The Visual Computer

Interval Ray Tracing – a divide and conquer strategy for realistic computer graphics

Wolfgang Enger*

Institut für Angewandte Mathematik,
Universität Freiburg, W-7800 Freiburg,
Federal Republic of Germany

A new method for ray tracing parametric surfaces is developed. The method uses a divide-and-conquer strategy for rapid finding of the sectors of constant intensity. Techniques from interval analysis allow reducing the number of ray/surface intersections that must be computed. Main results are presented in terms of solving a general system of nonlinear equations, and thus can be extended to a large class of problems. Examples with B-spline surfaces demonstrate an improvement in computation time over a conventional method with a factor between 1.5 and 3.0. Additionally, a way of preventing "cracks" in triangulating parametric surfaces is shown.

Key words: Ray tracing – Interval analysis – Parametric surfaces – B-splines – Triangulation

* Now working at ip Info Process GmbH, Gewerbestrasse 4, W-7801 Buchenbach, Federal Republic of Germany

1 Introduction

Ray tracing is a powerful approach to realistic image generation. It is a conceptually simple and powerful method, taking into account the effects of shadows cast, multiple reflection, and refraction for transparent objects. The most current implementations are first published by Whitted (1980). An excellent introduction and an overview of current research is given in Glassner (1989); some efficient datastructures can be found in Müller (1988).

Ray-tracing techniques reduce the problem of surface generation to the computation and shading of intersection points of rays with surfaces. For complex unstructured objects, the calculation of a point of intersection is equivalent to solving a system of nonlinear equations $F(x)=0$ (F: $\mathbf{R}^3 \to \mathbf{R}^3$). This can be done, for instance, using *Newton's Method* (see Faux and Pratt 1979 for parametric surfaces). To overcome the problem of finding a starting point, Toth (1985) uses the *Interval Newton Method* (Krawczyk 1969).

Reduced computation time may be achieved by generating a polygonal approximation of the surface to be rendered, and then performing ray tracing on the resulting polygons. In order to avoid a drastic increase in the number of intersection points to be calculated, the given scene may be partitioned by a *regular grid* (Schmitt et al. 1988) or, alternatively, each object may be representated by a (binary) tree, whose leaves are "primitives," e.g., linear subsurfaces. One method that combines these techniques is *accelerated ray tracing* (Glassner 1984; Yamaguchi et al. 1984; Fujimoto and Iwata 1986), where the rays are traced through a so-called *octree structure* of cells. Another method for rational Bézier patches – the Bézier clipping – is a combination of subdivision and numerical methods (Nishita et al. 1990).

Thirion (1990) uses *clusters of rays*, consisting of a fixed number of 4×4 or 8×8 rays, to reduce the number of intersection points to be calculated.

In this paper, we present a new technique for finding the intersection of a ray with a parametric surface. We use the fact that pictures – especially computer-generated pictures – often contain relatively large uniformly colored sections so that it is unsatisfactory to perform similar calculations for all pixels on the screen eventhough many adjacent pixels could be eventually colored together. Our goal is to obtain these common regions with only a few arithmetical operations, whereby the calculation with clusters requires mathematical methods that

The Visual Computer (1992) 9:91–104

extend real numbers to real intervals. Inspired to the present investigations by Mudur and Koparkar (1984), Koparkar and Mudur (1985), and Neumaier (1988a), we use methods of interval analysis to calculate the set of intersection points of ray clusters with objects in the scene, i.e., we compute an enclosure for the solution set of the system of *"interval equations"* $0 \in F(x)$ (F: $\mathbf{IR}^3 \to \mathbf{IR}^3$). With this set, we compute an enclosure for the intensity, and, unless the intensity interval is "thin" enough, we partition the cluster into two (or more) clusters. Although the calculation of the solution set is more complicated in the interval case, the saving in the number of such calculations results in an overall improvement in calculation time over the conventional method for bicubic B-spline surfaces with a factor between 1.5 and 3.0, depending on the resolution and the number of colors to be presented.

For the reader who is not familiar with interval analysis or B-splines, we present in Sects. 2–4 some basic facts of *interval analysis* and an introduction to *B-splines*. In Sect. 5, we apply the methods of the first two sections to B-spline functions to obtain a new and effective method for calculating all intersection points of ray clusters with objects in a scene. A method for dividing (triangulating) parametrically defined surfaces avoiding "cracks" during the subdivision process is described in Sect. 6. The algorithms mainly based on the results of Sect. 5 are presented in Sect. 7. Some experimental results complete the paper (Sect. 8).

Notation: We denote by

$\mathbf{R}, \mathbf{R}^n, \mathbf{R}^{m \times n}$ the set of real numbers, n-vectors, and $m \times n$-matrices, respectively.

$\mathbf{IR}, \mathbf{IR}^n, \mathbf{IR}^{m \times n}$ the set of intervals, interval n-vectors, and interval $m \times n$-matrices, respectively.

2 Interval analysis: basic facts, centered forms

In this section, we present the basic notions from interval analysis required in our investigations. More detailed introductions to this topic can be found in Moore (1966), Alefeld and Herzberger (1983), and Neumaier (1990).

We begin our discussion with the definition of intervals and related notions:

A (real) *interval* is a set of the form

$$x \equiv [\underline{x}, \bar{x}] := \{\tilde{x} \in \mathbf{R} \mid \underline{x} \le \tilde{x} \le \bar{x}\},$$

where $\underline{x}, \bar{x}$ are elements of $\mathbf{R}$ with $\underline{x} \le \bar{x}$. The (possibly empty) open interval $]\underline{x}, \bar{x}[$, the *interior* of x, is denoted by $\mathrm{int}(x) \equiv (\underline{x}, \bar{x}) := \{\tilde{x} \in \mathbf{R} \mid \underline{x} < \tilde{x} < \bar{x}\}$. An interval is called *thin* if $\underline{x} = \bar{x}$ and *thick* if $\underline{x} < \bar{x}$. The *midpoint* of an interval x is the point $\check{x}$ with $\mathrm{mid}(x) \equiv \check{x} := (\bar{x} + \underline{x})/2$. The *radius* of x is $\mathrm{rad}(x) := (\bar{x} - \underline{x})/2$. The *magnitude* of x is $\mathrm{mag}(x) \equiv |x| := \max\{|\tilde{x}| \mid \tilde{x} \in x\}$. If S is a nonempty bounded subset of $\mathbf{R}$ we denote by $[\,]S := [\inf(x), \sup(x)]$ the *hull* of S, i.e., the tightest interval enclosing S.

In order to calculate familiarily with intervals, we extend the order relations and elementary operations from real numbers to intervals in an obvious way.

The *order relations* $\mathrm{OR} \in \{<, \le, \ge, >\}$ are extended to interval arguments by defining $x \,\mathrm{OR}\, y$: $\langle = \rangle \tilde{x} \,\mathrm{OR}\, \tilde{y}$ for all $\tilde{x} \in x$, $\tilde{y} \in y$.

Elementary operations $\mathrm{EO} \in \{+, -, *, /, **\}$ are defined on the set of intervals by putting

$$x \,\mathrm{EO}\, y := [\,]\{\tilde{x} \,\mathrm{EO}\, \tilde{y} \mid \tilde{x} \in x, \tilde{y} \in y\} = \{\tilde{x} \,\mathrm{EO}\, \tilde{y} \mid \tilde{x} \in x, \tilde{y} \in y\}$$

for all x, $y \in \mathbf{IR}$ such that $\tilde{x} \,\mathrm{EO}\, \tilde{y}$ is defined for all $\tilde{x} \in x$ and $\tilde{y} \in y$.

