Ray Tracing Trimmed Rational Surface Patches

Tomoyuki Nishita*, Thomas W. Sederberg† and Masanori Kakimoto‡
*Fukuyama University
†Brigham Young University
‡Fujitsu Laboratories LTD

Abstract

This paper presents a new algorithm for computing the points at which a ray intersects a rational Bézier surface patch, and also an algorithm for determining if an intersection point lies within a region trimmed by piecewise Bézier curves. Both algorithms are based on a recent innovation known as *Bézier clipping, described herein. The intersection* algorithm is faster than previous methods for which published performance data allow reliable comparison. It robustly finds all intersections without requiring special preprocessing.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

General Terms: Algorithms

Additional Key Words and Phrases: Computer graphics, ray tracing, visible surface algorithms, parametric surfaces.

1 INTRODUCTION

The history, theory and capabilities of ray tracing are well documented [3], [5]. This paper deals with the ray tracing of parametric surface patches. Specifically, we present an algorithm for computing all points at which a ray intersects a rational Bézier surface patch of any degree. We also describe an algorithm for determining if a point lies within a trimmed region of the patch. We define a trimmed region to be an area bounded by piecewise (possibly rational) Bézier curves in the parameter plane of the patch.

1.1 Ray-Patch Algorithms

Solutions to the ray/patch intersection problem can be categorized roughly as being based on subdivision, algebraic or numerical techniques.

Subdivision approaches are described by Whitted [19], Rogers [10] and Woodward [20]. These algorithms harness the convex hull property of Bézier surfaces: if the ray does not intersect the convex hull of the control points, it does not intersect the patch. Through recursively subdividing the patch and checking convex hulls, the intersection points can be computed at a linear convergence rate, amounting to a binary search. Whitted's algorithm operates in three dimensions, whereas Rogers and Woodward map the problem to two dimensions.

Numerical solutions to the ray/patch intersection problem include those developed by Toth [18], Sweeney and Bartels [17], and Joy and Bhetanabhotla [4]. Toth's algorithm is based on interval Newton iteration. It works robustly on any parametric surface for which bounds on the surface and its first derivatives can be obtained. *Sweeney and Bartels ray trace B-spline surfaces by refining the con*trol mesh using the Oslo algorithm until the mesh closely approximates the surface. The ray intersection is then computed by intersecting the control mesh with the ray, and using that intersection point as a starting point for Newton iteration. Joy and Bhetanabhotla's algorithm uses quasi-Newton optimization to compute the point(s) on the patch nearest the ray, including intersection points.

Kajiya[6] devised an intersection algorithm based on **algebraic** techniques (ie., *resultants*). Kajiya's algorithm reduces the problem of intersecting a bicubic patch with a ray into one of finding the real roots of a degree 18 univariate polynomial.

Our ray/patch intersection algorithm is based on the convex hull property of Bézier curves and surfaces using a technique we refer to as *Bézier clipping*. Traditionally, intersection algorithms (such as curve/curve [13], surface/surface [7], or ray/surface [19]) based on the convex hull property (that is, subdivision based algorithms) perform a linearly converging binary search. Bézier clipping uses the convex hull property in a more powerful manner, by determining parameter ranges which are guaranteed to not include points of intersection. Variations of this concept have proven profitable in algorithms for algebraic curve intersection[12] and planar parametric curve intersection[15]. Bézier clipping has the flavor of a geometrically based interval Newton method, and thus might be categorized as partly a subdivision based algorithm and partly a numerical method.

1.2 Trimmed Patch Algorithms

Previous approaches to the rendering of trimmed patches include adaptive forward differencing [16] and polygonization [9]. Neither of these approaches adapts directly to ray tracing (unless one were to polygonize the patch[9] and then ray trace the polygons). If trimming is caused by a boolean operation involving solid geometric models, rendering can be performed using conventional constructive solid geometry methods[11]. Our algorithm renders trimmed patches defined in a boundary representation, by determining if a point on the patch lies inside or outside a trimmed region. One contribution of this paper is a fast, robust algorithm (based on Bézier clipping) for determining if a ray intersects a collection of trimming curves an even or odd number of times.

1.3 Paper Overview

Section 2 introduces Bézier clipping and applies it to the problem of point classification for trimmed patches. Section 3 describes our ray/patch intersection algorithm, and performance comparisons are presented in section 4.

This paper assumes the reader is familiar with rational Bézier curves and surfaces[2].

The ray/patch intersection algorithm requires patches to be expressed in Bézier form. The point classification algorithm requires trimming curves to be in Bézier form. Conversion from NURBS to Bézier representation is discussed in reference 9.

2 POINT CLASSIFICATION

Figure 1 shows a *trimmed* Bézier surface patch. Its parameter domain (see Figure 2) contains several *trimming curves*, expressed in Bézier (or rational Bézier) form. Regions enclosed by trimming curves are excluded from the patch.

Point classification, in the context of trimmed patch ray tracing, is the problem of determining if a given point **S** in s, t patch parameter space lies IN the patch, OUT of the patch (in a region enclosed by trimming curves), or ON a trimming curve. If **S** is a point at which a ray intersects the patch, then **S** qualifies as a hit if it is IN, and as a miss if it is OUT. If **S** is ON a trimming curve, it is reported as a hit, but is flagged for possible anti-alias supersampling.

Trimming curves completely enclose an OUT region. This may require linear Bézier curve segments along portions of the patch boundary (as in Figure 2).

Recall a corollary of the Jordan curve theorem: If any ray **R** in s, t patch parameter space (not to be confused with a "tracing ray" in R^3) emanating from **S** intersects the collection of trimming curves an even (odd) number of times, then **S** is IN (OUT of) the patch. Our point classification algorithm amounts to an efficient method of determining that even/odd intersection parity. For this discussion, **R** points in the positive s direction. In practice, choose the ray in the $\pm s$ or $\pm t$ direction which exits the parameter square in the shortest distance. It might appear that ray/curve tangencies can cause a problem. As discussed in **CASE** $\mathcal{B}$ below, this is not a concern.

The algorithm begins by splitting the patch parameter plane into quadrants which meet at **S** as shown in Figure 3. To determine if **R** intersects a given Bézier trimming curve an even or odd number of times, we categorize the curve based on which quadrants its control points occupy. For now, assume that no end control point lies on a quadrant boundary (a situation addressed in section 2.2.1).

CASE $\mathcal{A}$: All control points lie on the same side of the line containing **R** (in quadrants I, II, III, IV, I&II, or III&IV) or "behind" **R** (in quadrants II&III). The convex hull property of Bézier curves guarantees zero intersections with **R**.

