parts. In Barth (1990), Giger-Hofmann (1992) and Barth and Stürzlinger (1993) much effort has been undertaken to construct very tight bounding volumes. In these papers parallelepipeds are used. Their orientation and the angles between their edges are chosen in such a way that they enclose the part as tightly as possible; in general these parallelepipeds are not rectangular.

There is no convex hull property with the general parametric surfaces we treat in this paper. Therefore other methods must be found for calculating tight bounding volumes. We use interval arithmetic. Again we choose parallelepipeds, now computed from values of the parameter functions and enclosing intervals for their partial derivatives. Section 2 gives a detailed description of the construction method. An implementation of this method has been given by Lieger (1992).

In Sect. 3 we describe the tools — interval arithmetic and automatic differentiation — that serve for the calculation of inclusion intervals required in Sect. 2. The application to ray tracing is shown in Sect. 4. We use a binary tree of bounding volumes for

The Visual Computer (1994) 10:363–371

parts of the surfaces, similar to that of Barth (1990). Additionally, the leaves of this tree contain approximations of the small surface parts to allow the computation of a very accurate starting point for the Newton–Raphson iteration. Details of the partition procedure for the surface and of the iteration method are omitted. They are very similar to those used by Barth and Stürzlinger (1993) for Bézier and B-spline surfaces.

Toth (1985), Giger (1989), and Mitchell (1990 and 1991) use interval arithmetic for the iteration. This prevents any failures, but it is very time consuming. We prefer to employ interval arithmetic only in the preprocessing step for calculating bounding volumes, but iteration is performed in faster real arithmetic. Robustness comes from careful subdivision and from good starting points. Nishita et al. (1990) use a very different method, Bézier clipping. It combines subdivision and calculation of ray-patch intersection performed with interval arithmetic. However, this method is restricted to Bézier surfaces. Another completely different interval method is proposed by Enger (1992). He calculates an interval for the color of a surface part, and if this interval is small enough, all pixels belonging to this part get the calculated color. Results are given only for cubic B-spline surfaces. Other methods for acceleration of ray tracing, e.g., voxel or octree methods and ray coherence, may be used in combination with bounding volumes, but are not treated in this paper see e.g. Lischinski and Gonczarowski (1990).

2 Calculation of bounding volumes

This paper deals with parametric surfaces $s(u, v)$ described by three parametric functions x, y, and z depending on two parameters u and v. The parameter domain is rectangular:

$$s(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} \quad \begin{matrix} u \in \mathrm{U} = [\underline{u}, \bar{u}] \\ v \in \mathrm{V} = [\underline{v}, \bar{v}] \end{matrix} \tag{2.1}$$

In our discussion we always assume that the parameter functions are continuous, and that the partial derivatives exist and are also continuous in the whole domain. Additionally we are restricted to the case in which the normal vector $s_u \times s_v$ is uniquely defined at each point in the domain. This means that we only use regular parametrizations.

Repeatedly we need intervals of real numbers. We denote them with capitals; a bar below/above an identifier means its lower/upper bound.

For calculating bounding volumes we require intervals that include the set of values of a parameter function and intervals including its partial derivatives:

$$\begin{aligned} &x(u,v) \in \mathrm{X} \\ &x_u(u,v) \in \mathrm{X}_u \quad \text{for all } u \in \mathrm{U},\ v \in \mathrm{V} \\ &x_v(u,v) \in \mathrm{X}_v \end{aligned} \tag{2.2}$$

Such intervals are calculated by interval arithmetic as described in Sect. 3.1. It is a well known disadvantage of interval arithmetic that the resulting intervals may overestimate the true range by a significant amount. In our application such intervals are used only for small parts of the surface with a small rectangular parameter domain. In this case the disadvantage is tolerable. Additionally, we discuss some methods of reducing the effects of this disadvantage.

We begin with a 2D curve $c(t)$ given by the parametric representation

$$c(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \quad t \in \mathrm{T} = [\underline{t}, \bar{t}] \tag{2.3}$$

Under premises equivalent to those already mentioned, we may apply the mean value theorem of differential calculus, e.g., for $x(t)$

For all $t, t_o \in \mathrm{T}$ exists a $\tau \in [t, t_0]$ so that

$$x(t) = x(t_0) + (t - t_0)x'(\tau) \tag{2.4}$$

Note that arithmetic expressions containing intervals must be evaluated by interval arithmetic, and the results are intervals; see Sect. 3.1 of this paper or Moore (1966), Alefeld and Herzberger (1974), Neumaier (1990).

