RAY TRACING PARAMETRIC PATCHES

James T. Kajiya
California Institute of Technology
Pasadena, Ca. 91125

ABSTRACT. This paper describes an algorithm that uses ray tracing techniques to display bivariate polynomial surface patches. A new intersection algorithm is developed which uses ideas from algebraic geometry to obtain a numerical procedure for finding the intersection of a ray and a patch without subdivision. The algorithm may use complex coordinates for the (u, v)-parameters of the patches. The choice of these coordinates makes the computations more uniform, so that there are fewer special cases to be considered. In particular, the appearance and disappearance of silhouette edges can be handled quite naturally. The uniformity of these techniques may be suitable for implementation on either a general purpose pipelined machine, or on special purpose hardware.

KEYWORDS: computer graphics, raster graphics, ray tracing, parametric patches

CR CATEGORIES: I.3.3, I.3.5, I.3.7

From its inception, the method of ray tracing has always been the technique of choice when ultimate realism of computer generated images is the goal [Appel 1968]. This paper describes a new technique for generating ray traced images of piecewise polynomial bivariate parametric patches. In contrast to other algorithms to do this, the new algorithm does not repeatedly subdivide a patch, but rather calculates an intersection between a patch and a ray using more or less direct numerical procedures.

This procedure has a number of pleasant characteristics. First, for patches of low degree, the algorithm proceeds more quickly. In fact, if the patch coefficients degenerate to a planar surface the amount of computation needed to intersect a ray with it is roughly the same as for an algorithm tuned for ray-plane intersections. Second, the algorithm is robust — many patch algorithms need preliminary subdivisions to satisfy some ***a priori*** approximation [Blinn, ***et. al.*** 1980], the algorithm presented here has no such requirement.

This paper treats the intersection problem only. There are many other components to a ray tracing system, such as the lighting model calculation [Whitted 1980], the use of object coherence to mitigate scene complexity [Rubin and Whitted 1980], antialiasing in the ray tracing context [Whitted 1980], and the use of reflectance and normal vector perturbation mapping techniques to enhance realism [Blinn 1978b]. We study the intersection problem because it has been found to be the most time critical step.

§1 Notation

This section establishes the general notation of the paper. We also set up the basic equations for patch, line, and plane definitions.

A point in **RP3**, real projective 3 space, is given by its homogeneous coordinates, i.e. a 4-vector

$$x^k, \qquad k = 0, 1, 2, 3.$$

Usually (x^0, x^1, x^2, x^3) is written as

$$(x, y, z, w) = (x^1, x^2, x^3, x^0).$$

A typical representation of a bicubic patch is as a set of 4 by 4 matrices, one matrix for each homogeneous coordinate:

$$p^k_{ij}, \qquad i,j,k = 0,\ldots,3.$$

To calculate the 4D point corresponding to a chosen u,v-parameter value we use

$$x^k = p^k_{ij}u^{3-i}v^{3-j} \qquad k = 0,\ldots,3. \qquad (1)$$

Throughout this paper we shall use the ***summation convention***: if an index appears twice in a term then it is summed across the range of its values. The expression above when written in full is actually

$$x^k = \sum_{i=0}^{3}\sum_{j=0}^{3} p^k_{ij}u^{3-i}v^{3-j} \qquad k = 0,\ldots,3$$

or in the familiar matrix form

$$x^k = (u^3\ u^2\ u\ 1)(P^k)\begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix} \qquad k = 0,\ldots,3$$

where $P^k = (p^k_{ij})$. Note that u^n is an algebraic n^{th} power while x^k is a coordinate indexed by k. No confusion should ever result since parameter values are always multiplied and never indexed while spatial coordinates are always indexed and never multiplied.

Now a ray has several co-ordinate representations. One can represent it as two points on the line, a point and direction cosines, or the intersection of two planes. For our purposes we choose the latter representation. Thus a line is represented by the equation:

$$l^\alpha_k x^k = 0, \qquad \alpha = 0,1. \qquad (2)$$

§2 Intersecting A Ray with A Patch

This section discusses the process we use to transform the ray-patch intersection problem in real projective 3 space into the problem of intersecting two algebraic curves in complex 2 space. By performing such a transformation we will be able to make use of tools available from algebraic geometry.

To intersect a patch and a ray we substitute equation (1) into equation (2) to obtain:

$$l^\alpha_k p^k_{ij} u^{3-i}v^{3-j} = 0 \qquad \alpha = 0,1$$

or, setting

$$l^0_k p^k_{ij} = a_{ij}$$
$$l^1_k p^k_{ij} = b_{ij}$$

we have the equations:

$$a_{ij}u^{3-i}v^{3-j} = 0$$
$$b_{ij}u^{3-i}v^{3-j} = 0.$$

Each of these two equations define the algebraic curves formed by the intersecting the patch with each of the two line defining planes. The intersection points of these two algebraic curves in turn give the u,v-parameter values at which the ray intersects the patch. This proves our first theorem.