CASE $\mathcal{B}$: All control points lie in quadrants I&IV, but not case $\mathcal{A}$. Since the curve is continuous and obeys the convex hull property, if the curve endpoints lie in the same quadrant, the curve crosses **R** an even number of times. Otherwise, the curve intersects **R** an odd number of times. Note that tangencies between the ray and trimming-curve tangencies, even those of high order, do not pose a problem. The only question is whether the curve endpoints straddle the ray.

CASE $\mathcal{C}$: All other curves.

If a curve is case $\mathcal{A}$ or $\mathcal{B}$ no further processing is needed to determine its intersection parity with a ray. For a case $\mathcal{C}$ curve, we subdivide it using the de Casteljau algorithm[2] into three Bézier segments in such a way that the two end segments are guaranteed *a priori* to be case $\mathcal{A}$ or $\mathcal{B}$. The points at which to subdivide are determined using a technique we call *Bézier clipping*, described next. Discussion of the point classification algorithm resumes in Section 2.2.

2.1 Bézier Clipping

Figure 4 shows a Bézier curve **C** and the line **L** in s, t patch parameter space **C** is defined by its parametric equation

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{C}_i B_i^n(u). \qquad (1)$$

$\mathbf{C}_i = (s_i, t_i)$ are the Bézier control points and $B_i^n(u) = \binom{n}{i}(1-u)^{n-i}u^i$ denote the Bernstein basis functions. **L** is defined by its normalized implicit equation

$$as + bt + c = 0, \quad a^2 + b^2 = 1. \qquad (2)$$

The intersection of **L** and **C** can be found by substituting equation 1 into equation 2:

$$d(u) = \sum_{i=0}^{n} d_i B_i^n(u) = 0, \quad d_i = as_i + bt_i + c. \qquad (3)$$

Note that $d(u) = 0$ for all values of u at which **C** intersects **L**. Also, d_i is the distance from $\mathbf{C}_i$ to **L** (as shown in Figure 4).

The function $d(u)$ in equation 3 is a polynomial in Bernstein form, and can be represented as an "explicit" (or so-called "non-parametric") Bézier curve[1] as follows:

$$\mathbf{D}(u) = (u, d(u)) = \sum_{i=0}^{n} \mathbf{D}_i B_i^n(u). \qquad (4)$$

The Bézier control points $\mathbf{D}_i = (u_i, d_i)$ are evenly spaced in u ($u_i = \frac{i}{n}$). Since $\sum_{i=0}^{n} \frac{i}{n} B_i^n(u) \equiv u[(1-u)+u]^n \equiv u$, the horizontal coordinate of any point $\mathbf{D}(u)$ is in fact equal to the parameter value u. Figure 5 shows the curve $\mathbf{D}(u)$ which corresponds to the intersection in Figure 4.

Since $\mathbf{D}(u)$ crosses the u-axis at the same u values at which $\mathbf{C}(u)$ intersects **L**, we can apply the convex hull property of Bézier curves to identify ranges of u for which **C** does *not* intersect **L**. Referring again to Figure 5, the convex hull of the $\mathbf{D}_i$ intersects the u axis at points $u = u_{min} = \frac{1}{6}$ and $u = u_{max} = \frac{4}{7}$. Since $\mathbf{D}(u)$ lies inside the convex hull of its control points, we conclude that **C** does not intersect **L** in the parameter ranges $0 \le u < u_{min}$ or $u_{max} < u \le 1$. Bézier clipping is completed by subdividing **C** into three segments using the de Casteljau algorithm[2]. Segment 1 is defined over $0 \le u \le u_{min}$, segment 2 over $u_{min} \le u \le u_{max}$, and segment 3 over $u_{max} \le u \le 1$.

2.1.1 Rational Curves

For rational Bézier trimming curves

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} w_i \mathbf{C}_i B_i^n(u)}{\sum_{i=0}^{n} w_i B_i^n(u)} \qquad (5)$$

with control point coordinates $\mathbf{C}_i = (s_i, t_i)$ and weights w_i, the values of d_i must be modified as follows. Substituting equation 5 into equation 2 and clearing the denominator yields:

$$d(u) = \sum_{i=0}^{n} d_i B_i^n(u) = 0, \quad d_i = w_i(as_i + bt_i + c).$$

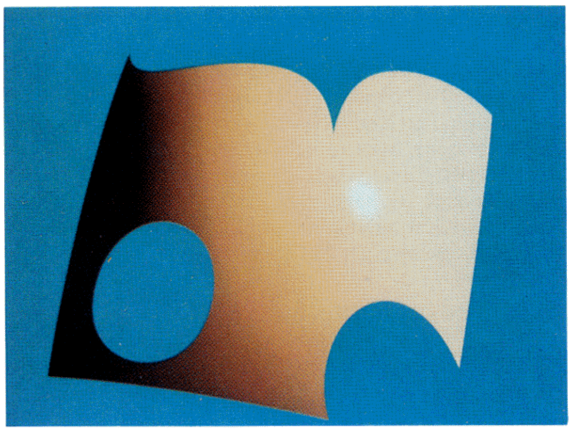
Figure 1: Trimmed Patch

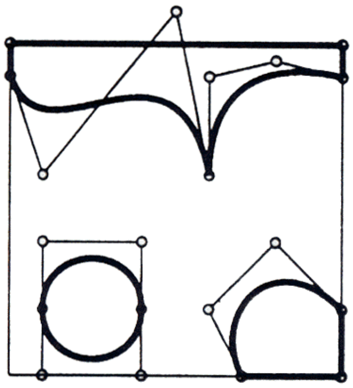
Figure 2: Trimming Curves

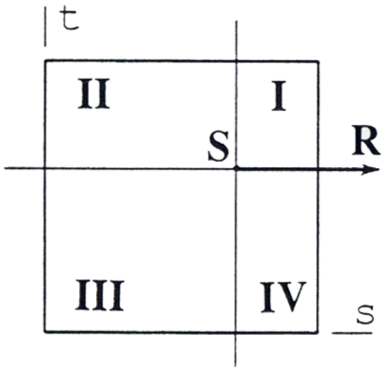

Figure 3: Quadrants

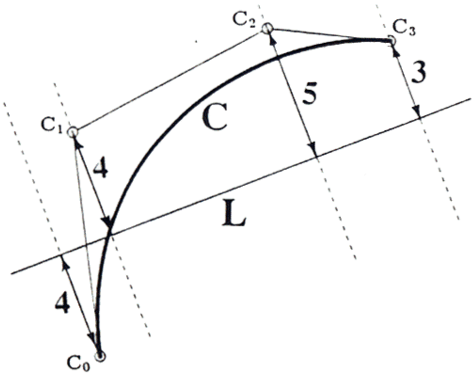

Figure 4: Bézier curve/line intersection

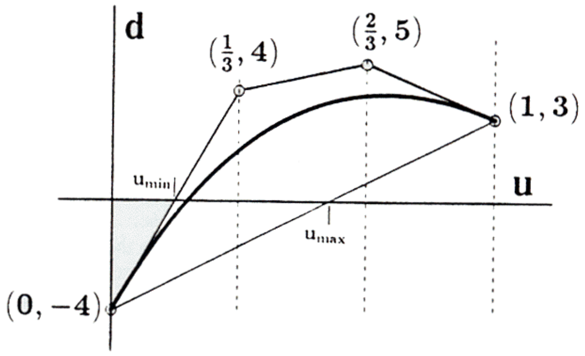

Figure 5: Explicit Bézier curve

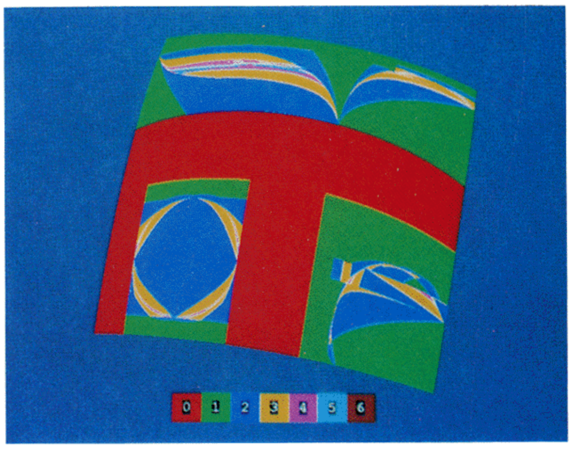

Figure 6: Bézier clips per classification

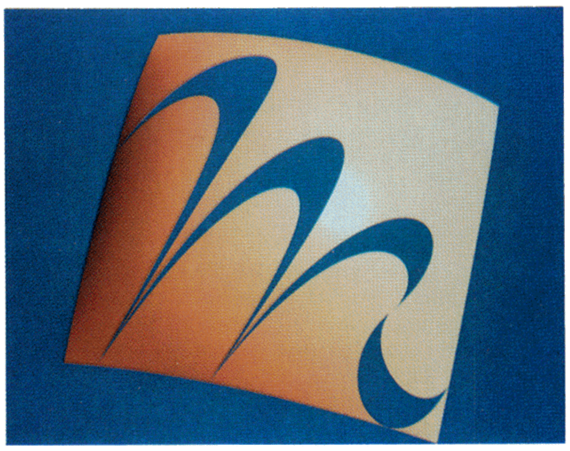
Figure 7: Sample trimmed patch

2.2 Point Classification Algorithm

Returning to the problem of point classification, our goal is to subdivide a case $\mathcal{C}$ curve into three segments, such that segments 1 and 3 are assured to be case $\mathcal{A}$ or $\mathcal{B}$. This can be accomplished by applying Bézier clipping against either the s quadrant axis ($\mathbf{L} = t - t_r = 0$) or the t quadrant axis ($\mathbf{L} = s - s_r = 0$) where the ray anchor $\mathbf{S} = (s_r, t_r)$. If a case $\mathcal{C}$ curve is Bézier clipped against the s quadrant axis, the resulting curve segments 1 and 3 must be case $\mathcal{A}$ and segment 2 could be any case. If a case $\mathcal{C}$ curve is Bézier clipped against the t quadrant axis, the resulting curve segments 1 and 3 must be case $\mathcal{A}$ or $\mathcal{B}$ and segment 2 could be any case.

We should clip against the axis which will result in the smallest segment 2. A good heuristic for this is to measure the distance from the curve endpoints to each of the axes. Generally, the larger the distance from an axis, the larger the clip tends to be. Denote $d_s = |d_0| + |d_n|$ for the case when $\mathbf{L}$ is the s quadrant axis, and $d_t = |d_0| + |d_n|$ when $\mathbf{L}$ is the t quadrant axis. Thus, if $d_s > d_t$, it is usually best to clip against $s - s_r = 0$.

The complete point classification algorithm appears as follows. For clarity, we assume that the point is nearest the edge $s = 1$ in the parameter square, so we count intersections of the trimming curves with the ray in the positive s direction. If the distance from the point to one of the trimming curves is less than a tolerance value ON_TOL, the point is declared to be ON a trimming curve. ON_TOL= 10^{-4} is conservative, and was used in producing Figure 7.

```
BEGIN CLASSIFY
INPUT: Trimming curves, Point, ON_TOL
OUTPUT: IN, OUT, or ON
$inter = 0$;
Push all trimming curves onto a stack;
WHILE Stack not empty
    POP a curve;
    SIZE = largest dimension of the bounding box;
    Determine the case;
    CASE:
     $\mathcal{A}$   No action needed.
     $\mathcal{B}$   If the curve endpoints $\mathbf{C}_0$ and $\mathbf{C}_n$ are in different quadrants, increment $inter$.
     $\mathcal{C}$   If SIZE < ON_TOL, report ON and RETURN.
         Else if $d_s \geq d_t$, perform Bézier clipping against $\mathbf{L} = s - s_r = 0$. The resulting segments 1 and 3 are guaranteed to be case $\mathcal{A}$, so discard them. Push segment 2.
         Else, perform Bézier clipping against $\mathbf{L} = t - t_r = 0$. Push all three segments.
     END CASE
END WHILE
If $inter$ is odd, report OUT; else report IN
END CLASSIFY.
```

2.2.1 Implementation

A problem can arise when $\mathbf{R}$ happens to pass through an end control point shared by two trimming curves, because two intersections will be reported when one is often the correct answer. The solution we use is to perturb $\mathbf{S}$ away from $\mathbf{R}$ a sub-pixel distance <ON_TOL. For the same reason, it is important that $d(u_{min}) \neq 0$ and $d(u_{max}) \neq 0$ (in equation 4 and Figure 5) to within floating point precision. Therefore, make the adjustment $u_{min} = 0.99 * u_{min}$ and $u_{max} = 0.99 * u_{max} + 0.01$. Another reason for this adjustment is to avoid infinite loops. Bézier clipping a curve whose endpoint happens to lie on $\mathbf{R}$ (ie., $d_0 = 0$) results in segments 1 and 2 having zero length, and segment 3 is simply the original curve.

2.3 Performance

ON is the classification which is most expensive to compute. Our algorithm can typically compute an ON classification to seven decimal digits accuracy in four Bézier clips. Figure 6 shows the trimmed patch in Figure 1, using color to indicate how many total Bézier clips were used in classifying each ray intersection. The average number of Bézier clips per point to be classified was 1.04 in this example involving ten trimming curves. Figure 7 shows a patch trimmed with precision by eight Bézier curves.