With an inclusion interval X′ for the derivative (defined according to equation 2.2) we get:

$$x(t) \in x(t_0) + (t - t_0)\mathrm{X}' \quad \text{for all } t, t_0 \in \mathrm{T} \tag{2.5}$$

and a corresponding equation for $y(t)$. Taking $\bar{t}$ for t this yields:

$$c(\bar{t}) \in R(\bar{t}) = c(t_0) + (\bar{t} - t_0)\begin{pmatrix} \mathrm{X}' \\ \mathrm{Y}' \end{pmatrix} \tag{2.6}$$

The endpoint $c(\bar{t})$ of the curve lies within the shaded rectangle $R(\bar{t})$ of Fig. 1.

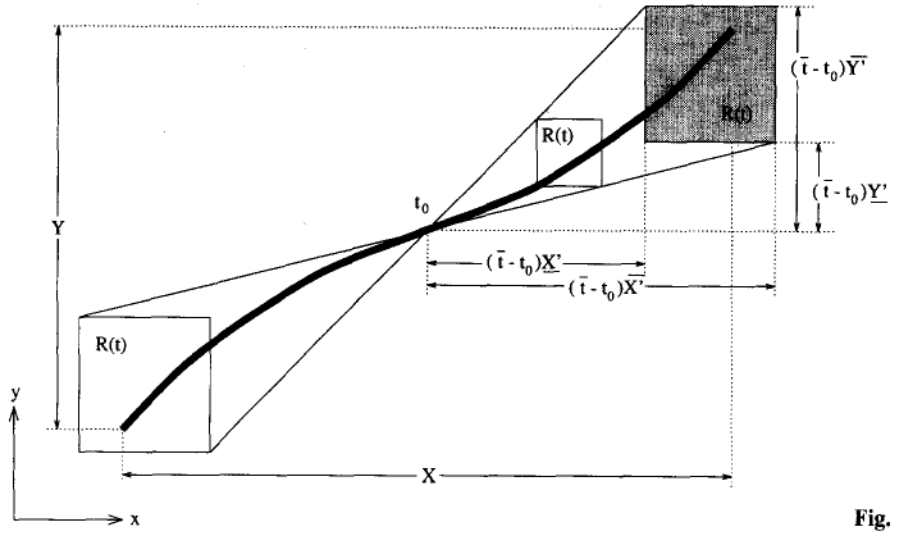


Fig. 1. Bounding area for a 2D curve

For each other point of the curve, an inclusion rectangle can be constructed by proportionally reducing all distances. This construction may be extended in the other direction as far as $\underline{t}$. The result is:

> The curve of Eq. 2.3 is completely included in the bounding area consisting of the two outermost rectangles $R(\bar{t})$ and $R(\underline{t})$ and the angular region connecting them.

For our application we require the following corollary:

> Each convex area, e.g., each parallelogram, enclosing the 2×4 vertices of the outermost rectangles is a bounding area for $c(t)$.

A tight enclosure may be constructed using a parallelogram with two edges parallel to the tangent of $c(t)$ in t_0 or parallel to a secant.
It is very easy to extend this result to 3D curves: Eqs. 2.3 and 2.6 with an additional $z(t)$ now lead to three intervals that define two boxes $B(\bar{t})$ and $B(\underline{t})$, orthogonal and parallel to the axes. Connecting these two boxes linearly, we get a pyramidal volume that is guaranteed to form a bounding volume for the 3D curve. Figure 2 shows a u and a v line of a surface, together with the four boxes enclosing the endpoints. Figure 3 contains complete bounding volumes (yellow) for the u and v line. The corollary is now:

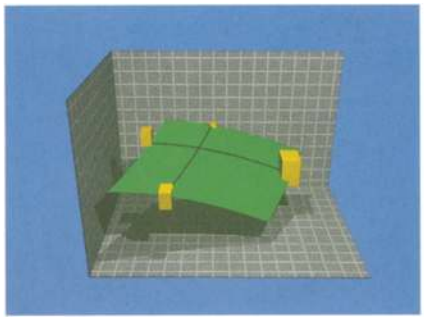
> Each convex volume, e.g., each oriented and nonrectangular parallelepiped enclosing the 2×8 vertices of the two outermost boxes $B(\bar{t})$ and $B(\underline{t})$ is a bounding volume for the parametric 3D curve $c(t)$.