Theorem 1. *Let a patch be given by a 4 by 4 by 4 array of coefficients p^k_{ij} and a line be given by its 2 by 4 array of coefficients l^α_k. To find the parameter values u,v at which they intersect we may find the solution of two algebraic equations:*

$$a_{ij}u^{3-i}v^{3-j} = 0$$
$$b_{ij}u^{3-i}v^{3-j} = 0$$

where

$$a_{ij} = l^0_k p^k_{ij}$$

and

$$b_{ij} = l^1_k p^k_{ij}$$

Proof: ∎

The above two equations must be solved simultaneously. They represent degree six algebraic curves in the (u,v)-plane. They are not, however, completely arbitrary degree six curves. We can consider algebraic curves as points in a vector space. This is done via their defining polynomial equations since the coefficients of the defining polynomial form a vector space. In our case above, the dimension of the vector space of curves is only half the dimension of the vector space of general degree six algebraic curves. As we shall see, the smaller space cuts down the number of possible solutions by half.

§3 The Resultant

We now present the key observation of this paper: that it is possible to intersect algebraic curves via a relatively straightforward mechanical procedure. We describe the actual intersection algorithm in section 5. This section and the next will justify the steps involved in the algorithm. The reader may safely skip to section 5 on a first reading — at the risk of being somewhat mystified by a few of the steps of the algorithm.

In finding a procedure to intersect two algebraic curves given by bivariate polynomials, we use some results about univariate polynomials. These results may be found in most books treating algebraic geometry or the theory of equations [Walker 1950, Littlewood 1970, or Uspenskii 1948].

We now state the central definition of this section.

Definition. The *resultant* of two polynomials $a(u) = a_i u^{n-i}$ and $b(u) = b_j u^{m-j}$ of degree n and m, resp. whose roots are $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$ is defined as:

$$R(a,b) = a_0^m b_0^n (-1)^{m+n-1} \prod_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}} (\alpha_i - \beta_j). \qquad (3)$$

From this definition follows two obvious facts.

Theorem 2. *The resultant of two polynomials is zero iff they share a common zero.*

Proof: Obvious from the definition, since if $\alpha_i = \beta_j$ for some i, j then and only then would the expression for $R(a,b)$ vanish. ▮

Theorem 3. *The resultant $R(a,b)$ is homogeneous and symmetric in each set of roots $\alpha_1, \ldots, \alpha_n$ of $a(u)$ and $\beta_1, \ldots, \beta_n$ of $b(u)$ considered independently.*

Proof: Obvious from inspection of the definition of the resultant and the definition of a symmetric polynomial given immediately below. ▮

Definition. An expression $f(x^1, \ldots, x^n)$ of n variables is called *symmetric* if its value is unchanged upon any permutation of its arguments.

Example. Here are some symmetric functions for $n = 3$:

$$\begin{aligned} s_1(x^0, x^1, x^2) &= x^0 + x^1 + x^2 \\ s_2(x^0, x^1, x^2) &= x^0x^1 + x^0x^2 + x^1x^2 \\ s_3(x^0, x^1, x^2) &= x^0x^1x^2. \end{aligned}$$

Why is it important to notice that the resultant is symmetric in the roots of the polynomials a and b? It is because of the next theorem.

Theorem 4. *Let*

$$a(u) = a_0 u^k + a_1 u^{k-1} + \cdots + a_k$$

be a polynomial with roots $\alpha_1, \ldots, \alpha_k$. Then any expression Φ symmetric in the α_i can be rewritten as a polynomial in the coefficients a_i of $a(u)$ divided by a_0 and a factor of -1.

Proof: The difference of two symmetric polynomials is symmetric. We form the basic symmetric polynomials given in the example below. It is possible to lower the degree of Φ simply by matching the highest degree monomial with a power of one of the basic symmetric terms. Induction finishes the proof. ▮

Example. Let $a(u) = a_i u^{3-i}$ be a cubic polynomial. Then from the previous example we have:

$$\begin{aligned} (-1)^1 a_0 s_1(\alpha_1, \alpha_2, \alpha_3) &= \alpha_1 + \alpha_2 + \alpha_3 = a_1 \\ (-1)^2 a_0 s_2(\alpha_1, \alpha_2, \alpha_3) &= \alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_3 = a_2 \\ (-1)^3 a_0 s_3(\alpha_1, \alpha_2, \alpha_3) &= \alpha_1\alpha_2\alpha_3 = a_3. \end{aligned}$$

Finally, we can now state the key fact that we need to intersect algebraic curves.