3 RAY-PATCH INTERSECTION

Our algorithm for computing the intersection of a patch with a ray uses the Bézier clipping concept to iteratively clip away regions of the patch which don't intersect the ray.

The most costly single operation in a subdivision-based ray-patch intersection algorithm is de Casteljau subdivision. Typically, subdivision is performed in R^3 for non-rational patches, and in R^4 for rational patches. Woodward[20] (also alluded to by Rogers[10]) shows how the problem can be projected to R^2. This means that the number of arithmetic operations to subdivide a non-rational patch is reduced by 33% (since subdivision is applied only to x, y components, rather than x, y, z) and for a rational patch is reduced by 25% (subdivision in x, y, w rather than in x, y, z, w). Section 3.1 reviews that projection, and further shows how the rational case can be handled by subdividing only *two* components.

Section 3.2 then shows how to apply Bézier clipping to iteratively clip away regions of the projected patch which don't intersect the ray.

3.1 Projection to R^2

A rational Bézier surface patch in Cartesian three space $(\hat{x}, \hat{y}, \hat{z})$ is defined parametrically by

$$\hat{\mathbf{P}}(s,t) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(s) B_j^m(t) w_{ij} \hat{\mathbf{P}}_{ij}}{\sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(s) B_j^m(t) w_{ij}} \tag{6}$$

where $\hat{\mathbf{P}}_{ij} = (\hat{x}_{ij}, \hat{y}_{ij}, \hat{z}_{ij})$ are the Bézier control points with corresponding weights w_{ij}. (The symbols are hatted to later distinguish them from the projected (x, y) coordinate system).

As does Kajiya[6], we define the ray to be the intersection of two planes given by implicit equations

$$a_k \hat{x} + b_k \hat{y} + c_k \hat{z} + e_k = 0, \quad k = 1, 2. \tag{7}$$

We assume that the plane equations are normalized: $a_k^2 + b_k^2 + c_k^2 = 1$. In practice, it is best if the two planes are orthogonal. For primary rays, we use the scan plane and the plane containing the ray, parallel to the screen y axis.

The intersection of plane k and the patch can be represented by substituting equation 6 into equation 7 and clearing the denominator:

$$d^k(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(s) B_j^m(t) d_{ij}^k = 0 \tag{8}$$

where

$$d_{ij}^k = w_{ij}(a_k \hat{x}_{ij} + b_k \hat{y}_{ij} + c_k \hat{z}_{ij} + e_k). \tag{9}$$

$\hat{\mathbf{P}}(\hat{s}, \hat{t})$ lies on plane k iff $d^k(\hat{s}, \hat{t}) = 0$.

Note that d_{ij}^k is related to the distance from control point $\hat{\mathbf{P}}_{ij}$ to plane k:

$$d_{ij}^k = w_{ij} \times DISTANCE(\hat{\mathbf{P}}_{ij}, Plane\ k). \tag{10}$$

We can now project the patch to a two dimensional (x, y) coordinate system by taking the projected control point coordinates to be

$$\mathbf{P}_{ij} = (x_{ij}, y_{ij}) = (d^1_{ij}, d^2_{ij}) \qquad (11)$$

from equation 9. The projected patch is defined:

$$\mathbf{P}(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} B^n_i(s)B^m_j(t)\mathbf{P}_{ij} \qquad (12)$$

In this projection, plane 1 becomes the y axis, plane 2 becomes the x axis, and the ray projects to the coordinate system origin, $\mathbf{0}$. Figure 8 shows a sample projected patch $\mathbf{P}(s,t)$.

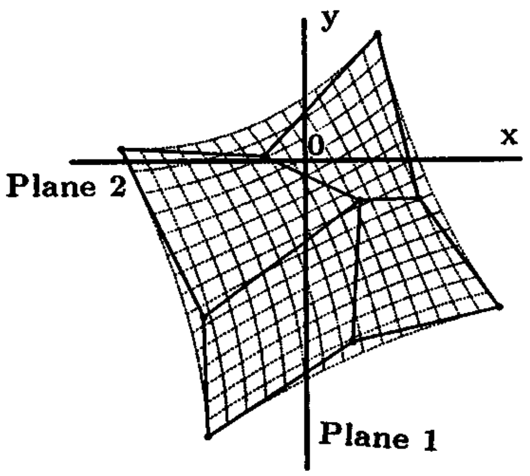

Figure 8: Projected patch **P**

The ray-patch intersection problem now becomes one of finding

$$\{(s,t) | \mathbf{P}(s,t) = \mathbf{0};\; 0 \le s,t \le 1\}. \qquad (13)$$

For a non-rational patch (that is, all $w_{ij} = 1$), $\mathbf{P}$ is a simple orthographic projection of $\hat{\mathbf{P}}$ along the ray. $\mathbf{P}(s,t)$ is always non-rational (all its weights are equal), *even if* $\hat{\mathbf{P}}(s,t)$ *is rational.*

3.2 BÉZIER CLIPPING P

This section applies Bézier clipping to the problem of finding all solutions to

$$\mathbf{P}(s,t) = \mathbf{0}. \qquad (14)$$

Begin by defining a line $\mathbf{L}_s$ through $\mathbf{0}$ parallel to the vector $\mathbf{V}_0 + \mathbf{V}_1$ as shown in Figure 9. Bézier clipping will be used to identify ranges of the s parameter in which $\mathbf{P}(s,t)$ does not map to $\mathbf{0}$.

If $\mathbf{L}_s$ is defined by its implicit equation

$$ax + by + c = 0, \quad a^2 + b^2 = 1 \qquad (15)$$

then the distance D_{ij} from each control point $\mathbf{P}_{ij} = (x_{ij}, y_{ij})$ to $\mathbf{L}_s$ is

$$D_{ij} = ax_{ij} + by_{ij} + c. \qquad (16)$$

The D_{ij} are shown in Figure 10. Likewise, the distance $D(s,t)$ from $\mathbf{L}_s$ to any point $\mathbf{P}(s,t)$ on the projected patch is

$$D(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} B^n_i(s)B^m_j(t)D_{ij} \qquad (17)$$

The function $d(s,t)$ can be represented, in an (s,t,d) coordinate system, as an *explicit* (or so-called *non-parametric*) surface