Now we construct bounding volumes for surfaces. With the mean value theorem for two independent variables, we can derive (similarly to Eq. 2.5) the inclusion

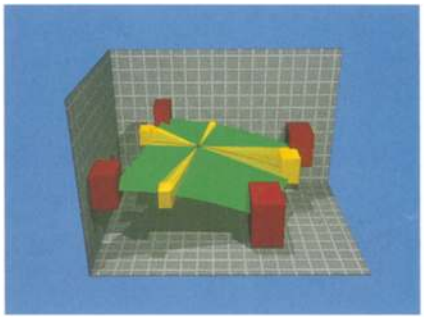
$$x(u,v) \in x(u_0,v_0) + (u-u_0)\mathrm{X}_u + (v-v_0)\mathrm{X}_v \quad \text{for all } u,u_0 \in \mathrm{U} \text{ and } v,v_0 \in \mathrm{V} \tag{2.7}$$

and corresponding equations for y and z, where X_u and X_v are intervals as defined by Eq. 2.2.
We first construct the four bounding boxes $B(u_0,\bar{v})$, $B(u_0,\underline{v})$, $B(\bar{u},v_0)$, $B(\underline{u},v_0)$ at the endpoints of the u and v lines crossing (u_0,v_0) (yellow in Fig. 2) with the method for curve inclusion as explained before. Then we follow the edges to reach the four corners of the surface part; the four boxes $B(\underline{u},\underline{v})$, $B(\underline{u},\bar{v})$, $B(\bar{u},\underline{v})$, $B(\bar{u},\bar{v})$ containing the corner points (red in Fig. 3) are constructed by starting at the vertices of the yellow boxes and taking the bounds of the intervals X_u and X_v as gradients, e.g., for $s(\bar{u},\bar{v})$, so that

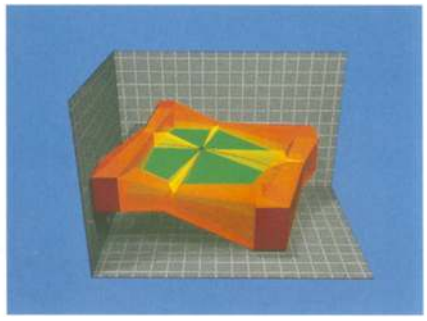
$$s(\bar{u},\bar{v}) \in B(\bar{u},\bar{v}) = s(u_0,v_0) + (\bar{u}-u_0)\begin{pmatrix}\mathrm{X}_u\\ \mathrm{Y}_u\\ \mathrm{Z}_u\end{pmatrix} + (\bar{v}-v_0)\begin{pmatrix}\mathrm{X}_v\\ \mathrm{Y}_v\\ \mathrm{Z}_v\end{pmatrix} \tag{2.8}$$



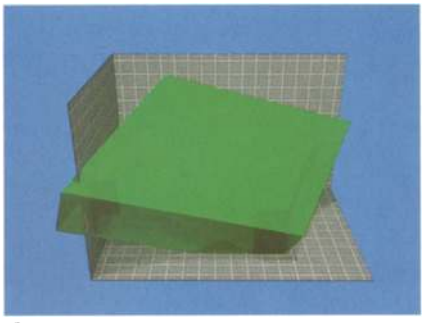
2



3



4



5

Fig. 2. Surface part with u and v lines and boxes including their endpoints

Fig. 3. Pyramidal bounding volumes for u and v lines and corner boxes

Fig. 4. The wreath

Fig. 5. The enclosing parallelepiped

With the same arguments as used in the simpler cases, we prove that all points on the border of the surface lie within the so constructed wreath (Fig. 4). The whole surface is enclosed by each convex volume that encloses the four corner boxes (Fig. 5). From this follows the theorem:

> Each convex volume, e.g., each oriented and nonrectangular parallelepiped enclosing the 4×8 vertices of the four corner boxes $B(\underline{u}, \underline{v})$, $B(\underline{u}, \bar{v})$, $B(\bar{u}, \underline{v})$, $B(\bar{u}, \bar{v})$ is a bounding volume for the surface $s(u, v)$ from Eq. 2.1.