Theorem 5. *The resultant of two polynomials can be expressed as a polynomial in the coefficients of each.*

Proof: Use theorem 3 and theorem 4. ▮

This important result allows us to determine whether two univariate polynomials share a common root by calculating directly from their coefficients without first factoring each polynomial into its roots. We merely calculate the resultant and test for zero. For univariate polynomials the resultant gives us a convenient method for testing for common zeroes of two polynomials. For bivariate polynomials such a convenience becomes necessity: the resultant allows us to solve the curve intersection problem in u and v separately.

The resultant $R(a, b)$ may be expressed as a polynomial in the coefficients. What does this polynomial look like? We might multiply out the product in equation (3) and attempt to calculate the polynomial directly. Fortunately, there is an easier way.

§4 The Bezout Determinantal Form of the Resultant

This section discusses a method for directly calculating the resultant $R(a, b)$ as a polynomial in the coefficients of a and b. The resultant takes on a determinantal form whose appearance depends on the degrees of the polynomials a and b. We show how to generate a determinant for each degree and display determinants for two cases: the full degree six bicubic case and the case for the intersection of two lines.

If $a(u)$ and $b(u)$ share a common root ξ then $a(\xi) = 0$ and $b(\xi) = 0$. Furthermore, if $a(u)$ and $b(u)$ have degrees n, m we can without loss of generality assume that $n \geq m$ (or, in other words, that $n = m + r, \quad r > 0$). It is then easy to see following set of equations also obtain:

$$\begin{aligned}
b_0 a(\xi) - a_0 \xi^r b(\xi) &= 0 \\
(b_0\xi + b_1)a(\xi) - (a_0\xi + a_1)\xi^r b(\xi) &= 0 \\
&\vdots \\
(b_0\xi^{m-1} + b_1\xi^{m-2} + \cdots + b_{m-1})a(\xi) & \\
-(a_0\xi^{m-1} + a_1\xi^{m-2} + \cdots + a_{m-1})\xi^r b(\xi) &= 0 \\
\xi^{r-1} b(\xi) &= 0 \\
\xi^{r-2} b(\xi) &= 0 \\
&\vdots \\
b(\xi) &= 0.
\end{aligned}$$

If we multiply out these equations and collect like powers of ξ we get a system of algebraic equations. Now, if $1, \xi, \xi^2, \ldots, \xi^n$ are considered to be independent variables we then have a linear system of equations. For that system to have a solution the determinant of the coefficients must be zero. This determinant is exactly the resultant $R(a, b)$.

Let us illustrate the case for $n = 3$, $m = 3$, and $r = 0$. We have the following determinant:

$$R(a,b) = \begin{vmatrix}
\begin{vmatrix} b_0 & b_1 \\ a_0 & a_1 \end{vmatrix} & \begin{vmatrix} b_0 & b_2 \\ a_0 & a_2 \end{vmatrix} & \begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} \\
\begin{vmatrix} b_0 & b_2 \\ a_0 & a_2 \end{vmatrix} & \begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} + \begin{vmatrix} b_1 & b_2 \\ a_1 & a_2 \end{vmatrix} & \begin{vmatrix} b_1 & b_3 \\ a_1 & a_3 \end{vmatrix} \\
\begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} & \begin{vmatrix} b_1 & b_3 \\ a_1 & a_3 \end{vmatrix} & \begin{vmatrix} b_2 & b_3 \\ a_2 & a_3 \end{vmatrix}
\end{vmatrix}$$

If this determinant is zero then the two cubic polynomials have a root in common. Another example is if $n = m = 1$. The system of equations then reduces to

$$R(a,b) = \begin{vmatrix} b_0 & b_1 \\ a_0 & a_1 \end{vmatrix}$$

which is the ordinary determinant encountered when solving this particular linear system of equations obtained by intersect the two lines by more conventional means.

The above argument shows that the vanishing of the Bezout determinant is a necessary condition for a common root to occur, but not a sufficient condition. The argument for sufficiency is more involved: it involves a manipulation of the definition given in the previous section.