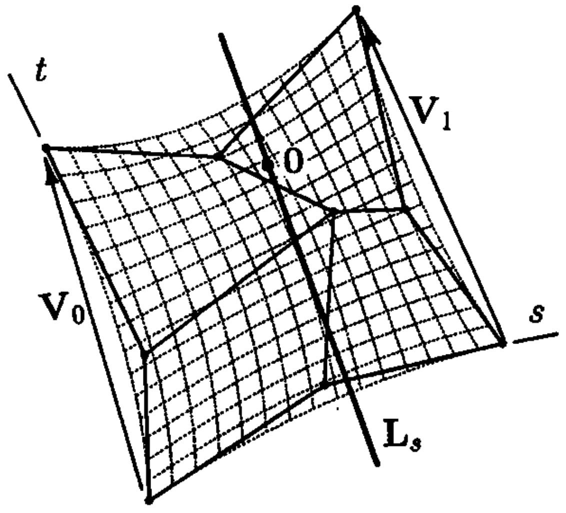

Figure 9: Line $\mathbf{L}_s$

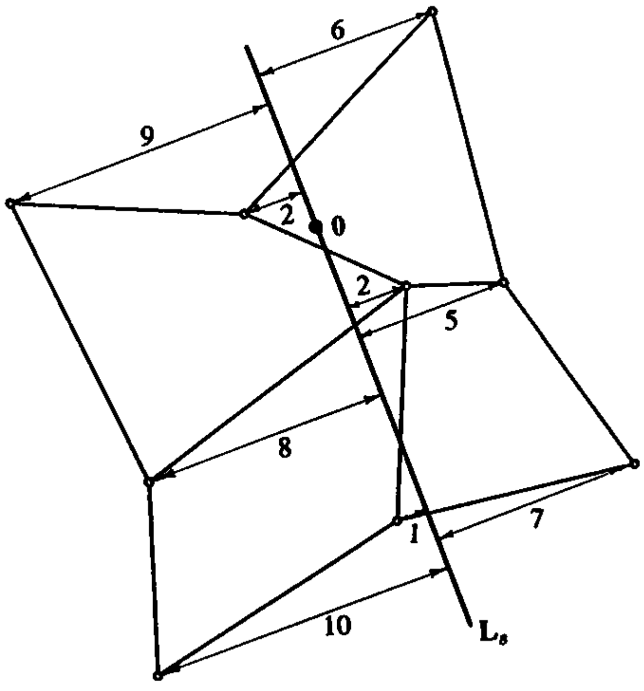

Figure 10: Control point distances

patch[1] whose control points $\mathbf{D}_{ij} = (s_{ij}, t_{ij}, D_{ij})$ are evenly spaced in s and t: $s_{ij} = \frac{i}{n}$, $t_{ij} = \frac{j}{m}$. A point on such a patch has coordinates

$$\mathbf{D}(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(s) B_j^m(t) \mathbf{D}_{ij} = (s, t, D(s,t)). \quad (18)$$

The top view of this patch is shown in Figure 11, with control point D coordinates labeled. Compare those values with the distances in Figure 10.

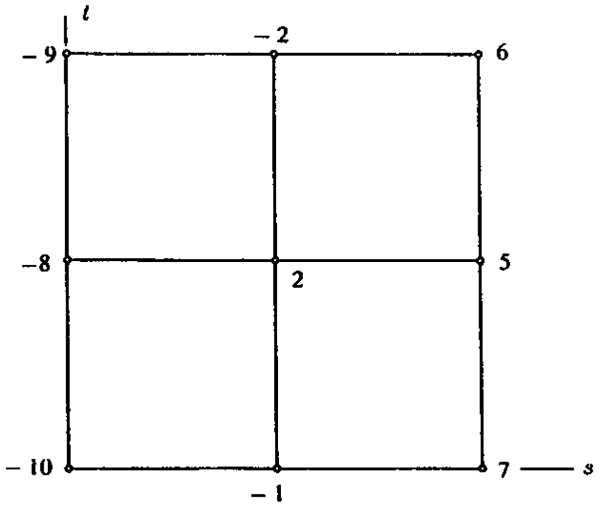

Figure 11: Top view of $\mathbf{D}(s,t)$ patch.

A side view of the $\mathbf{D}(s,t)$ patch, looking down the t axis, is shown in Figure 12. The convex hull of the projected control points

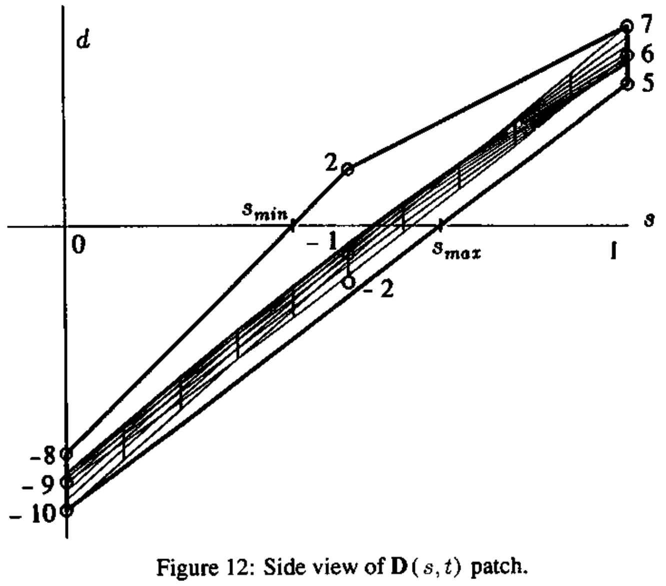

Figure 12: Side view of $\mathbf{D}(s,t)$ patch.

bounds the projection of the $\mathbf{D}(s,t)$ patch. In this example, that convex hull intersects the s axis at points $s_{min} = 2/5$ and $s_{max} = 2/3$. We conclude that $d(s,t) \neq 0$, and therefore $\mathbf{P}(s,t) \neq \mathbf{0}$, for $s < 2/5$ and $s > 2/3$. The de Casteljau subdivision algorithm is applied to clip away those regions, leaving the two dimensional patch in Figure 13.

This process of identifying values s_{min} and s_{max} which bound the solution set, and then subdividing off the regions $s < s_{min}$ and $s > s_{max}$ will be referred to as *Bézier clipping in s*. In an obviously similar manner, we define the process of Bézier clipping in t.

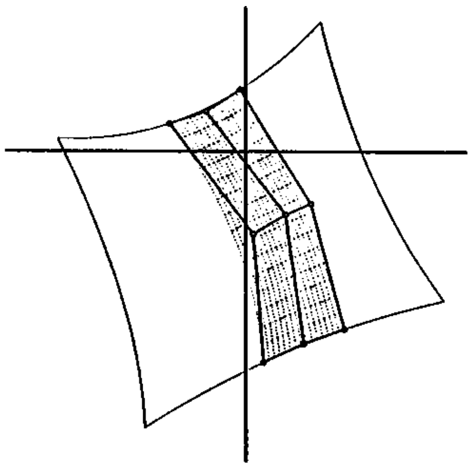

Figure 13: First clip in s.