This theorem allows the following simple construction of a bounding volume for a surface part. First construct an approximating nonrectangular parallelogram by taking the tangential plane at (u_0, v_0) and cutting out that part belonging to the actual (u, v) domain. This may be interpreted as the bottom and top plane of a parallelepiped of zero height. Then, in a second step, expand this volume until all vertices of the four corner boxes are included. This expansion is mainly done in the height direction, but the other two directions must also be considered. For details of these calculations see Barth and Stürzlinger (1993).

In the case of a small surface part, that is, after subdividing finely enough, this yields a very tight enclosure. The parallelogram is a linear approximation; deviations from the surface are caused only by higher order terms of the Taylor expansion. Interval arithmetic leads to small intervals for the derivatives (if not, we must make further subdivisions); this yields small corner boxes and therefore a thin parallelepiped.

The question arises: why use such a complicated construction of bounding volumes instead of using the intervals generated by evaluating the arithmetic expressions for the three parameter functions X, Y, Z? The latter method independently calculates one inclusion interval for each coordinate direction, yielding a rectangular bounding box parallel to the axes for every surface part. Figure 1 explains the 2D case: it is clear that in the case of small parts this box is much larger than our bounding area, because it must contain at best the whole $c(t)$, and overestimation will also occur with this interval evaluation. The 3D case is similar: a thin and sloping parallelepiped with an acute angle is much tighter than a rectangular box parallel to the axes.

Note that the approximating parallelograms of all surface parts do not necessarily build a continuous surface without cracks (Fig. 7 — right hand side). However, the union of the bounding volumes includes the whole surface. Therefore each ray that hits the surface also hits at least one parallelepiped, which in turn leads to the start of an iteration (Sect. 4).

3 Interval arithmetic and automatic differentiation

In the previous section we needed enclosing intervals for the derivatives of parameter functions and used them for building bounding volumes for surface parts. Now we will explain the method for calculating such intervals.

3.1 Interval arithmetic

We consider real-valued functions depending on one or two independent variables. Normal arithmetic solves the problem of calculating the real function value at a single point in parameter space. Interval arithmetic generates bounds for the set of function values that belong to all parameter values contained in given intervals. We define the set of function values, e.g., for a parameter function of a surface, by

$$\mathrm{F(U,V)} = \{f(u,v) | u \in \mathrm{U}, v \in \mathrm{V}\} \tag{3.1}$$

In the case of continuous functions such a set is an interval. The definitions for the partial derivatives are similar and need not be written down explicitly. A single operation of interval arithmetic is defined by

$$\mathrm{A}\ op\ \mathrm{B} = \{a\ op\ b | a \in \mathrm{A}, b \in \mathrm{B}\} \tag{3.2}$$

Especially for the basic arithmetic operations we have the explicit definitions

$$\begin{aligned}
\mathrm{A} + \mathrm{B} &= [\underline{a} + \underline{b}, \bar{a} + \bar{b}] \\
\mathrm{A} - \mathrm{B} &= [\underline{a} - \bar{b}, \bar{a} - \underline{b}] \\
\mathrm{A} \times \mathrm{B} &= [\min\{\underline{a} \times \underline{b}, \underline{a} \times \bar{b}, \bar{a} \times \underline{b}, \bar{a} \times \bar{b}\}, \\
&\qquad \max\{\underline{a} \times \underline{b}, \underline{a} \times \bar{b}, \bar{a} \times \underline{b}, \bar{a} \times \bar{b}\}] \\
\mathrm{A} / \mathrm{B} &= \mathrm{A} \times [1/\bar{b}, 1/\underline{b}] \text{ if } 0 \notin \mathrm{B}
\end{aligned} \tag{3.3}$$