§5 Calculating the Intersection of Two Curves

We need to intersect the two algebraic curves given by the locus of solutions of two bicubic polynomials $a(u, v)$, $b(u, v)$. We use the resultant to find such intersections by a trick. This trick is to consider bivariate cubic polynomials in u and v to be univariate polynomials in v with coefficients polynomials in u, i.e.

$$C[u, v] = C[u][v].$$

Thus to calculate the intersection of the $a(u, v)$ and $b(u, v)$ curves we simply find common roots of the (univariate) cubic polynomials in v. By theorem 2, the vanishing of the resultant

$$R(a,b) = 0$$

gives us a necessary and sufficient condition for an intersection to occur. From theorem 5, the resultant is a polynomial in the coefficients of a and b. Since each of the coefficients of these polynomials are again polynomials in u, the resultant is a polynomial expression in these polynomials. Upon substituting, we obtain for the resultant a polynomial $r(u)$ in u, i.e.

$$R(a,b) \equiv r(u)$$

The vanishing of this polynomial is the condition for a common root in v, i.e. The roots of the equation

$$r(u) = 0$$

gives the u-coordinates of the intersection points of the two curves.

The roots $\mu_1, \mu_2, \ldots, \mu_q$ of the polynomial $r(u)$ give the values of the u parameter at which intersections occur. To find the values of the v parameter at which intersections occur we successively substitute μ_i into the coefficients to obtain polynomials in v. Since we already know that these polynomials have a common root it is unnecessary to solve each polyomial equation separately and match roots. Rather, we may simply compute the GCD of the two polynomials to obtain a polynomial which divides both polynomials. Often this GCD polynomial is of degree one allowing us to read off the v parameter value directly from its coefficients.

Here is a summary of the intersection procedure:

1. Compute the Bezout determinantal form $r(u)$ of the resultant $R(a,b)$ of the two bicubic polynomials $a(u,v)$,$b(u,v)$ each considered as (univariate) cubic polynomials in v with coefficients in $C[u]$.
2. Solve $r(u) = 0$ to obtain $\mu_1, \ldots, \mu_q$ the u parameter values of the intersection points.
3. For each μ_i calculate via the Euclidean algorithm the GCD of the cubic polynomials $a(\mu_i, v)$, $b(\mu_i, v)$.
4. Find the roots ν_{ij} of the GCD to obtain the v parameter values of the intersection points.

There are a number of special cases in the above algorithm. First, when calculating the Bezout determinant one of the polynomials, say $a(u,v)$, may have degree 0 in v. In this case the determinant need not be computed at all since $a(u,v)$ and $b(u,v)$ have a common zero when and only when $a(u)$ has a zero. Second, the resultant relies on the fact that $a_0, b_0 \neq 0$ for the two polynomials. When these coefficients are themselves polynomials we get spurious zeroes of the Bezout determinant at the roots of a_0, b_0. This is easily detected at the GCD stage: the the two substituted polynomials will be mutually prime. Third, the occurrence of multiple intersections will cause the GCD of the substituted polynomials to have degree higher than one. This is easily treated by simply solving the resulting GCD polynomial for all roots. Finally, the presence of numerical error makes computing the GCD polynomial somewhat delicate. We use the Euclidean algorithm modified by treating very small coefficients as zero. We caution however that a careful evaluation of the algorithm with respect to numerical error and stability has not yet been undertaken.

Figures 1 and 2 show the results of applying this procedure to two different patches. We have not yet incorporated the lighting models which include reflection, refraction, and shadows.

§6 Bezout's Theorem

How many different intersections can a ray have with a bicubic patch? We shall show that there are 18 possible intersections—they are, however, not necessarily real. The next theorem that tells us at how many points two distinct algebraic curves can intersect. It is a generalization of the fundamental theorem of algebra.

Theorem 6. *(Bezout). If two curves of degree m and n do not share a common factor, then they intersect in exactly mn points counted with multiplicities.*

Proof: See Walker (1950). ∎

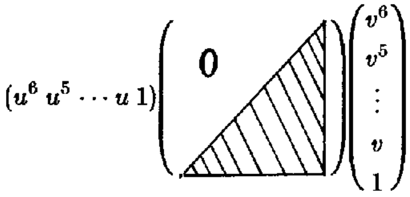
Note that if one of the curves is linear then it has degree 1 and we collapse to the fundamental theorem of algebra. This example points out that the theorem does not hold if we limit our attentions to the real case. Nor should we ignore the points at infinity. Thus to properly hold, Bezout's theorem is stated in the setting of complex homogeneous coordinates.

Because bicubic polynomials are not general polynomials of degree six we may improve upon Bezout's theorem. Bezout's theorem implies that there may be up to 36 intersections of a ray with a patch, but for the case of bicubic polynomials many of these intersections are always multiple.