Remarks

1. The definition of the division x/y is restricted to intervals y not containing zero.
2. In most cases, multiplication and division of intervals can be performed with only two real arithmetic operations (Neumaier 1990).

The definitions and basic notions from interval analysis can be easily extended to vectors and matrices.

An *interval* $m \times n$*-matrix* is a matrix $A = (A_{ik})$ of intervals $A_{ik} \in \mathbf{IR}$ (we denote the elements of A by A_{ik} instead of a_{ik}).

We interpret $A \in \mathbf{IR}^{m \times n}$ as the set of all matrices $\tilde{A} \in \mathbf{R}^{m \times n}$ with $\tilde{A}_{ik} \in A_{ik}$ for $i = 1, \ldots, m$; $k = 1, \ldots, n$. Interval n-vectors are considered as interval $n \times 1$-matrices. We define:

$$\underline{A} \equiv \inf(A) := (\underline{A}_{ik}); \quad \bar{A} \equiv \sup(A) := (\bar{A}_{ik});$$

$$\check{A} \equiv \mathrm{mid}(A) := (\check{A}_{ik})$$

$$\mathrm{rad}(A) := (\mathrm{rad}(A_{ik})); \quad |A| := (|A_{ik}|)$$

$$\mathrm{int}(A) := \{\tilde{A} \in A \mid \underline{A}_{ik} < \tilde{A}_{ik} < \bar{A}_{ik} \text{ whenever } \underline{A}_{ik} \ne \bar{A}_{ik}\}$$

$$A \,\mathrm{OR}\, B: \langle = \rangle A_{ik} \,\mathrm{OR}\, B_{ik} \quad \text{for } i = 1 \ldots m;$$

$$k = 1 \ldots n; \mathrm{OR} \in \{<, \le, \ge, >\}.$$

We now introduce centered forms for vector-valued functions in real variables, which we use to solve systems of "interval equations," whereby ranges of functions have to be enclosed.
If f is a vector-valued function in n real variables, the interval evaluation over a rectangular box $x \in \mathbf{IR}^n$ provides an enclosure for the range $f^*(x) = \{f(\tilde{x}) | \tilde{x} \in x\}$ with an overestimation of order $\mathfrak{O}(\mathrm{rad}(x))$. Because this may be large in adverse circumstances, we now provide a method for enclosing the range $f^*(x)$ in such a way that the overestimation remains small for sufficiently narrow boxes x.

Proposition 1 (Neumaier 1990). *(i) The centered form of f with center $\tilde{z} \in x$ and slope $s \in \mathbf{IR}$*

$$f_z(x, \tilde{z}) := f(\tilde{z}) + s \cdot (x - \tilde{z})$$

is an enclosure for the range $f^*(x)$ over the box $x \in \mathbf{IR}^n$ if, for all $\tilde{x} \in x$, there is a $\tilde{s} \in s$ such that $f(\tilde{x}) = f(\tilde{z}) + \tilde{s} \cdot (\tilde{x} - \tilde{z})$.
(ii) All centered forms have the quadratic approximation property, i.e., for narrow boxes x and Lipschitz continuous interval extentions of s, the overestimation of the range is of order $\mathfrak{O}(rad(x)^2)$.

In this paper we only use *mean value forms*

$$f_m(x) := f(\check{x}) + f'(x)(x - \check{x}),$$

because the derivative is easy to compute and $\tilde{z} = \check{x}$ is an optimal choice to obtain minimal radius (Baumann 1988).
Alternatively, if f is a polynomial with interval coefficients, an enclosure for the range of this function may be computed as proposed by Dussel and Schmitt (1970).

Lemma 2 (Enger 1990). *Let $p(u) = \sum_{i=0}^{n} b_i u^i$ be a polynomial of n-th degree with coefficients $b_i \in \mathbf{IR}$ and $u \in \mathbf{ID} \subseteq \mathbf{IR}$, $u \geq 0$. We denote by $p^*(u)$ the range of this polynomial on $\mathbf{ID}$. Then*

$$p^*(u) = \left[\min_{\tilde{u} \in u} \left(\sum_{i=0}^{n} \underline{b}_i u^i \right), \max_{\tilde{u} \in u} \left(\sum_{i=0}^{n} \overline{b}_i u^i \right) \right].$$

In Sect. 5, we shall use this result for directly enclosing the range of cubic B-splines over a parameter interval.

3 Nonlinear systems of equations

A special case in our considerations is the computation of the intersection points of a single ray (viewed as a real vector) with the objects in the scene. This leads to a non-linear system of equations $F(x) = 0$ (F: $\mathbf{R}^3 \to \mathbf{R}^3$), i.e., the computation of an enclosure for all *zeros* x^* of a continuous function F: $\mathbf{D}_0 \subseteq \mathbf{R}^n \to \mathbf{R}^n$ in a given subset $\mathbf{D}$ of $\mathbf{D}_0$.
We first present some definitions further required.

Definition 3. If $A \in \mathbf{IR}^{n \times n}$ then we call

A regular : $\Leftrightarrow$ $\tilde{A}$ is regular for all $\tilde{A} \in A$.
A strongly regular : $\Leftrightarrow$ $\check{A}^{-1} A$ is regular.
A H-matrix : $\Leftrightarrow$ $\langle A \rangle u > 0$ for some positive $u \in \mathbf{R}^n$.
Here $\langle A \rangle_{ii} := \min \{ |\tilde{A}_{ii}| \, | \tilde{A}_{ii} \in A_{ii} \}$ and $\langle A \rangle_{ik} := -|A_{ik}|$ for $i \neq k$.

Remark. Every regular real matrix is strongly regular.

Definition 4. Let F: $\mathbf{D}_0 \subseteq \mathbf{R}^n \to \mathbf{R}^m$. If there is an A such that for every $\tilde{x}$, $\tilde{y} \in \mathbf{D} \subseteq \mathbf{D}_0$: $F(\tilde{x}) - F(\tilde{y}) = \tilde{A}(\tilde{x} - \tilde{y})$ for some $\tilde{A} \in A$, then A is called a *Lipschitz matrix* for F on $\mathbf{D}$. If A is in addition closed, convex, and bounded then $A \in \mathbf{IR}^{m \times n}$.
Note that if F is continuously differentiable on $\mathbf{D}$ and $x \in \mathbf{ID}$ then the matrix $A = F'(x)$ is a Lipschitz matrix for F whenever the interval extension $F'(x)$ is defined.

Definition 5. Let F: $\mathbf{D}_0 \subseteq \mathbf{R}^n \to \mathbf{R}^n$ be *Lipschitz continuous* on $\mathbf{D} \subseteq \mathbf{D}_0$, i.e., $\|F(\tilde{x}) - F(\tilde{y})\| \leq \delta \|\tilde{x} - \tilde{y}\|$ for all $\tilde{x}$, $\tilde{y} \in \mathbf{D}$ with a *Lipschitz constant* δ, and let A be a Lipschitz matrix for F. For a suitable $n \times n$-matrix C, the *preconditioner*, and $\tilde{x} \in x \in \mathbf{ID}$, we define the *Hansen-Sengupta operator*

$$H(x, \tilde{x}) := \tilde{x} + \Gamma(CA, -CF(\tilde{x}), x - \tilde{x}),$$

where $\Gamma(A, b, x) := y$ with

$$y_i = \{ (b_i - \sum_{k<i} A_{ik} y_k - \sum_{k>i} A_{ik} x_k) / A_{ii} \} \cap x_i,$$

which is called the *Gauß-Seidel operator* applied to A, b, and x.

Remark. If n is small, the recommended choice for C is $C \approx \check{A}^{-1}$ (Neumaier 1990).