Normally both operands of these binary operators are intervals. The case of a scalar operand may be included by simply taking an interval with equal lower and upper bounds, e.g., $\mathrm{A} = [a, a]$. Division by an interval containing zero must be avoided since the result is no longer a finite interval.
For basic functions $b(t)$, e.g., square root, square, sine, etc., the interval B(T) for the function value is defined by

$$\mathrm{B(T)} = \{b(t) | t \in \mathrm{T}\}, \tag{3.4}$$

which can be easily computed for monotonic functions (the bounds are the function values at $t = \underline{t}$ and $t = \bar{t}$), but computation may be more complicated if b is not monotonic.
The following method allows the calculation of an enclosing interval

$$\mathrm{I} \supseteq \mathrm{F(U,V)} \tag{3.5}$$

for every function $f(u, v)$ given in the form of an arithmetic expression of the parameters, constants $[c, c]$ and basic functions: Beginning with the components we must apply (3.3) or (3.4) repeatedly to get enclosing intervals for parts of the expression until the final result is obtained.
In general this method does not yield the tightest interval possible, i.e., F(U, V). It only generates an enclosure I (see Eq. 3.5), which might be a large overestimation. We discuss this disadvantage and some methods to cope with it in Sect. 4. It should be mentioned here that some effort is required to write subroutines for all commonly used basic functions that calculate inclusions according to Eqs. 3.4 and 3.5 as tightly as possible.

3.2 Automatic differentiation

If a function $f(t)$ is defined by an arithmetic expression, then it is possible to compute the derivative $f'(t)$ during the evaluation of $f(t)$ for a single value of $\underline{t}$ with little extra effort. This is performed by a special arithmetic for the tuples

$$(f, f') \tag{3.6}$$

This arithmetic serves for the simultaneous calculation of such tuples beginning with the basic operands. Neumaier (1990) calls it *recursive*

differentiation. The four basic operations are:

$$(f,f') + (g,g') = (f + g, f' + g')$$
$$(f,f') - (g,g') = (f - g, f' - g') \quad (3.7)$$
$$(f,f') \times (g,g') = (f \times g, f \times g' + f' \times g)$$
$$(f,f') / (g,g') = (f/g, (f' \times g - f \times g')/(g \times g))$$

As with interval arithmetic, rules for basic functions are derived easily, e.g.,

$$\sin(f,f') = (\sin(f), \cos(f) \times f')$$
$$\sqrt{(f,f')} = (\sqrt{f}, f'/(2\sqrt{f})) \quad (3.8)$$

Again we begin evaluation with the components $(t,1)$ for the independent variable, $(v,0)$ for any other variable, and $(c,0)$ for a constant. Then we repeatedly compute tupels for partial expressions until we reach the complete function. Expanding this to our parameter functions, e.g., $x(u,v)$, with two partial derivatives is an easy task.

We have incorporated this method into our system for ray tracing parametric surfaces. It works as an interpretative system on the parameter functions. Therefore the user only needs to provide the formula for these functions; he is not required to perform the differentiation of the formula.

3.3 Combining interval arithmetic with automatic differentiation

Now we want to describe how to calculate interval enclosures for a parameter function and its two partial derivatives for a rectangular parameter domain, i.e., how to get (X, X_u, X_v) (see Eq. 2.2). The description is very short: take intervals in Sect. 3.2 instead of reals and use interval arithmetic.

4 Ray tracing parametric surfaces

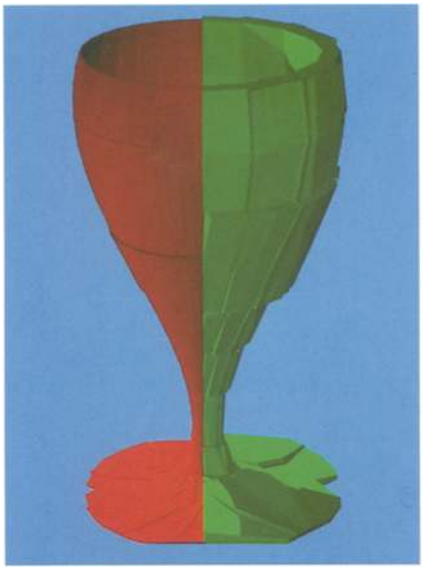
We use our method to speed up ray tracing when applied to parametric surfaces. When doing so, one must shoot a large number of rays (1,000,000 or more) into a computer-modeled scene to generate a realistic picture of the scene. Each of these rays must be intersected with objects of the scene. Some objects allow explicit calculation of intersections (e.g., boxes and spheres), while others, especially the so-called free form surfaces and parametric surfaces require an iterative calculation of intersections. These iterations are very time consuming. Therefore efforts must be made to speed up the calculations; it is extremely important to reduce the number of necessary iterations. This can be achieved by finding suitable, very accurate starting points for all rays that hit the surface and by finding out which rays do not hit the surface at all at a very early stage in the computation.