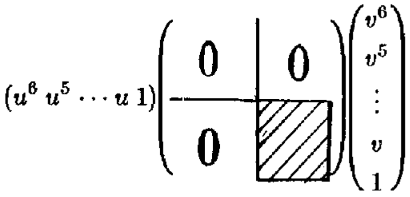
Each a_{ij} is obviously a point in 16-dimensional polynomial space. A general degree six curve is the locus of solutions of the equation

$$\sum_{i+j\leq 6} c_{ij}u^{6-i}v^{6-j} = 0$$

or as matrices, the general degree six looks like



while our case appears as



Theorem 7. *Two bicubic curves intersect at exactly 18 points in* **CP2**, *the complex projective plane.*

Proof: Computing the resultant of the two polynomials as in the intersection procedure above we see that the resultant polynomial $r(u)$ has degree at most 18. Each root of this polynomial gives a u-value at which the a and b polynomials have a root in common.

§7 Solving Polynomial Equations

At the heart of the algorithm presented above is a requirement to repeatedly solve univariate polynomial equations. Indeed, as we shall see, the bulk of the computation time for the whole intersection calculation is taken by the root finding procedure. It is thus crucial to use a highly efficient root finder. The method we have chosen is Laguerre's method [Ralston 1965]. It is an iterative scheme similar to Newton's but with a more robust behavior. Laguerre's method iterates according to the following procedure:

$$\xi_{k+1} = \frac{\xi_k + na(\xi_k)}{a'(\xi_k) \pm \sqrt{H(\xi_k)}}$$

where n is the degree of the polynomial a, and

$$H(\xi) = (n-1)[(n-1)a'(\xi)^2 - na(\xi)a''(\xi)].$$

The sign of the square root is chosen to make the update term have smallest magnitude.

The derivation of Laguerre's method proceeds essentially the same way as does the Newton procedure except that one additional term is preserved in the power series expansion. This, in effect, approximates the function by a parabola instead of a line.

The Laguerre procedure has a number of advantages of the Newton. It is more robust. It is cubically convergent, trebling the number of digits of accuracy at each step. It converges in a single iteration for linear and quadratic equations. It naturally finds all the complex roots.

The well known Newton procedure also possesses some well known instabilities when presented with initial guesses that do not satisfy certain criteria. In contrast, for a polynomial $a(u)$ with real roots $\xi_1 < \xi_2 < \cdots < \xi_n$, the Laguerre procedure is unconditionally stable. No matter what the initial guess, it will converge to the nearest real root. Unfortunately, for complex roots the behavior of the Laguerre procedure is not so thoroughly understood although it has been found in practice to work well. Certainly our experience with solving the polynomial equations encountered during the intersection process above has shown the procedure to be remarkably stable.

Because of the cubic convergence of the Laguerre procedure, roughly five iterations are sufficient for the accuracy required for graphics.

Since most patches tend to have degree lower than the full cubic, it is an advantage to be able to naturally solve quadratics and linear equations in one step. The Laguerre algorithm does this within a single iteration.

Finally, the ability of the Laguerre algorithm to find complex roots naturally is in contrast to the Newton procedure. For a real polynomial, if the initial guess is real then all subsequent Newton iterations will be so. To obtain complex roots one must resort to a modification of Newton's algorithm such as, for example, Bairstow's.

For these reasons we have chosen the Laguerre procedure to calculate roots of the resultant polynomial. However, we consider the selection of a suitable root finding procedure to be an open question — especially when considering hardware implementation.

Once a root has been found, the polynomial is deflated by synthetically dividing by the monomial corresponding to the root. This step is exactly Horner's rule whose intermediate results form the new deflated coefficients. If the root ξ is complex and the coefficients of the polynomial are real then it is more efficient to deflate by the root and its conjugate simultaneously. This is done by synthetically dividing by the quadratic:

$$x^2 + 2\mathrm{Re}\{\xi\}x + |\xi|^2.$$

§8 Algorithm Complexity

Because the probability distribution of the degrees of the incoming bicubic patches are unknown, as well as of the number of iterations for the Laguerre procedure to converge (it is strongly dependent on the degree), we are unable to estimate the average number of operations to compute a ray-patch intersection. We will, however, estimate the worst case number of floating point operations. These estimates appear in Table 1.

What do these numbers mean? The following is a highly speculative discussion of the performance of this algorithm in the context of a full ray tracing system. We emphasize that we have as yet not constructed such a system so that the following estimates are *very* approximate. The reader is cautioned to regard this speculation with skepticism. However, by giving the reader a rough indication of the order of magnitude of computation involved, this discussion may have some small value.

Estimate that there are perhaps 1 million rays that need to be traced per frame at 30 frames per second. Assume also that the hierarchical representation technique of Rubin and Whitted culls intersection computations to on the average of one patch per ray. Since the Laguerre root finding procedure is $O(n^3)$ in the number of iterations, let us also assume that the degree of the patches is such that one tenth to one quarter the number of floating point operations is needed in the average as in the worst case. We then estimate that using this technique we would need a 200-1000 gigaFlop machine to make real time ray traced images possible.