Theorem 6 (Neumaier 1990). *Under the assumptions of Definition 5, if $\tilde{x} \in x \in \mathbf{ID}$, then $x' := H(x, \tilde{x})$ has the following properties:*

i) Every zero $x^* \in x$ of F satisfies $x^* \in x'$.
ii) If $x' = \emptyset$, then F contains no zero in x.

iii) If $\emptyset \neq x' \subseteq int(x)$, *then F contains a unique zero in x.*

Definition 5 and Theorem 6 show that the application of the Hansen-Sengupta operator to an interval x_0 leads to an interval x_1, which has at most the same radius as x_0 and no zero will be lost. Thus, it is natural to consider the iteration

$x^0 := x$

1) $x^{k+1} := H(x^k, \tilde{x}^k)$ for $k = 0, 1, 2, \ldots$

with $\tilde{x}^k \in x^k$. Here we put $x^{k+1} := \emptyset$ if $x^k = \emptyset$.
Due to the preceding theorem and the definition of H, we have

2) $x^{k+1} \subseteq x^k$ for all $k \geq 0$.

3) $x^* \in x, F(x^*) = 0 \Rightarrow x^* \in x^k$ for all $k \geq 0$.

The next theorem gives conditions under which the x^k contracts to a solution (if one exists).

Theorem 7 (Neumaier 1990). *Let A be a strongly regular Lipschitz matrix on* $x \in \mathbf{ID}_0$ *for* F: $\mathbf{D}_0 \subseteq \mathbf{R}^n \to \mathbf{R}^n$. *Let* $C \in \mathbf{R}^{n \times n}$ *be such that CA is an H-matrix. Then the Hansen-Sengupta iteration* (1) *is strongly convergent for* $\tilde{x}^k = \check{x}^k$, *i.e., either F has a unique zero* x^* *in x and* $\lim_{l \to \infty} x^l = x^*$ *or F has no zero in x and* $x^k = \emptyset$ *for some* $k \geq 0$.

In order to satisfy the conditions of this theorem, we put $\tilde{x}^k = \check{x}^k$ for all $k \geq 0$. As it is suggested from Definition 4, we take the Lipschitz matrix A to be an interval extension of the derivative $F'(x)$. If A is not strongly regular, the iteration breaks off with intervals $x^{k+1} = x^k$ for some $k \geq 0$. If this occurs, the interval x^k must be partitioned into two subintervals and the iteration must be performed with both intervals. This method – the covering method – is described in more detail in Sect. 5. The H-matrix property may be satisfied by

Proposition 8 (Neumaier 1990). *Every interval* $n \times n$*-matrix A satisfying* $\|I - A\|_u < 1$ *for some* $u > 0$ *is an H-matrix, where* $\|A\|_u$ *with* $u \in \mathbf{R}^n$, $u > 0$ *is the scaled maximum norm*

$$\|A\|_u := \max_{i=1 \ldots n} \left\{ \sum_{k=1}^{n} |A_{ik}| \, u_k / u_i \right\}.$$

This proposition shows that $\check{A}^{-1} A$ is an H-matrix if $\check{A}^{-1} A \approx I$, i.e., if A is thin enough. A comparison between Hansen-Sengupta iteration and *Newton* or *Krawzcyk iteration* shows that the first one strongly converges under the weakest assumptions (Neumaier 1990); thus we use the Hansen-Sengupta operator instead of the (simpler) Newton- or Krawzcyk operators.

4 Bicubic B-splines; basic facts

In this section, we define bicubic B-spline surfaces. For further details, we refer the interested reader to deBoor (1978) and Farin (1990).
We define B-spline surfaces as so-called *tensor product surfaces:*

Definition 9. The representation

$$S(u, v) := \sum_{i=0}^{n-m-1} \sum_{k=0}^{r-s-1} d_{i,k} \, M_{k,s}(v) N_{i,m}(u),$$

where the $M_{k,s}(v)$ are *(normalized) B-splines* of degree s over the *knot vector* $V = (v_0, \ldots, v_r)$ and the $N_{i,m}(u)$ are (normalized) B-splines of degree m over the knot vector $U = (u_0, \ldots, u_n)$, is called a *B-spline surface.* The coefficients $d_{i,k}$ are called *deBoor points;* in their natural ordering they form the vertices of the *deBoor net* of the surface.
B-spline surfaces satisfy the *convex hull property,* which can be used to enclose the range. By inserting new knots into the knot vectors U and V, any B-spline surface may be partitioned into a net of *Bézier surfaces* – called *patches.*
Computing enclosures for the range of such patches may be accomplished in several ways. One way is to exploit the convex hull property on the deBoor points for smaller and smaller parameter intervals u and v (Cohen et al. 1980; Rokne 1982; Toth 1985). Unfortunately, this method requires a recalculation of the deBoor points whenever the parameter intervals decrease. Alternatively, another method is to convert the patch into the power representation and to evaluate this polynomial in each step.
Because the bicubic case (i.e., $m = s = 3$) is a good compromise between accuracy and computation time (Neumaier 1988b), we restrict our further discussions to this case, assuming patches converted into the power form

$$P(u, v) := \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} \, u^i \, v^j,$$

where $a_{ij} \in \mathbf{R}$ and $u, v \in [0, 1]$, i.e., P is a bicubic polynomial with real coefficients defined on the grid $[0, 1] \times [0, 1]$.

5 Interval methods for ray tracing bicubic B-spline patches

In this section, we present the theoretical part of our method for ray tracing objects consisting of bicubic B-spline patches. For a given screen, we have to compute an enclosure for the intensity by calculating hulls for the set of the intersection points and to trace the rays to the light sources and (recursively) the reflected and refracted rays. In our discussion, we have to compute intersections of

1) A cluster of "primary" rays with patches.
2) A single ray with patches.
3) A cluster of "secondary" rays with patches.

The main difference is that in case of single rays we have to compute the nearest intersection point of a ray with all patches in finitely many steps, whereas in case of clusters of rays we have to compute those parts of the patches that possibly have an intersection point with at least one of the rays of the cluster. In the latter case, we need a method that (eventually) decreases the number of the relevant patches and improves the parameter intervals of the remaining patches.

The difference between cases (1) and (3) is that the problem for primary rays may be restricted to the two-dimensional case if the scene is projected into the image plane using a unique perspective projection determined by the eyepoint and the screen.

We continue the discussion of this problem later in the section, because the first step in solving the intersection problem is to compute enclosures for the range of patches efficiently. As mentioned in the previous section, B-spline patches may be enclosed using the convex-hull property. Particulary, this property leads to effective ray-tracing methods if successive subdivision of patches is performed for each single ray [a hardware implementation of the procedure is described in Pulleyblank and Kapenga (1987)]. In contrast to these methods, our method computes parameter intervals of patches depending on a given cluster of rays. Thus, the convex-hull property is not effectively applicable, because the recalculation of the deBoor points for arbitrary parameter intervals requires more computation time than the preconversion of the patches to their power representation together with the evaluation of these polynomials whenever a parameter has been changed. Of the two possible representations $P(u,v)=\Sigma b_i(v)u^i=\Sigma c_j(u)v^j$, we take the first one if v has smaller radius than u and the second one if u has smaller radius than v. The ranges of the derivatives $P_u(u,v)$ and $P_v(u,v)$ of $P(u,v)$ with respect to the first and the second variable are calculated in a similar way.

Note that these calculations must be performed with outward rounding in order to avoid faults due to rounding errors. For further details, we refer the interested reader to Enger (1990).