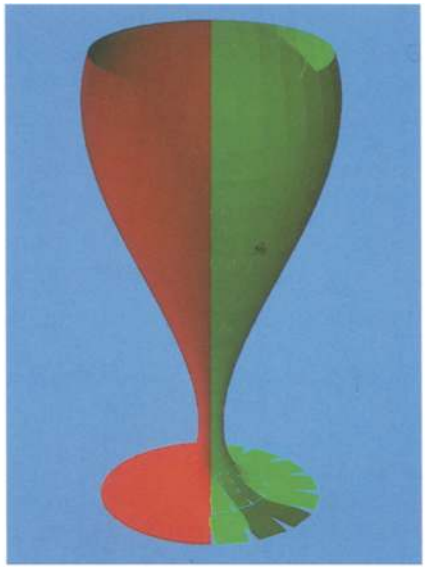
For this purpose we use a preprocessing step similar to that of Barth and Stürzlinger (1993) for Bézier or B-spline surfaces. This step has to be performed before the first ray–surface intersection is calculated. Because of the huge number of rays and because preprocessing is executed only once, even a very time-consuming preparation that yields only a tiny saving per intersection may reduce the total time significantly.

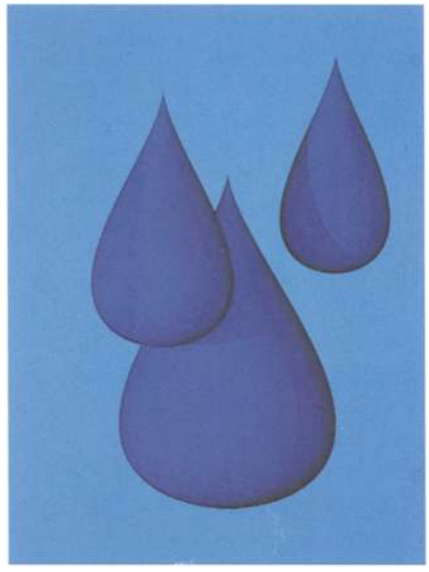
During preprocessing we adaptively subdivide the surface into parts. Each subdivision is done by halving the surface or a part of it, parallel to the u or the v axis. Subdivisions are repeated for all parts that cannot be accurately approximated by plane parallelograms. Whether a surface part is small enough can be seen from our inclusion method (see Sect 2): The red corner boxes of Fig. 3 must be small against the other dimensions of the part.

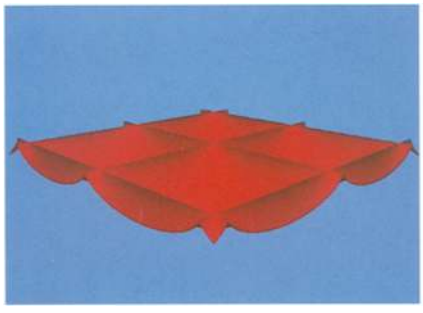
Each leaf contains a linear approximation for the small surface part belonging to it. By intersecting the rays with these approximations we get very good starting points for the Newton–Raphson iteration. Each intermediate node and each leaf of the tree contains an enclosing parallelepiped for its surface part; it is constructed by the method of Sect. 2. We choose parallelepipeds because they are tightly bounding volumes adapting well to the orientation and form of the surface parts.

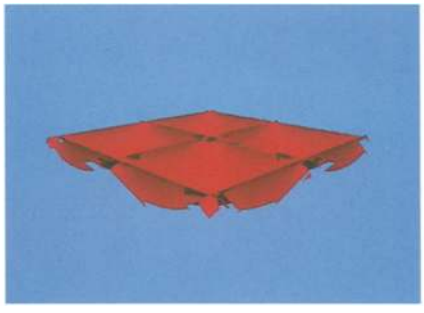
When calculating the intersection of a ray with the surface we must first search the surface parts (leaves of the tree) that may be hit. Beginning at the root we follow only those paths where the parallelepipeds are hit. This test can be performed very quickly. Additionally, if the enclosures are tight enough, only those paths are followed where a hit actually occurs. Rays that miss the surface are excluded from further consideration very early. Finally we reach all leaves the enclosures of