It should be mentioned that an independent calculation performed by one of the reviewers of this paper yielded a complexity of 30 Hr/Frame on a quarter megaFlop machine (VAX-11/780), which makes it approximately as fast as existing programs. Also it puts the real time figure at 810 gigaFlops.

The state of the art for general purpose machines is roughly 200-1000 megaFlops. It is clear that in order to realize real-time ray traced images we need substantial advances both in algorithm development as well as in special purpose hardware design.

§9 Comparison with Other Algorithms

At present only one other procedure is available for computing the intersection of a ray with a patch, *viz.* Whitted's [1980].[1] We can liken the current situation in ray tracing to the suite of algorithms for rendering patches in scan-line order. There are roughly two camps: the numerical analysis camp, of which Blinn's algorithm is the sole resident; and the subdivi-

[1] I understand that Michael Potmesil of Rensselaer Polytechnic Institute has also implemented a ray tracing scheme for bicubic patches that is a hybrid of subdivision and numerical techniques. Unfortunately, I haven't yet seen it.

sion camp, populated by (another) Whitted's method as well as the Carpenter-Lane and Lane-Carpenter approach, *cf.* Blinn, *et. al.* [1980].

The new method is similar in spirit to Blinn's, especially his "slow-but-accurate" method, Blinn [1978a]. In fact, the new method may be used as a scan-line algorithm and represents a new entry in the numerical analysis camp. Our method differs from Blinn's in that we do not have the various silhouette edge trackers and maxima finders needed in his algorithm. We never need worry when contours appear and disappear since we are not using Newton's algorithm which has a critical need for a good initial guess.

Whitted's ray tracing method repeatedly subdivides patches using a necessary condition to determine upon which subpatch to recurse. This method bears a striking similarity to the Carpenter-Lane algorithm for displaying patches in scan-line order, except that subdivision must proceed along both parameters since the object is to subdivide down to a point rather than a span. The subdivision algorithm uses quite a bit of storage and converges only at a linear rate to the desired point. Thus many more iterations are needed than with the new method. On the other hand, each iteration being a Catmull subdivsion [Catmull 1974] plus bounding sphere calculation is far cheaper than the steps proposed here.

We can liken the subdivision algorithms to root finding by binary search along both co-ordinates. The method presented here calculates a more powerful iteration step. The method converges more quickly but is more expensive at each step. Which method is preferable? Certainly in the numerical analysis field, experience has shown that iterative calculation is preferable to subdivision search. Whether the same tradeoffs apply in the computer graphics context — especially when considering special purpose hardware — remains a question to be answered by a more careful and detailed comparison than we have done here.

Finally we mention an approach due to Ullner[1981] which is similar to the Lane-Carpenter algorithm. This method does not actually subdivide but rather simply evaluates at specified points, until a flatness criterion is satisfied, then it directly calculates the ray-plane intersection.

§10 Speeding up the Computation

There are several optimizations which may be carried out on the above procedure. First, when roots of the Bezout polynomial are either found to lie outside the parameter interval (usually $[0,1]$) or are found to be complex, then we may simply delete them from the list of roots and deflate the resultant accordingly. This step saves the subsequent substitution and GCD steps, which are unnecessary since these roots do not contribute to the visible portion of a patch. Second, if we shoot rays from the observer in scan-line order, we may retain one of the plane equations and move only the second. This eliminates half the algebraic curve computations in the step which calculates the resultant. Third, we may pick up a small amount of coherence by retaining the solutions to the previous round of rays from past scan lines to predict where the new intersection may be. Blinn [1978a] has found this to be a most effective heuristic, reducing the average number of iterations to less than one. If we retain the out of bounds and complex roots then we may capitalize on Blinn's optimization further, for a ray which just misses a patch actually intersects it at a pair of complex conjugate points. See figure 3.

Finally, it may be that eliminating the Laguerre procedure in favor of some form of Sturm's algorithm (which finds real roots within some interval), would yield advantages. We have not yet investigated this question.

§11 Suitability for Hardware Implementation

The lack of coherence in the ray tracing computation can be considered to be both unfortunate and an advantage. Incoherence can be an advantage when seeking to parallelize an algorithm. The opportunity for virtually unbounded parallelism is afforded by the fact the each pixel computation proceeds independently of all others. Because it uses well known numerical procedures, fast floating point pipelines — both embedded within large scientific machines and implemented as quasi-autonomous units — will without any difficulty directly accelerate the speed of image generation. The algorithm is bound almost exclusively by floating point operation speed, not by storage management, not by dataflow complexity, and not by program control complexity.

Hardware to implement this technique must perform two operations with facility. First, it must be able to swiftly calculate fixed length recurrence formulae such as Horner's rule. And second, it must be able to find the roots of polynomials very quickly and in a robust manner.