Enclosures for the range of the surface normal are calculated similarily. Because the normal of a parametric surface is given by the cross product of the two partial derivatives, we obtain such an enclosure using

$$N=\begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}=\begin{bmatrix} P_{y,u}\cdot P_{z,v}-P_{y,v}\cdot P_{z,u} \\ P_{z,u}\cdot P_{x,v}-P_{z,v}\cdot P_{x,u} \\ P_{x,u}\cdot P_{y,v}-P_{x,v}\cdot P_{y,u} \end{bmatrix},$$

where $P_{i,t}$ for $i\in\{x,y,z\}$ is an enclosure for the derivative of the i-th coordinate of $P(u,v)$ with respect to the parameter $t\in\{u,v\}$.

We now present methods for computing an enclosure for the set of the intersection points of the patch $P(u,v)$ with a *cluster of rays* $R(q)=O+qD$, where O is the origin, D is the direction, and q is a parameter interval for the rays (in finite precision arithmetic q may be bounded; hence, q is indeed an interval). In general, O and D are real interval vectors.

If a projection with respect to the eyepoint and the screen is performed, the x-y-coordinate intervals of R only depend on the x-y-coordinates of the related screen (i.e., $q_x=q_y=0$) and the z-coordinate interval of R is given by $q+e$, where e is a real constant. Hence, this is a two-dimensional problem, and the next result follows by applying centered forms to the patch $P(u,v)$.

Theorem 10 (Enger 1990). *Let $P(u,v)$ $(0\le u,v\le 1)$ be a patch projected into the image plane and let $R(q)$ $(q\in\mathbf{IR})$ be a cluster of primary rays. Then the parameter intervals defining the set of intersection points of the rays with the patch are computed by solving*

$$0\in P_i(u_0,v)-R_i+P_{i,u}(u,v)*(u-u_0)$$
$$0\in P_i(u,v_0)-R_i+P_{i,v}(u,v)*(v-v_0)$$

for $i\in\{x,y\}$ with $u_0\in u, v_0\in v$.

q is computed from $0\in P_z-R_z(q)$.

Remarks

1) The system is solved for all relevant patches.
2) The new parameter intervals will be intersected

with the old ones to increase the improvement of the parameter intervals of the patch. If the intersection leads to an empty interval, i.e., there is (certainly) no intersection point of one of the rays with the patch, the patch is discarded.

3) u_0, v_0 are chosen to be the boundaries of u and v. The system is also solved for the midpoints of u and v to enforce a separation of the patch if it possibly contains the silhouette, i.e., if the hull of the z-component of the normal contains zero.

4) Because the q-interval is a measure for the distance from the origin, we may discard some additional patches in a special case (Fig. 1): let $\{P_i\}_{i=1\ldots k}$ be the set of all relevant patches. Then we call a subset S of $\{P_i\}$ *q-connected* iff S is maximal with respect to the property that either S contains only one patch or for each P_n of S there exists at least one $P_m \neq P_n$ of S such that the q-intervals of these patches have an nonempty intersection. Now, assuming that S is a q-connceted set with a) there certainly exists an intersection point of a ray of the cluster with a patch of S, b) the patches of S do not contain the boundary of any surface, and c) the patches of S do not contain the silhouette, then all patches $\notin S$ that lie beyond all patches $\in S$ may be discarded. Additionally, a q-connceted set S is called *nearest*, iff for all patches $P \in S$, $P' \notin S$: $\bar{q}_P < \underline{q}_{P'}$, where $\underline{q}$, $\bar{q}$ is the minimal or respectively the maximal distance from the origin.

5) The "equations" that are of the form $0 \in a + b(x - x_0)$, a, b, $x \in \mathbf{IR}$, $x_0 \in x$ may be solved with only a few arithmetical operations if x_0 is a boundary of x (for the technical details see Enger (1990)).

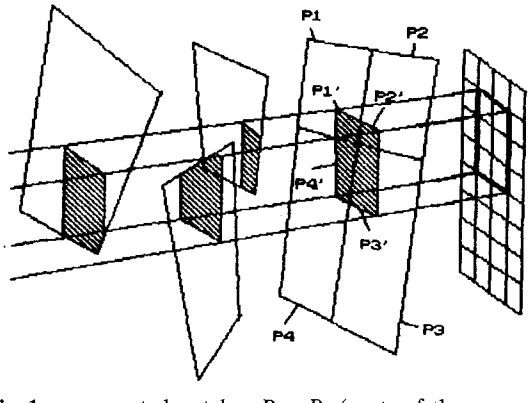


Fig. 1. q-connected patches $P_{1'} - P_{4'}$ (parts of the patches $P_1 - P_4$)

If the cluster includes a single ray only, the above method is not practicable, because Hansen-Sengupta iteration applied to all relevant patches P of the scene converges under the weakest conditions (see Theorem 7). Due to the results of Sect. 3, we take the Hansen-Sengupta operator $H(w, w_0) = w_0 + \Gamma(CA, -CP'(w_0), w - w_0)$ with

$$A = \begin{bmatrix} P_{x,u}(u, v) & P_{x,v}(u_0, v) \\ P_{y,u}(u, v_0) & P_{y,v}(u, v) \end{bmatrix},$$

$$P'(w_0) = \begin{bmatrix} R_x - P_x(u_0, v_0) \\ R_y - P_y(u_0, v_0) \end{bmatrix},$$

$u_0 = \check{u}$, $v_0 = \check{v}$, $w = (u, v)^T$, $w_0 = \check{w}$, $C = \check{A}^{-1}$.

Remarks

1) q is calculated as in Theorem 10.

2) Because a regular matrix A is strongly regular and $C = \check{A}^{-1}A$ is an H-matrix if A is "thin enough," a partitioning of the patches during the iteration process (especially in the neighborhood of the silhouette) is sometimes necessary to achieve (nearly) planar (sub-)patches, i.e., patches with narrow interval extensions for the derivative. If no further improvement is achieved and the accuracy requirement is not met, the patch is patitioned into two subpatches (by partitioning one of the parameter intervals) and both parts are placed onto a stack. The iteration must then be performed with all the parts on the stack until the stack is empty. This method is also called the *covering method* (Neumaier 1988a), because the solution set for the system of equations is covered by a collection of smaller and smaller boxes, which give increasingly fine information about the location of the solution set. If an intersection point is already found, all patches with a greater distance from the eyepoint are removed from the stack.

In case of secondary rays, an application of one of the previously described methods is not practicable, because the ray-parameter interval q leads both to wide matrix elements in the Hansen-Sengupta operator and to an additional pair of equations with wide interval factors in Theorem 10. On the other hand, solving the equations of Theorem 10 for planar patches leads to good enclosures for the parameter intervals, because all intervals occurring in the equations are narrow. Thus, partitioning all patches into (nearly) planar subpatches, approximating each subpatch by a quadrangle and

computing the intersection points of a cluster of rays with all relevant quadrangles according to a method similar to those described in Theorem 10, should lead to an effective method. Because the computation with triangles is easier, we divide each quadrangle into four triangles (the triangulation of a patch is described in the next section). Unfortunately, the approximation of nearly planar subpatches by triangles leads to a decreasing accuracy if the subdivision process breaks off too early and to an increasing number of triangles, and hence to increased computation time, if the accuracy requirements of the process are sharpened. Thus, the tolerances should be chosen as a compromise between accuracy, computation time, and storage cost (see Sect. 8).

At least, we present a method for computing the intersections of a cluster of (secondary) rays with triangles. Using a parametric representation of a triangle with vertices T_1, T_2, T_3 of the form $T(\beta, \delta) = T_1 + \beta \cdot (T_2 - T_1) + \delta \cdot (T_3 - T_1)$, $\beta, \delta \in [0, 1]$, $\beta + \delta \leq 1$, enclosures for the parameter intervals defining the set of intersection points may be calculated in a manner similar to Theorem 10.