Our ray-patch intersection algorithm consists of alternately performing Bézier clipping in s and t. Figure 14 shows the patch after Bézier clipping in s, then t, and again in s. The remaining sub-

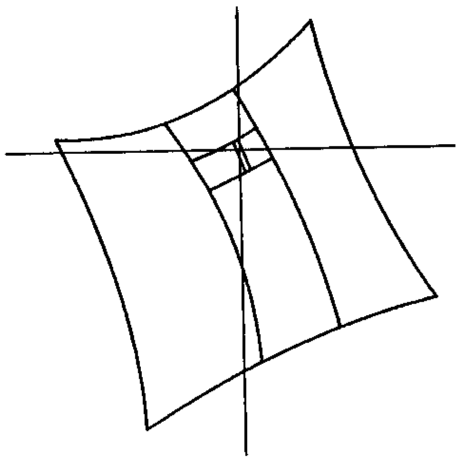

Figure 14: Iterating to the solution.

patch is not yet small enough to satisfy our tolerance conditions. (In this case, let's assume the tolerance value is 10^{-4}. In practice, one should pick a tolerance value which assures sub-pixel accuracy. This is done by finding a bound on the largest first derivative of screen space x or y with respect to parameter space s or t.) However, in computing t_{min}, t_{max} for the next t clip, it turns out that $t_{max} - t_{min} < 10^{-4}$. *Without subdividing in t*, we then compute s_{min}, s_{max} for an s clip, and discover that $s_{max} - s_{min} < 10^{-4}$ also. Thus, in the final step, we compute the intersection to within tolerance without actually subdividing to the clip values. The total number of operations in this typical example is: s_{min}, s_{max} or t_{min}, t_{max} is computed five times, and three pairs of de Casteljau subdivisions are performed.

3.3 No intersection

If a Bézier trim calculation determines $s_{min} > 1$, $s_{max} < 0$, $t_{min} > 1$, or $t_{max} < 0$, the ray does not intersect the patch.

It is possible to have $t_{max} - t_{min} < 10^{-4}$ and $s_{max} - s_{min} < 10^{-4}$, and yet the ray does not intersect the patch. However, this can only occur if $s_{min} = 0$, $s_{max} = 1$, $t_{min} = 0$, or $t_{max} = 1$. Whenever the subpatch lies on the boundary of the original patch, even if the tolerance criteria is satisfied, an additional Bézier clip calculation should be performed to assure that the intersection does lie within the patch boundaries.

3.4 Multiple intersections

If there are multiple intersections, Bézier clipping will not converge to a single value. Therefore, if a Bézier clip fails to reduce the parameter interval width by at least, say, 20%, split the patch in half and resume Bézier clipping on each half. There is little theoretical basis for this value of 20%. Empirically, 20% seems to provide nearly optimal performance in most cases. Execution speed is not highly sensitive to small changes in this 20% value, although for values less than 10%, excessive Bézier clipping can occur and for values greater than 40%, unnecessary binary subdivision can occur.

The case of multiple intersections is illustrated in Figure 15. First, Bézier clipping in s discards regions labelled 1. In attempting to clip in t, it turns out that $t_{max} - t_{min} > 0.8$. Therefore, the remaining domain is subdivided in half at $t = 0.5$. A stack data structure is used to store subpatches. We push one of the two subpatches onto the stack, and proceed to process the other subpatch by Bézier clipping in s to eliminate regions 2 and clipping in t to remove regions 3. As in the example in Figure 14, without further subdivision we can compute the intersection which lies between regions 3 to within tolerance.

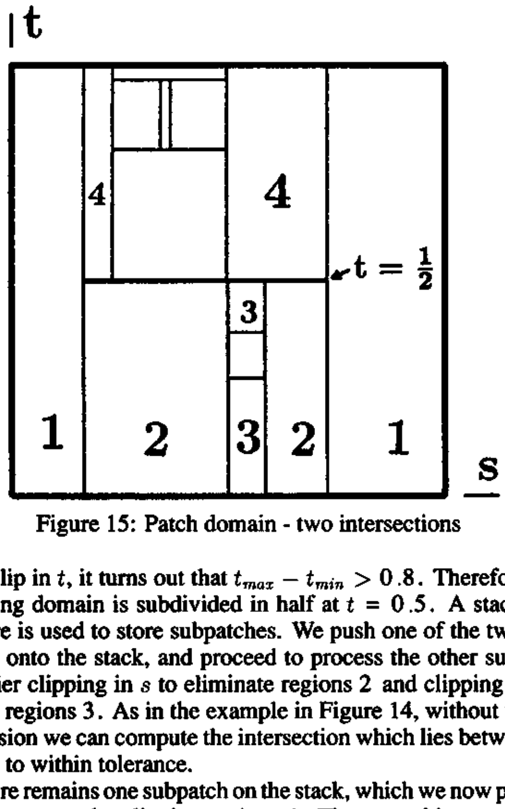

Figure 15: Patch domain - two intersections

There remains one subpatch on the stack, which we now pop and begin to process by clipping regions 4. The second intersection is refined in two more Bézier clips, as shown.

3.5 Primary Ray Preprocessing

Bézier clipping can be used to advantage in a preprocessing step applied at the initialization of each scan line. By Bézier clipping **P** against the scan line in both s and t directions, regions of **P** can be discarded which do not intersect any primary ray along the scan line. Figure 16 shows the patch in Figure 8 after Bézier clipping against the scan line (x axis). By performing this preprocess, a savings of up to two subdivisions per primary ray/surface intersection can be realized, and also non-intersecting rays can be detected more often. For example, in applying this preprocessing to rendering the teapot in Figure 17, 86% of the calls to the intersection routine resulted in a hit.

3.6 Implementation

To avoid potential infinite loops due to numerical roundoff, make the adjustment $s_{min} = 0.99 * s_{min}$ and $s_{max} = 0.99 * s_{max} + 0.01$

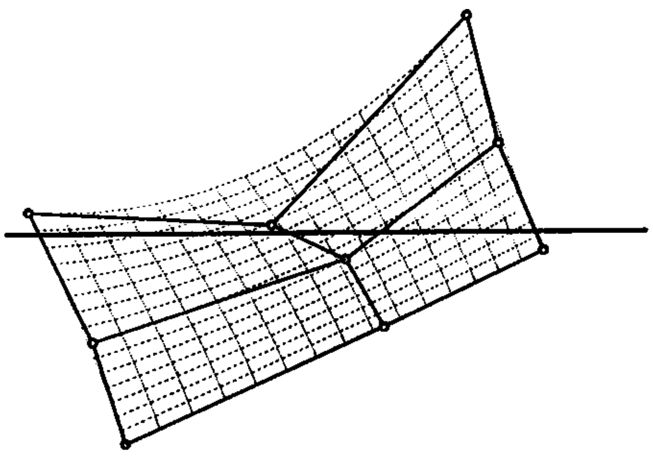

Figure 16: Trimming to the Scan Line.

and similarly for t in computing values at which to Bézier clip.