The algorithm as presented here is free of all *ad hoc* heuristics which may complicate the implementation of it in hardware. There are no silhouette edge trackers, maxima finders, etc. involved. The algorithm is either straight line code or conditional code dependent upon the degree of the polynomials. Thus the algorithm consists of only feedforward datapaths with no feedback datapaths. There is one indeterminate WHILE iteration in the root finder, which because of the excellent convergence of the Laguerre method may be unrolled to say 5 stages. Upon loop unrolling everything may be totally pipelined.

Adaitionally, the possibility of using a single unified representation for planes and patches presents itself, since aside from a penalty in data storage and a small computational overhead, the cost of solving ray-plane intersections via this method compares not unfavorably with the direct linear solution.

§12 Disadvantages and Open Problems

Because no significant ray-to-ray coherence is utilized, aside from a rather weak object coherence, the algorithm consumes a tremendous amount of computing bandwidth. Given that a typical frame requires tracing approximately 1 million rays, a real time processor would have to dispatch the computation for each ray in 33 ns. Thus, if only 10 floating operations are required per ray, real-time ray tracing requires a 300 megaFlop processor. As we have seen, we have at worst case approximately 6000 floating operations per ray.

Thus it would seem that incorporating very strong ray-to-ray coherence is a *sine qua non* for a practical ray tracing system. On the other hand, techniques such as normal perturbation destroy almost any coherence between neighboring reflected and transmitted rays within a patch. It is difficult to see how one can retain any but weak coherence when employing the various intensity and normal perturbation mapping techniques which are so vital to the convincing realism offered by parametric patches. Perhaps it is impossible to reconcile the two. It may well be that high quality images are by their nature very expensive.

Another open problem is how to properly anti-alias ray traced images. Whitted applies a Catmull subdivision to each pixel [Whitted 1980]. He also optimizes by restricting the aliasing procedure to areas which have high luminance gradients—such as edges. Even so, this form of anti-aliasing triples the cost of ray-tracing. For heavily mapped patches the penalty for anti-aliasing simply staggers the imagination. We believe that another way may be found to anti-alias ray-traced images. We are currently investigating the possibility that Hamilton's point characteristic from geometrical optics may be of some help here.

ACKNOWLEDGMENTS. I would like to thank Mike Ullner for many discussions and ideas about the ray tracing process. Also, the reviewers were helpful in making many valuable suggestions.

§13 References

APPEL, A., "Some Techniques for Shading Machine Renderings of Solids", 1968 SJCC, pp. 37-45.

BLINN, J.F., "Computer Display of Curved Surfaces" Ph.D. Thesis, U. of Utah, Computer Science Department, Salt Lake City, Utah. 1978.

BLINN, J.F., "Simulation of Wrinkled Surfaces", *Computer Graphics*, v.12, August 1978, pp.286-292.

BLINN, J.F., CARPENTER, L.C., LANE, J.M. AND WHITTED, T., "Scan Line Methods for Displaying Parametrically Defined Surfaces",*Comm. ACM,* v.23, January 1980, pp.23-34.

CATMULL, E.E., "A Subdivision Algorithm for Computer Display of Curved Surfaces" Ph.D. Thesis, U. of Utah, Computer Science Department, Salt Lake City, Utah. 1974.

LANE, J.M., AND CARPENTER, L.C., "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces", *Computer Graphics and Image Processing,* v.11, 1979, pp.290-297.

LITTLEWOOD, D.E., *A University Algebra,* Dover, 1970.

RALSTON, A., *A First Course in Numerical Analysis,* McGraw-Hill 1965.

ROTH, S.D., "Ray Casting for Modeling Solids" *Computer Graphics and Image Processing,* v.18, 1982, pp.109-144.

RUBIN, S., AND WHITTED, T.,"A Three-Dimensional Representation for Fast Rendering of Complex Scenes", *Computer Graphics,* v.14, 1980, pp.110-116.

ULLNER, M., private communication, 1981.

USPENSKY, J.V., *Theory of Equations,* McGraw-Hill, 1948.

WALKER, R.J., *Algebraic Curves,* Springer-Verlag, 1950.

WHITTED, T., "An Improved Illumination Model for Shaded Display",*Comm. ACM,* v.23, June 1980, pp.343-349.

Compute the curve coefficients	224
Compute the Bezout determinate . . .	447
Find the roots of the resultant	3988
Compute the GCD	1476
Total	6135

Table 1. Number of floating operations required to intersect a ray with a full degree (3,3) bicubic parametric patch.