Proposition 11. *Let $T(\beta, \delta)$ $(\beta, \delta \in \mathbf{IR})$ be a parametrically defined triangle and let $R(q) := O + q \cdot D$ $(O, D \in \mathbf{IR}^3, q \in \mathbf{IR})$ be a cluster of rays. Then the intervals (β, δ, q) defining the set of intersection points of R with T may be achieved by solving the system*

$$0 \in b + Aw,$$

where $w = (\beta, \delta, q - \check{q})^T$, $b \in \mathbf{IR}$, $A \in \mathbf{IR}^{3\times 3}$ with $b = T_1 - R(\check{q})$, $A = (T_2 - T_1, T_3 - T_1, R'(q))$.

Remarks

1) In order to decrease computation time, we compare the hull of the coordinates of the triangles with the coordinates of the cluster of rays, applying the above method only to those triangles that produce a nonempty intersection.

2) We may improve the above hull-comparison method if we represent each patch by a binary tree (see Fig. 2), where the root represents the patch and each of both sons of a knot is one of the subpatches of the "father" patch resulting from the subdivision process. The leaves of the tree represent the triangles. Computing the hull of the coodinates of each knot as the interval hull of the coordinate hulls of the sons, we may improve the above method as follows: starting with the root, we compare the coodinate hull of each knot with the coodinates of the cluster. If this leads to a nonempty intersection, we repeat the process with both sons until we reach a leaf, i.e., a triangle. Note that the tree and the coordinate hulls may be computed in a precalculation step. The trees of all patches of the scene could be combined into an unique tree. However, we have found no advantage in doing this.

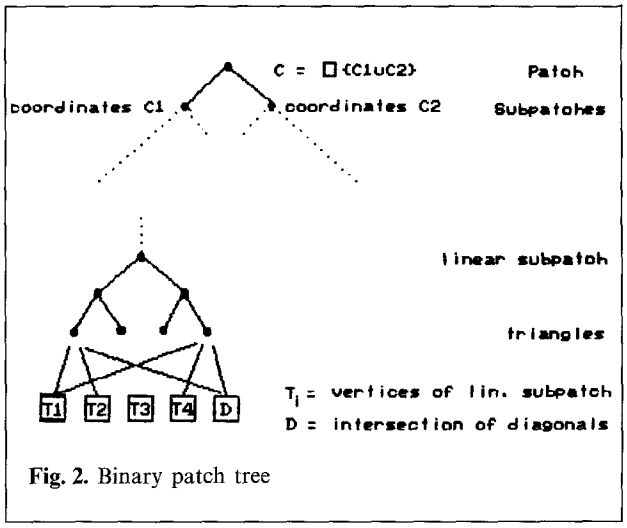


Fig. 2. Binary patch tree

3) A similar argument as for q-connected patches may reduce the number of relevant (q-connected) triangles.

Note that in the case of a single secondary ray, all vectors and the matrix A have real entries.

6 Triangulation

In this section, we give a brief discussion of the problem of triangulating a parametric surface. Because we use B-Spline surfaces represented as a collection (or a net) of patches, we have to partition any patch into subpatches until the subpatches are planar within some given tolerance. Each of these "planar" subpatches (viewed as a rectangle) is partitioned again into four triangles. Inasmuch as the splitting of the patches depends on the geometry of the surface, we have to take care of a proper implementation to avoid "cracks" in the surface. We illustrate this phenomenon in Fig. 3.

Let P_1 and P_2 be two subpatches such that P_1 is planar within the given tolerance, but P_2 has to

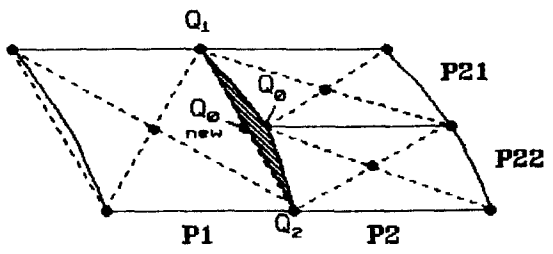


Fig. 3. The splitting of adjacent subpatches

be split once more giving the new subpatches P_{21} and P_{22}. The 12 triangles related to these subpatches are printed with dashed lines. Clearly, the shaded section is not contained in any of these triangles and would be lost during the tracing process. To avoid this, we have to chose the point Q_0 as the midpoint of the straight line joining the two points Q_1 and Q_2 and not as the midpoint of the (cubic) curve representing the common boundary curve of the patches P_1 and P_2. If one of the patches P_{21} or P_{22} is split once more to achieve the required planarity, we have to use in the further calculations the new point $Q_{0,\text{new}}$ instead of Q_0. A similar discussion of this problem is found in Clay and Moreton (1988).

Following Lane et al. (1980), we take advantage of the convex-hull property of the B-Spline curves and surfaces to obtain a criterion for planarity: we call a patch *planar* iff the Euclidean distance of the deBoor points to the plane of any three corner points is less than a tolerance ε_1 and for all four boundary curves the Euclidean distance of the interior deBoor points to the line segment joining the end points is less than a tolerance ε_2 (these two tolerances will be specified in Sect. 8).

7 Algorithms

The presentation of our algorithm *INTRACY* for ray tracing a scene consisting of several objects given as parametric B-spline surfaces is not straightforward, because we use different techniques for the different classes of rays. The strategy is as follows:

Starting with the whole screen, compute those parts of all patches that possibly have an intersection point with one of the rays. Compute the intensity interval of the actually considered screen. If the screen could not be uniquely colored, split the screen and place the parts onto a stack. In the other case, color the screen. Proceed with the next part from the stack until the stack is empty. The input parameters are the set of B-spline surfaces defining all objects of the scene and the coordinates of the eyepoint and of the screen.

The precalculation, which is independent from the subdivision process, includes projecting the scene into the image plane (determined by the eyepoint and the screen), computing the net of patches representing all surfaces, converting the deBoor points of all these patches into the power form, and computing their related trees.

Furthermore, a stack is required for storing the information about those parts of the screen not yet considered. This information is supplemented by a set of data structures (one structure for all those patches that possibly contain an intersection point), including the actual u- and v-interval and a pointer to the original deBoor points.

Algorithm ***INTRACY***

1) Precalculation.
2) Set B:=coordinates of the whole screen.
Initialize the stack. Mark all patches.
3) For all marked patches:
3.1 Compute the hull of the x-y-coordinates and compare them with B. Delete the mark of the patch and goto 3 if no solution exists.
3.2 Compute the new parameter intervals u and v according to Theorem 10 if B is a cluster, or perform Hansen-Sengupta iteration if B is thin. Delete the mark of the patch if no solution exists.
4) If no patch is marked, goto 11, else sort the patches in order according to their minimal distance from the eyepoint.
5) Improve B by intersecting it with the hull of the x-y-coordinates of all marked patches.
6) If B is thin, compute the coordinates and the normal at the nearest intersection point [see remark 1) below].
7) If B is not thin and a nearest set S of q-connected patches exists, compute the hulls of the coordinates and the normal of the patches $\in S$ as wide intervals and delete the marks of all patches $\notin S$. Additionally, the coodinates and the normals at the intersection points at the four "corners" of screen B are computed using Hansen-Sengupta iteration (see step 9 and Remark 2 below). If no nearest q-conneted set exists, goto 10.
8) Call ***SHADE*** (see below) to compute the intensity interval of (sub-)screen B.

9) If the intensity interval contains at most two (discrete) graphic intensities, use the bilinear form determined by the intensities at the corners of B to calculate the intensity for each pixel of B and goto 11 (see Remark 2).
10) Split B into two parts (see Remark 3 below) and place all parts onto the stack (together with the information about all marked patches).
11) If the stack is not empty, pop next B from the stack and goto 3.
12) End.