Other than the ray/patch intersection algorithm, all of the other implementation details are standard. Antialiasing was performed using adaptive supersampling [5], and Murakami's voxel partitioning [8] was implemented. Shadows, reflection and refraction are dealt with in the conventional manner, using our ray-patch intersection algorithm.

4 DISCUSSION

4.1 Examples and Timings

We tested the algorithm on an Iris-4D/70GT workstation and created Figures 17-20 at 500X500 resolution. Each figure caption lists the number of patches in the scene, total CPU time for rendering, (CPU time for rendering a similar scene, not antialiased, with primary rays only), percentage of background pixels, and average number of patch subdivisions per foreground pixel for primary rays. All patches are non-rational bicubics. The chains in Figure 19 were oriented using Free-Form Deformation[14].

4.2 Performance Comparisons

It is difficult to derive precise quantitative comparisons between various ray/patch intersection algorithms. The predominant single expense in most ray-patch intersection algorithms is de Casteljau subdivision. Our intersector spends 45% of its time computing subdivisions. To split a projected two-dimensional bicubic patch in either parameter direction requires 144 floating point operations. All previous subdivision-based algorithms can take advantage of this projection to R^2 (which, as mentioned, saves 33% of subdivision costs for non-rational and 50% for rational patches).

We compared the number of subdivisions per non-background pixel required by our algorithm with the algorithms of Toth[18] and Woodward[20]. To attain three digits accuracy in s, t, Toth reported an average of 19.66 subdivisions for each non-background pixel in his Figure 6 (an example of a single patch in which roughly 30% of non-background pixels involve two ray-patch intersections). We duplicated the patch and viewing parameters for Toth's Figure 6 as nearly as possible, and tested our and Woodward's algorithms. The average number of subdivisions per non-background pixel is listed in Table 1. SUN SPARCstation I CPU times for 120X120 image are shown in parenthesis.

It is difficult to make quantitative comparisons with the other published algorithms. Newton [17] and quasi-Newton [4] itera-

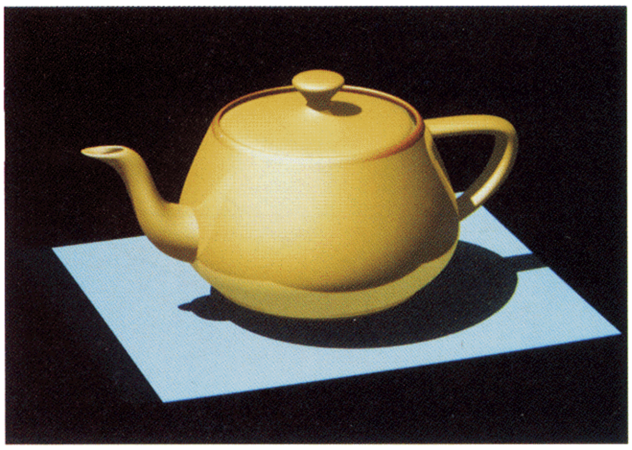

Figure 17: Newell's teapot: 33 patches, 65% background, 5.2 subdivisions/ray, 6.7 cpu min. (3.9 min. primary rays only)

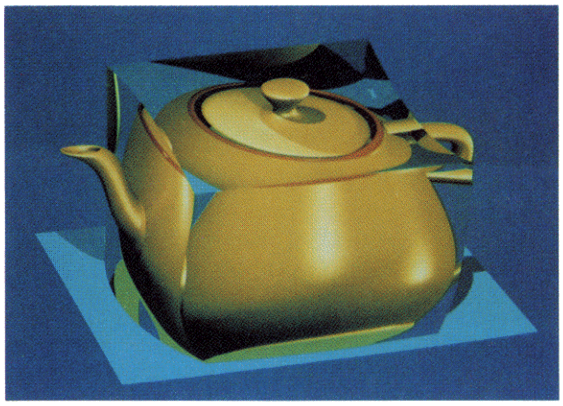

Figure 20: Teapot encased in deformed glass cube: 39 patches, 21.3 cpu min on SUN SPARCstation I

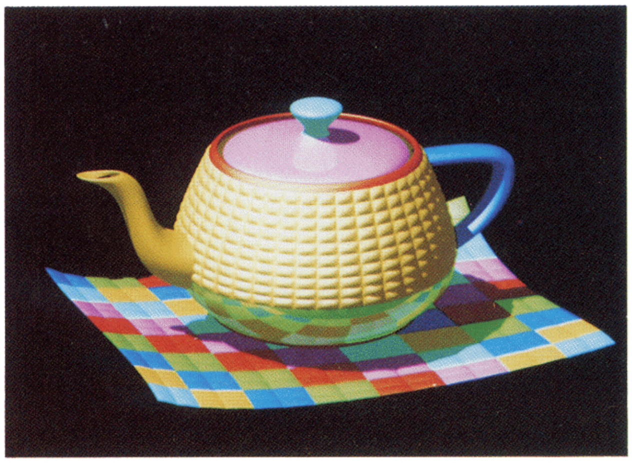

Figure 18: Modified teapot: 2304 patches, 64% background, 4.0 subdivisions/ray, 12.5 cpu min. (8.6 min. primary rays only)