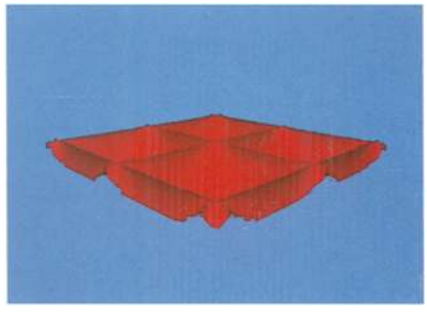


Fig. 6. Parallelopipeds for fine and coarse subdivisions

Fig. 7. Surface and approximating parallelograms

Fig. 8. Rain:
$X(u, v) = \cos(u)*\sin(v)*(1-\cos(v))/2$
$Y(u, v) = \sin(u)*\sin(v)*(1-\cos(v))/2$
$Z(u, v) = \cos(v)$
$U \times V = [-,] \times [0,]$

Fig. 9. Cycloid surface:
$X(u, v) = u + \cos(u)$
$Y(u, v) = v + \cos(v)$
$Z(u, v) = -0.5*(1 - \sin(u))*(1 - \sin(v))$
$U \times V = [0,5] \times [0,5]$

Fig. 10. Errors caused by too coarse subdivision

Fig. 11. Errors caused by stopping iteration too early

which are hit. As mentioned earlier for each of these parts an iteration must be performed.

The method described in previous paragraphs works very well, but experiments show that there is still one remarkable disadvantage. Due to overestimation by the interval arithmetic, the bounding volumes of the surface parts calculated according to the rules of Sect. 2 tend to be much larger than the optimal enclosures. This effect is stronger the larger the parts, i.e., the nearer to the root of the tree of parallelepipeds. Therefore, excluding parts of the tree from further consideration because the corresponding surface part is not hit takes place too seldom in the upper part of the tree. Much time is wasted in this way for searching the leaves hit by the ray.

We solve this problem in the following way: first, bounding volumes for all surface parts are calculated by interval arithmetic, i.e., the tree of bounding volumes is built top down. After this, but still in the preprocessing step, we exploit the fact that every surface part is enclosed by the union of the enclosures of its halves. That means we perform — bottom up — a reduction of all

enclosing parallelepipeds by cutting those slices that do not belong to the parallelepiped of one of its successors. Our experiments show that this method leads to much tighter enclosures.

5 Experiences

On the right side Fig. 6 shows the enclosing parallelepipeds of a very coarse subdivision (64 parts) for a goblet. The surface parts are too large for our method. The finer subdivision (512 parts) shown on the left side allows efficient iteration with correct results.

On the right side Fig. 7 shows the approximating parallelograms for the finer subdivision of Fig. 6. In general these approximations do not form a continuous surface without gaps, but the parallelepipeds do. The surface on the left side was rendered from the finer subdivision. Only an average of 2.7 iterations per ray was required. The preprocessing step, i.e., calculating all 512 parts, their approximating parallelograms, their enclosures, and building up the binary tree of the surface parts took only a few seconds. Ray tracing, however, i.e., calculating the intersections of the rays with the surface for 1000×750 pixels, including antialiasing, required more than 2 h on a VAX Station 3100. For Fig. 8 every drop of the rain has been subdivided adaptively into 420 parts; the average for the iteration was 2.36.

The cycloid surface of Fig. 9 shows that our iteration procedure is stable enough to cope with surfaces containing folds also. The subdivision method automatically divides more finely near the critical folds and then iteration has no problems. Note that the instability of the Newton–Raphson iteration in some extraordinary cases has been overcome by drawing back all values to the parameter domain of the surface part. In the case of Fig. 9 we used a subdivision into 3437 parts; the average number of iterations was again very low (2.36).

In Figs. 10 and 11 we demonstrate the effect of carelessly using the method. The result of a too coarse subdivision of a difficult surface such as our cycloid can be seen. No more than 64 surface parts were generated in Fig. 10. Therefore the iteration fails for many pixels, mainly near the folds and the brink. This failure is interpreted by the program as "no hit", therefore it works like seeing the background (blue) or another part of the surface from the shadowed side (black). Similar results can be recognized in Fig. 11, where we used finer subdivision (256 parts), but we restricted the maximal number of iterations to two.