Remarks

1) Hansen-Sengupta iteration is performed until condition iii) of Theorem 6 is satisfied or the radii of the computed parameter intervals are less than a specified accuracy requirement. To obtain the nearest intersection point, Hansen-Sengupta iteration is repeated with all remaining patches until a unique nearest patch is found or all patches satisfy the accuracy requirement (in this case an arbitrary one is taken; however, this did not occur in our experience). The coordinates and the normal are computed at the midpoints of the parameter intervals of the remaining patches.
2) The additional computation of the intersections with the four "corner rays" (the rays determined by the corners of the screen) increases computation time, but has the advantage of approximating the intensities of all pixels by a *bilinear form* if the simultaneously calculated intensity interval for the whole cluster is thin enough. In order to obtain the intensities of the pixels (x, y) of screen B, we use the bilinear form

$$I(x, y)=a+bx'+cy'+dx'y' \quad \text{with}$$
$$x' := (x-x_{\min})/(x_{\max}-x_{\min}),$$
$$y' := (y-y_{\min})/(y_{\max}-y_{\min}),$$

where the min- and max-values are the minimal and maximal x-y-coordinates of B and the coefficients a–d are determined by the intensities at the corners of B. To avoid sharp transitions in intensity, we add as a dithering component a random number taken from the interval $[-0.5, 0.5]$ to each of these pixel intensities (assuming that the intensities of the graphic card are integers).
3) According to our experience, an improvement in computation time is achieved if the screen is split into four pieces instead of only two in step 10 (as long as it is possible).
4) A further improvement in computation time is obtained if the screen is prepartitioned into 8×8 congruent parts and the algorithm is applied to each of these parts (except the precalculation).
We now present the routine ***SHADE***, which computes recursively the intensity interval for a (sub-) screen B.
Hulls of the coordinates and of the normals at the set of intersection points and the coordinates and the normals at the four "corner intersection points" (if it is not an unique intersection point) are input together with the related set of marked patches, respectively triangles. Output are an intensity interval I and the intensities I_1-I_4 at the four "corners."
1) Set $I := [0, 0]$, and $I_{1-4} := 0$. Mark all triangles that are subsurfaces of the marked patches.
2) For all light sources
2.1 Compute the cluster of (secondary) rays from the light source to the hull of the intersection points.
2.2 For all nonmarked triangles, compute the parameter intervals β and δ according to Proposition 11 or directly in the case of a single ray.
2.3 If no solution exists for any triangle, calculate the intensity interval with respect to a given illumination model and add it to I. Compute also the intensities at the four "corners" and add them to I_{1-4}. Goto 2.
2.4 If the ray is a single ray or if the triangles form a nearest q-connected set, goto 2.
2.5 Set $I := [-\infty, +\infty]$ and return to the calling routine.
3) If the intersection object is a reflecting medium,
3.1 Compute the cluster of reflected rays and the four reflected "corner" rays.
3.2 For all nonmarked triangles, compute the parameter intervals β and δ according to Proposition 11 or directly in case of a single ray. Denote by T the set of all triangles leading to a solution.
3.3 If T is empty, goto 4.
3.4 If the ray is a single ray or if the triangles of T form a nearest q-connected set,
3.4.1 Delete the marks of all triangles (but keep them in mind for step 4).
3.4.2 Mark all triangles of T.
3.4.3 Compute the hull of the coordinates and the normal interval of all triangles of T.
3.4.4 Compute the intersection points and the normals for the four "corner" reflected rays.
3.4.5 Call ***SHADE***.
3.4.6 If the intensity interval calculated by

SHADE is not finite, set $I:=[-\infty, +\infty]$ and return to the calling routine.

3.4.7 Add the calculated intensity interval and the intensities at the four "corners" to I and I_{1-4} (with respect to the illumination model). Goto 4.

3.5 Set $I:=[-\infty, +\infty]$ and return to the calling routine.

4) If the intersection object is a refracting medium, execute steps 3.1–3.5 with the cluster of the refracted rays.

5) Return to the calling routine.

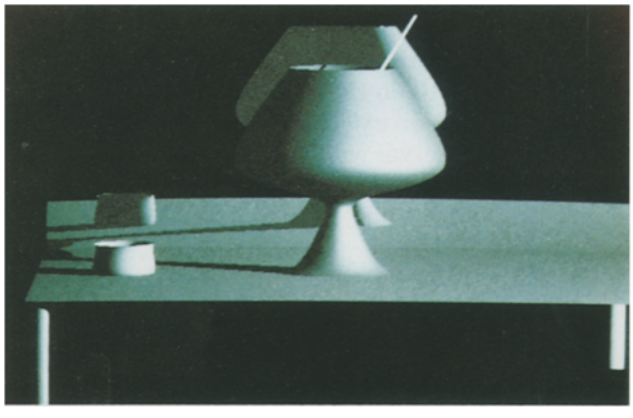
Remarks

1) The recursive process defined by step 3.4.5 (respectively 4.4.5) is executed until the maximum level of recursion has been reached.

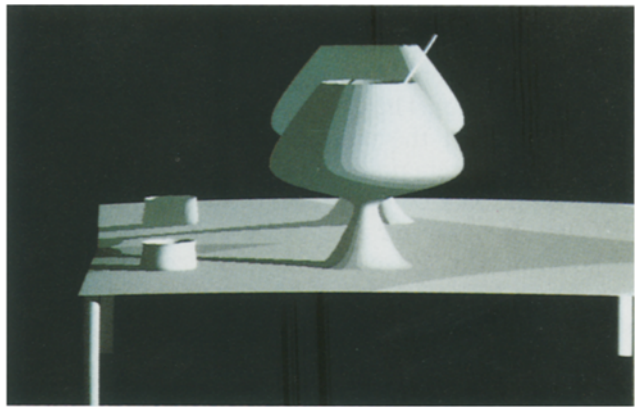
2) An alternative method avoiding approximation errors is described in Woodward (1989). It directly subdivides the patches in the viewing plane. Unfortunately, the use of integer arithmetic to speed up the process seems to be crucial in this method, which is therefore not applicable to our real arithmetic algorithm.

8 Test results

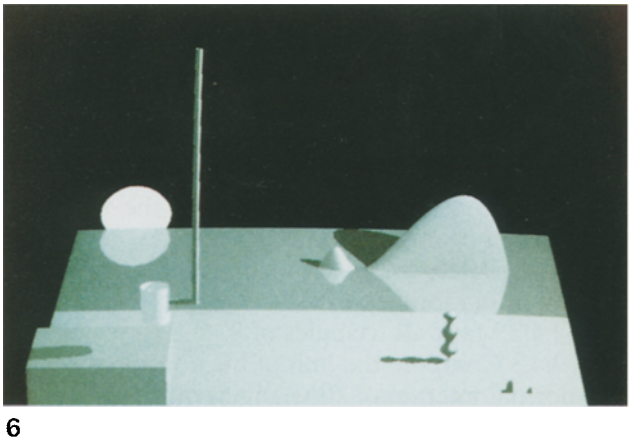
Experiments have been performed by implementing the algorithms in TURBO-PASCAL 4.0 on a PC/XT with a V20 processor without arithmetic coprocessor for three different scenes (SCENE 1–3) and for several resolutions. Following Schmitt et al. (1988), the maximum level of recursion has been set to 2.