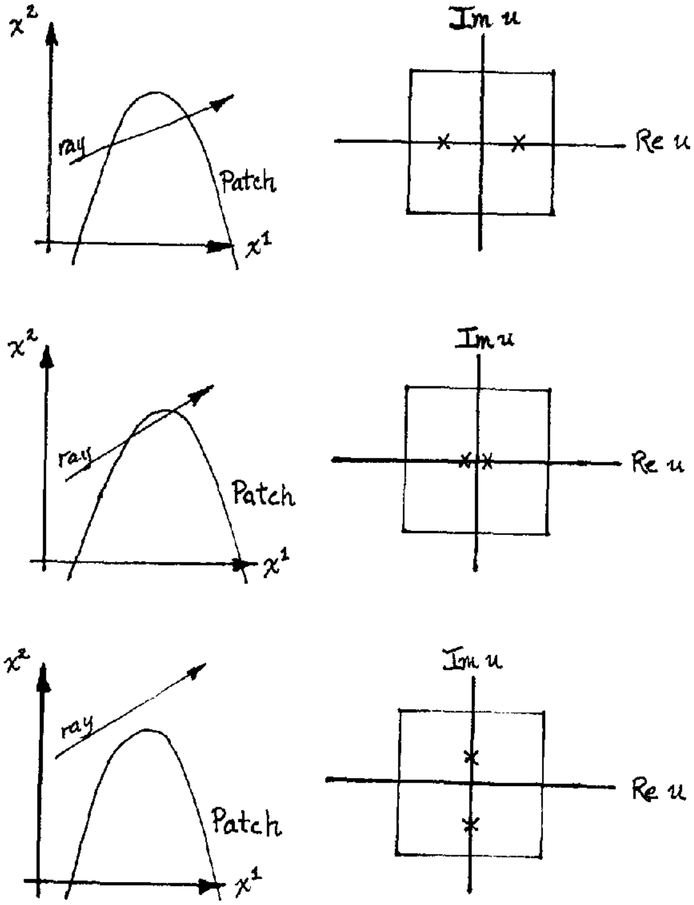
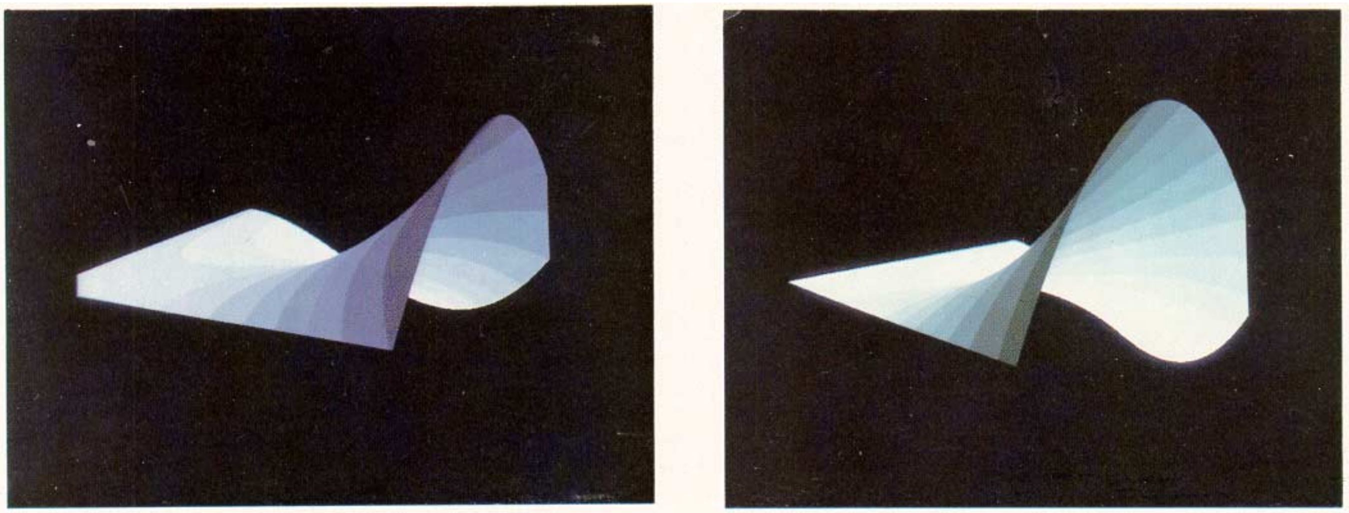


Figure 3. A ray crossing a silhouette edge. The crosses mark the parameter values at which the ray intersects the patch. As the ray crosses the silhouette edge, the intersection points become complex. (We have suppressed the v-parameter and two spatial coordinates for clarity.)



Figures 1 and 2. The contouring is an artifact of insufficient grey level resolution. These images were displayed on a 4 bit/pixel frame buffer.