References

Alefeld F, Herzberger J (1974) Einführung in die Intervallrechnung (in German). B.I. Wissenschaftsverlag, Mannheim

Barth W (1990) Effizientes Ray-Tracing für Bézier- und B- Spline Flächen. In: Encarnacao, Hoschek, Rix (eds): Geometrische Verfahren der Graphischen Datenverarbeitung (in German). Springer Berlin, pp 180–197

Barth W, Stürzlinger W (1993) Efficient ray tracing for Bèzier and B-spline surfaces. Computers & Graphics 17:423–430

Enger W (1992) Interval ray tracing – a divide and conquer strategy for realistic computer graphics. Visual Comput 9:91–104

Giger C (1989) Ray tracing polynomial tensor product surfaces. In: Hansmann, Hopgood, Strasser (eds): Proc Eurographics, North Holland, Amsterdam, New York, Oxford, Tokyo, pp 125–136

Giger-Hofmann C (1992) Ein Ray-Tracing-Verfahren zur Visualisierung polynomialer Tensorproduktflächen (in German). Dissertation, Technische Hochschule Darmstadt, Germany

Kay T, Kajiya J (1986) Ray tracing complex scenes. Comput Graph proc SIGGRAPH 20:269–276

Lane J, Carpenter L, Whitted T, Blinn J (1980) Scan line methods for displaying parametrically defined surfaces. Commun ACM 23:23–34

Lieger R (1992) Ray Tracing allgemeiner Flächen in Parameter-Darstellung (in German). Diplomarbeit, Technische Universität Wien, Vienna

Lischinski D, Gonczarowski J (1990) Improved techniques for ray tracing parametric surfaces. Visual Comput 6:134–152

Mitchell DP (1990) Robust ray intersection with interval arithmetic. graphics interface, pp 68–74

Mitchell DP (1991) Three applications of interval analysis in computer Graphics. Tutorial Notes SIGGRAPH

Moore RE (1966) Interval Analysis. Englewood Cliffs, New York

Neumaier A (1990): Interval Methods for Systems of Equations. Cambridge University Press, New York

Nishita T, Sederberg T, Kakimoto M (1990) Ray tracing trimmed rational surface patches. Comput Graph 24:337–345

Sweeney M, Bartels R (1986) Ray tracing free-form B-spline surfaces. IEEE Comput Graph Appl 6:41–49

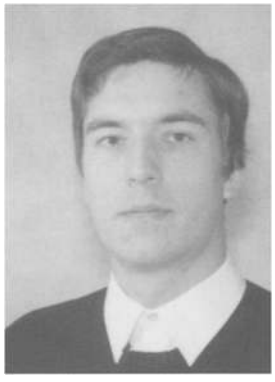
Toth D (1985) On ray tracing parametric surfaces. Proc SIGGRAPH 85:171–179

Weghorst H, Hooper G, Greenberg D (1984) Improved computational methods for ray tracing. ACM Trans Graph 3:52–69

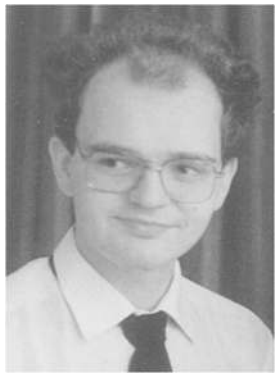
Whitted T (1980) An improved illumination model for shaded display. Commun ACM 23:343–349

Yen J, Spach S, Smith M, Pulleyblank R (1991) Parallel boxing in B-spline intersection. IEEE Comput Graph Appl 11:72–79

WILHELM BARTH received his Dr. rer. nat. degree in mathematics from the Technical University of Darmstadt, Germany. He worked in numerical analysis there, primarily in interval mathematics. Since 1973 he has been a professor of computer science at the Technical University of Vienna, Austria. His research areas are computer graphics, algorithms, and data structures.



ROLAND LIEGER received his Diploma of Engineering degree in computer science from the Technical University of Vienna in 1992. In his thesis he developed and implemented important parts of the algorithms of this paper. Now he is working on his Ph.D. thesis.



MICHAEL SCHINDLER received his Diploma of Engineering degree from the Technical University of Vienna in 1991 with a thesis on real-time systems. He currently works as a research assistent in the Department for Computer Graphics of that university. His special interest is locating and overcoming bottlenecks.