4



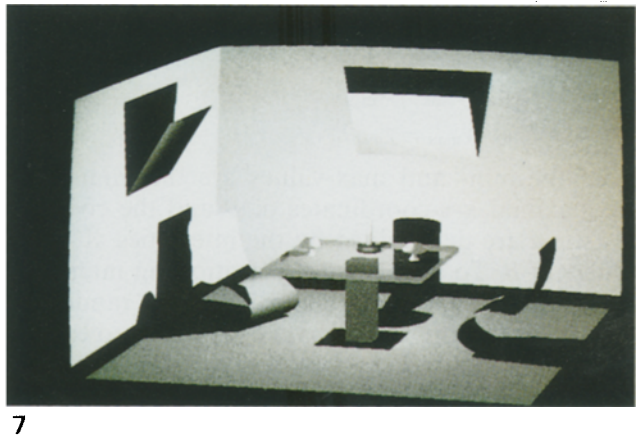
5



6

Fig. 4. SCENE 1

Fig. 5. SCENE 1 without dithering



7

Fig. 6. SCENE 2

Fig. 7. SCENE 3

Table 1. Number of patches, triangles, and computation time required for the precalculation

	No. objects	No. patches	No. triangles	Time min:s
SCENE 1	9	104	4×1106	6:51
SCENE 2	15	103	4×1050	6:35
SCENE 3	21	113	4×846	5:34

Figures 4, 6, and 7 show the scenes (with a resolution of 640 × 480 pixels and 16 different intensities), while Fig. 5 shows SCENE 1 without dithering; a comparison with Fig. 4 shows that graphics are much more realistic with dithering. SCENE 1 represents a pan and a glass on a table in front of a mirror, SCENE 2 represents a scene with two light sources, and SCENE 3 shows a room with a candle on a glass-plate.

Table 1 gives the computation times for the initial calculation (i.e., step 1 of algorithm INTRACY). As a compromise between accuracy and storage cost, we have obtained the best results in the triangulation process with the tolerances

$$\varepsilon_1 = 0.001 \cdot d_{\max}, \varepsilon_2 = 0.02 \cdot d_{\max},$$

where $d_{\max}$ is a boundary on the maximal diameter of the coordinates of the scene. This choice leads to an average number of 38 triangles for each patch.

To show the influence of dithering on computation time, we have compared the computation time for ray tracing SCENE 1 with dithering (17:10 h) and without dithering (19:20 h). This result seems to be surprising, because dithering requires additionally computing the intensities at the corners of the screen and a time penalty should be expected. The reason for a faster execution time with dithering lies in the fact that in this case we are able to finish the recursive splitting of the screen if a computed intensity interval still contains two different representable intensities, while in the other case we have to split until the intensity interval contains exactly one representable intensity. This more than compensates the extra time needed to compute the intensities at the four corners.

Table 2. Execution times INTRACY–TRAY (resolution of 640 × 480 pixels, 16 different intensities)

	Execution time (h:min)		Speed-up factor
	INTRACY	TRAY	
SCENE 1	17:10	42:08	2.5
SCENE 2	17:13	45:12	2.6
SCENE 2a	10:59	33:58	3.1
SCENE 3	19:30	49:51	2.6

Table 2 compares the execution time needed for our ray-tracing technique, called *INTRACY*, with a "conventional" technique called *TRAY*. This "conventional" technique shades the screen pixel by pixel by computing separately the intersection points of a single ray with triangles, supported by the hull comparison method and the patch trees [see Remark 2) after Proposition 11]. Because this is the same kind of method as used for the secondary rays in INTRACY, the time ratios are expected to be independent from details of the implementation. Due to lack of proper implementations, further interesting comparisons between the author's method and other "conventional" methods [e.g., the method of accelerated ray tracing presented in Fujimoto and Iwata (1986)] are not presented here.

In order to obtain an indication of the dependence on the number of light sources, we have considered SCENE 2 with one of the two light sources switched off (SCENE 2a).

For this resolution and number of intensities, we achieve an improvement with a factor between 2.5 and 3.0 over the conventional technique.

We now check the dependencies of the two techniques on the resolution and the number of intensities. For INTRACY, we expect a logarithmic dependency on the resolution, while TRAY should have a linear dependency. The influence of the number of intensities cannot be predicted a priori, because it strongly depends on the structure of the scene. Only TRAY should be independent of this number. We illustrate this for SCENE 1 in Figs. 8 and 9. Extrapolating these results, we would expect an improvement factor for the standard resolution and number of intensities for workstations (1280 × 1024 pixels, 256 intensities) of about 3.1.

In order to represent colored images, the calculation of the intensity has to be performed simultaneously for the three colors red, green, and blue (assuming a RGB-model is used). In this case, we expect only a small increase in computation time, because the screen is already partitioned into small parts at the "critical points" (boundary of surfaces, areas with a strong curvature with great change of intensity). Hence the representation of more than one color is expected to lead to a few further splittings of the screen only. Therefore, the execution

time for representing colored images on a graphic card with 24-bit memory/pixel will be nearly the same as for black/white images with 256 different intensities.

A further interesting analysis is the consideration of the sizes of the subscreens, which are known to display a unique color, i.e., which do not need any further subdivision. We illustrate the frequencies of the sizes for SCENE 3 in Table 3 (the distribution of the sizes for the other scenes are similar). Sizes are given in number of pixels (Nops), where the maximum attainable size here is 4560 Nops (with a sub-splitting of the total screen with 608×480 pixels into 64 congruent parts). Table 3 illustrates that equally colored subscreens are still known in a relatively early stage, which is the power of the presented method. Additionally, it can be seen that only about 1/9 of the total number of pixels requires the full subdivision process. Nevertheless, much computation time is required to calculate the intersection points for these pixels, because these are generally the "critical points." According to our experience, Hansen-Sengupta iteration seems to be the fastest method (for SCENE 3 an average number of 2.1 (sub-)patches per single pixel have to be considered, whereby an average number of only 1.6 iteration steps per patch have to be executed).

Our algorithm can reduce *aliasing* to an acceptable level with little expense. In the case of shading more than one pixel at a time, there is no need for antialiasing, because the methods of interval analysis yield a certain enclosure for the range of the intensity, while shading a single pixel with respect to antialiasing can be performed using a method of *supersampling*, where the pixel is partitioned into a matrix of typically 4×4 subpixels. Inasmuch as the surface parameters are stored for each pixel and the information about the 2×1-pixel (respectively 3×1-pixel) screen of the preceding step can be used, it is easy to supersample the patch in a local area.

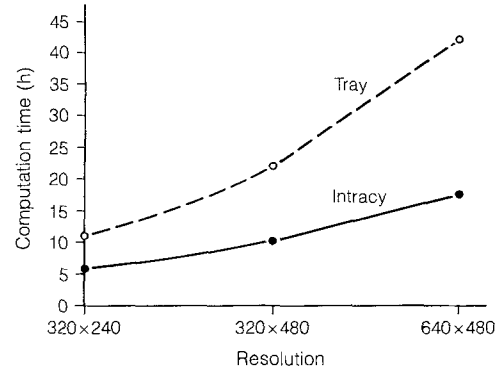


Fig. 8. Computation time as a function of the resolution with 16 intensities (SCENE 1)

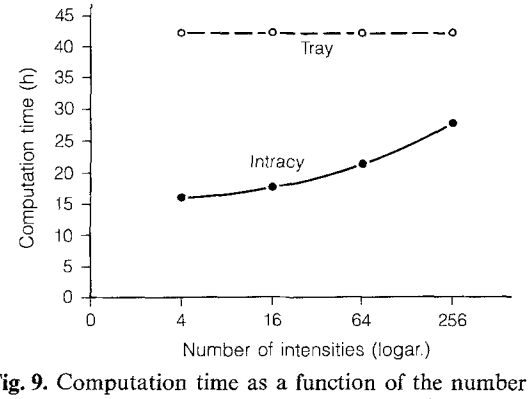


Fig. 9. Computation time as a function of the number of intensities with a resolution of 640×480 (SCENE 1)

9 Conclusions

In this paper, we have developed a technique that applies methods of interval analysis to the problem of intersecting a cluster of rays with a parametrically defined surface. Our algorithm provides a speed-up factor between 1.5 and 3.0 over a conventional algorithm.