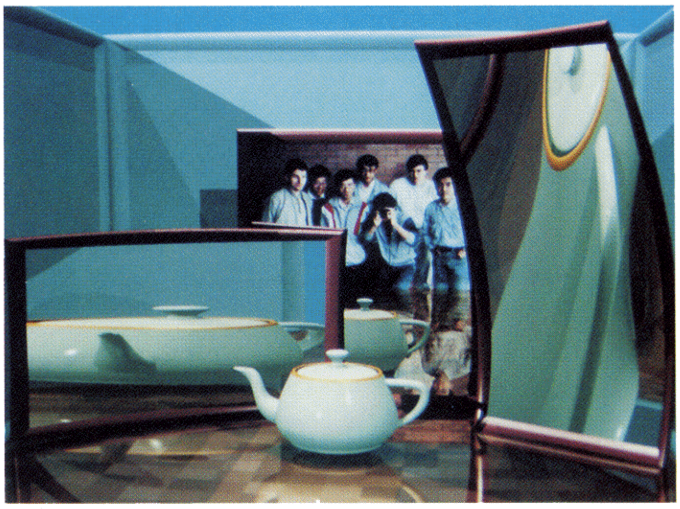

Figure 21: House of mirrors

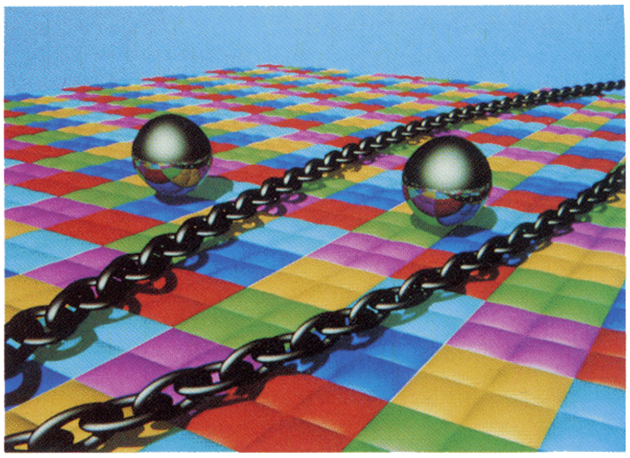

Figure 19: Chain on patch-work quilt: 4024 patches, 18% background, 5.6 subdivisions/ray, 29.0 cpu min. (17.6 min. primary rays only)

Algorithm	Tolerance	
	2^{-6}	2^{-10}
Our's	8.6 (24 sec)	11.4 (26 sec)
Woodward	29.2 (80 sec)	48.6 (116 sec)
Toth	NA	19.6

Table 1: Subdivisions per pixel and (total cpu time)

tion appear to converge quickly, but without actually implementing those algorithms, we cannot estimate the computational expense required to assure robustness.

4.3 Higher Degree Patches

This algorithm works on patches of arbitrary degree. However, since de Casteljau subdivision is an $O(n^3)$ operation for surface patches, execution speed suffers as patch degree increases. A sample bicubic patch containing a silhouette curve took 50 cpu seconds to render with 11.4 subdivisions per pixel. After elevating that same patch to degree four, the rendering took 94 cpu seconds with 11.2 subdivisions per pixel. The subdivisions per pixel tends to decrease with degree because the control polygon approximates the patch more closely as degree elevation is applied. Table 2 tallies the execution speed for patches elevated up to degree eight.

Degree	3	4	5	6	7	8
Seconds	50	94	130	186	243	322
sub/pixel	11.4	11.2	10.9	10.9	10.7	10.7

Table 2: Higher Degree Patches

Acknowledgements: The authors gratefully acknowledge the support of Professor Nakamae of Hiroshima University in providing access to his computer graphics lab. The second author was supported in part by the National Science Foundation under grant number DMC-8657057. This work originated while the first author was visiting Brigham Young University, on leave from Fukuyama University.

REFERENCES

1. Böhm, Wolfgang, Farin, Gerald and Kahmann, Jurgen. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design,* 1, 1 (1984), 1-60.

2. Farin, Gerald. *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1988.

3. Glassner, Andrew ed., *Introduction to Ray Tracing.* Academic Press, 1989.

4. Joy, Keneth and Bhetanabhotla, Murthy. Ray Tracing Parametric Patches Using Numerical Techniques and Ray Coherence. Proceedings of SIGGRAPH '86 (Dallas, TX, August 18-22, 1986). In *Computer Graphics*, 20, 4 (August 1986), 279-285.

5. Joy, Kenneth , Grant, Charles ,Max, Nelson and Hatfield, Lansing. *Computer Graphics: Image Synthesis.* Computer Society Press, 1988.

6. Kajiya, Jim. Ray tracing parametric patches. Proceedings of SIGGRAPH '82 (Boston, MA, July 26-30, 1982). In *Computer Graphics*,16 ,3 (July 1982), 245-254.

7. Lane, Jeff and Riesenfeld, Rich. A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces. *IEEE Trans. RAMI*,2 (1980), 35-46.

8. Murakami, Koichi, Hirota, Katsuhiko and Ishii, Mitsuo. Fast ray tracing. *Fujitsu Science and Technology Journal*, 24, 2 (1988), 150-159.

9. Rockwood, Alyn, Heaton, Kurt and Davis, Tom. Real-Time Rendering of Trimmed Surfaces. Proceedings of SIGGRAPH '89 (Boston, MA, July 31 - August 4, 1989). In *Computer Graphics*, 23, 3 (July 1989), 107-117.

10. Rogers, Dave. *Procedural Elements for Computer Graphics.* McGraw-Hill, New York, 1985, 296-305.

11. Roth, Scott. Ray Casting for Modeling Solids. *Computer Graphics and Image Processing*, 18, 1982, 109-144.

12. Sederberg, Tom. An Algorithm for Algebraic Curve Intersection. *Computer-Aided Design*, 21, 9 (1989), 547-554.

13. Sederberg, Tom and Parry, Scott. A Comparison of Three Curve Intersection Algorithms. *Computer-Aided Design*, 18, 1 (1986), 58-63.

14. Sederberg, Tom and Parry, Scott. Free-Form Deformation of Solid Geometric Models. Proceedings of SIGGRAPH '86 (Dallas, TX, August 18-22, 1986). In *Computer Graphics*, 20, 4 (August 1986), 151-160.

15. Sederberg, Tom , White, Scott and Zundel, Alan. Fat Arcs: A Bounding Region with Cubic Convergence. *Computer Aided Geometric Design*, 6 (1989), 205-218.

16. Shantz, Mike and Chang, Sheue-Ling. Rendering Trimmed NURBS with adaptive Forward Differencing. Proceedings of SIGGRAPH '88 (Atlanta, GA, August 1-5, 1988). In *Computer Graphics*, 22, 4 (August 1988), 189-198.

17. Sweeney, Michael and Bartels, Richard. Ray Tracing Free-Form B-Spline Surfaces. *IEEE CG&A*, 6, 2, 1986, 41-49.

18. Toth, Dan. On Ray Tracing Parametric Surfaces. Proceedings of SIGGRAPH '85 (San Francisco, CA, July 22-26, 1985). In *Computer Graphics* 19, 3 (July 1985), 171-179.

19. Whitted, Turner. An Improved Illumination Model for Shaded Display. *CACM*, 23, 6, 1980, 96-102.

20. Woodward, Charles. Ray Tracing Parametric Surfaces by Subdivision in Viewing Plane. in W. Strasser and H.-P. Seidel, editors, *Theory and Practice of Geometric Modeling*, Springer-Verlag, 1989, 273-290.