An experimental analysis has shown that much of the CPU time is used to evaluate quadratic and cubic polynomials. Thus, a hardware implementation for evaluating a spline with Horner's scheme should lead to a further significant speed-up of the algorithm.

Because the main tools of interval analysis used in our method are independent of the representation of the surface, the technique is also applicable to other surfaces, e.g., algebraic surfaces defined by an *implicit function* $F(x, y, z) = 0$. A polygonization of implicit surfaces (similar to that used in our algorithm) can be found in Bloomenthal (1988).

Furthermore, because of the divide-and-conquer strategy of our algorithm, it extends trivially to animated graphics by working in spacetime. If the space coordinates are extended by a further param-

Table 3. Sizes of equally-colored subscreens of SCENE 3

Nops	Frequency	Nops	Frequency	Nops	Frequency
1	30982	54	2	240	14
2	4222	55	1	252	3
3	627	56	37	260	14
4	2116	60	90	261	1
5	64	63	75	266	15
6	1884	64	7	270	1
7	32	70	427	280	65
8	306	72	16	285	4
9	184	78	1	300	19
10	80	80	64	320	1
11	1	84	2	343	1
12	305	90	3	344	1
14	21	100	11	380	1
15	1055	112	13	420	2
16	128	114	2	456	2
18	3	116	1	460	2
20	897	117	1	532	2
21	2	120	10	551	1
22	2	126	26	560	3
23	1	130	3	580	2
24	3	135	6	600	1
25	63	140	146	627	1
28	27	145	1	715	1
29	1	150	29	720	2
30	8	160	3	788	1
32	5	168	1	800	1
33	2	170	1	1120	8
35	330	171	1	1160	8
40	40	200	5	1520	2
42	3	215	1	2080	1
45	2	220	7	2490	1
50	35	238	4	2850	1
51	1				

eter t, *time*, presenting an interval with several discrete values, e.g., 25 values per second for the TV norm, the dimension increases by 1 with no further changes to the algorithm. This should lead to an effective technique for rapidly finding the (probably relatively large) sections of constant intensity. The produced spacetime boxes, i.e., equally shaded subscreens related to a time interval, should be stored until the procedure is finished and the pictures at each point of time should then be build up from these boxes. A method for creating efficient bounding-volume hierarchies supporting our technique can be found in Glassner (1988).

An interesting future work could be the research on a combination between the author's method and the method of accelerated ray tracing.

Acknowledgements. The author wishes to thank Professors Dr. A. Neumaier and Dr. H. Müller for their helpful discussions during the development of this method and for their suggestions, which improved the content and the presentation considerably.

References

Alefeld G, Herzberger J (1983) Introduction to interval computation. Academic Press, New York

Baumann E (1988) Optimal centered forms. BIT 28:80–87

Bloomenthal J (1988) Polygonization of implicit surfaces. Comput Aided Geometric Design 5:341–355

deBoor C (1978) A practical guide to splines. Springer, Berlin Heidelberg New York

Clay RD, Moreton HP (1988) Efficient adaptive subdivision of Bézier surfaces. In: Proc Eurographics. North-Holland, Amsterdam, pp 357–377

Cohen E, Lyche T, Riesenfeld R (1980) Discrete B-splines and subdivision techniques in Computer Aided Geometric Design and Computer Graphics. Comput Graph Image Proc 14:87–111

Dussel R, Schmitt B (1970) Die Berechnung von Schranken für den Wertebereich eines Polynoms in einem Intervall. Computing 6:35–60
Enger W (1990) Intervall Ray Tracing – ein Divide-and-Conquer Verfahren für photorealistische Computergrafik. Thesis, Institut für Angewandte Mathematik, Universität Freiburg
Farin GE (1990) Curves and surfaces in computer aided geometric design, (2nd edn). Academic Press, San Diego
Faux ID, Pratt MJU (1979) Computational geometry for design and manufacture. Ellis Horwood, Chichester
Fujimoto A, Iwata K (1986) ARTS: accelerated ray-tracing systems. IEEE Comput Graph Appl 16–27
Glassner AS (1984) Space subdivision for fast ray tracing. IEEE Comput Graph Appl 15–22
Glassner AS (1988) Spacetime ray tracing for animation. IEEE Comput Graph Appl 60–70
Glassner AS (1989) An introduction to ray tracing. Academic Press, San Diego
Koparkar PA, Mudur SA (1985) Subdivision techniques for processing geometric objects. In: Earnshaw RA (ed) Fundamental algorithms for computer graphics. Springer, New York Berlin Heidelberg, pp 751–801
Krawczyk R (1969) Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. Computing 4:187–201
Lane JM, Carpenter LC, Whitted T, Blinn JF (1980) Scan line methods for displaying parametrically defined surfaces. Commun ACM 23:23–34
Moore RE (1966) Interval analysis. Prentice-Hall, Englewood Cliffs
Mudur SA, Koparkar PA (1984) Interval methods for processing geometric objects. IEEE Comput Graph Appl 7–17
Müller H (1988) Realistische Computergrafik. Informatik-Fachberichte 163, Springer, Berlin Heidelberg New York
Neumaier A (1988a) The enclosure of solutions of parameter dependent systems of equations. In: Moore RE (ed) Reliability in computing – the role of interval methods in scientific computing. Academic Press, San Diego, pp 269–286
Neumaier A (1988b) Einführung in die Numerische Mathematik. Vorlesungsskript, Teil 3, Institut für Angewandte Mathematik, Universität Freiburg
Neumaier A (1990) Interval methods for systems of equations. Cambridge University Press, Cambridge
Nishita T, Sederberg T, Kakimoto M (1990) Ray tracing trimmed rational surface patches. Comput Graph 24:337–345
Pulleyblank R, Kapenga J (1987) The feasibility of a VLSI chip for ray tracing bicubic patches. IEEE Comput Graph Appl 7:33–44
Rokne J (1982) Optimal computation of the Bernstein algorithm for the bound of an interval polynomial. Computing 28:239–246
Schmitt A, Müller H, Leister W (1988) Ray tracing algorithms – theory and practice. In: Earnshaw RA (ed) Theoretical foundations of computer graphics and CAD. Springer, NATO ASI Series F 40:997–1029
Thirion JP (1990) Utilisation de la cohérence de rayons lumineux pour le lancer de rayons. Thesis, Université de Paris-Sud
Toth DL (1985) On ray tracing parametric surfaces. SIGGRAPH 19:171–179
Whitted T (1980) An improved illumination model for shaded display. Commun ACM 23:343–349
Woodward C (1989) Ray tracing parametric surfaces by subdivision in viewing plane. In: Strasser W, Seidel HP (eds) Theory and practice of geometric modelling. Springer, Berlin Heidelberg New York, pp 273–290
Yamaguchi K, Kunii TL, Fujimura K, Toriya H (1984) Octree-related data structures and algorithms. IEEE Comput Graph Appl 4:53–59

W. ENGER received his MSc in mathematics in 1986 and PhD in mathematics in 1991 at the University of Freiburg. He also tought courses in computer science at the Berufsakademie Lörrach (Germany). Currently, he is working as a software engineer at ip Info Process GmbH in Buchenbach (Germany). Enger is a specialist in interval analysis. His current research interests include numerical mathematics, computer graphics, and geometric modelling techniques.