グラフィクスとCAD 22－3
（1986・7・11）

パラメトリック曲面のための高速光線追跡表示

アンソニー　A．マチエフスキー
日立製作所　中央研究所

光線追跡法によるパラメトリック曲面の表示アルゴリズムは、通常長大な計算時間を要し、広範囲には適用されていない。本報告では、光線追跡法の従来アルゴリズムに、改良したニュートン法を組み合わせ、光線追跡表示において、パラメトリック曲面も能率的に計算する手段を提案する。多重反射または屈折を平面にのみ許すことによって、光線間の相関関係を結果として使用でき、インクリメンタルなアルゴリズムで処理可能である。この結果、既存のアルゴリズムに比較し、計算速度において一桁程度の向上を図れた。

Computationally Efficient Ray Tracing of Parametric Surfaces

Anthony A. Maciejewski
Hitachi Central Research Laboratory
(Kokubunji-shi, Tokyo 185, Japan)

Algorithms for ray tracing parametric surfaces are in general too computationally expensive to be widely applicable. The algorithm presented here combines well-known graphics procedures with a modified Newton iteration to provide a computationally efficient means of including parametric surfaces in a ray traced image. By allowing only planar surfaces to be reflective and/or refractive the resulting high degree of ray coherence is utilized to make the algorithm incremental and results in an order of magnitude improvement in computation speed over existing algorithms.

I. Introduction

Ray tracing is unquestionably the most powerful method to date for displaying realistic images. Whitted's paper [1] is the classic reference on the basic technique with Rogers [2] providing an excellent overview of the algorithm as well as discussions on related issues. While the realism achievable with the technique is unparallelled [3], the computation time required precludes its use for many applications. Thus the past five years have seen a number of techniques proposed in order to reduce the computational expense incurred, the majority of which is due to intersection calculations.

Bounding volumes [1], hierarchical environment descriptions [4], space subdivision [5,6], and adaptive tree depth control [7] are all useful for reducing the total number of intersection calculations. However, in order to be effective they must be combined with a computationally efficient means of finding the actual ray-surface intersection. Excluding a few exceptions [8,9,10,11] most traditional ray tracing programs are limited to polygonal or quadric surfaces for which the ray surface intersection calculations are particularly simple. The direct calculation of the intersection of a ray and a parametric surface has proven to be extremely time consuming [12]. An approach based on interval techniques [13] represents a significant decrease in computation time, however, it is still prohibitive for many applications.

The very generality which makes parametric surfaces difficult to ray trace also makes them attractive for a variety of applications. CAD/CAM modelling in particular makes wide use of parametric surface descriptions [14]. However, since rapid visual feedback is desirable in the design process, scan line display algorithms [15,16] are typically employed. Unfortunately, when objects are displayed using only local illumination information, some useful three-dimensional cues are not present. Shadows and reflections from planar surfaces can provide additional information which the CAD/CAM designer can utilize in evaluating a model. This application was the original motivation for the algorithm presented here.

II. Algorithm Overview

From the results of previous research it appears that Newton iteration is the most promising approach in obtaining a computationally efficient intersection algorithm for parametric surfaces. Unfortunately, existing algorithms [13] require a significant amount of computation time in determining if the iteration will converge before it is even applied. It will be shown that by using a modified iteration procedure [17], one can avoid these time consuming convergence tests as well as other numerical difficulties. Furthermore, if coherence is fully exploited as in the case of scan line algorithms [15], then the calculations can be made incremental with a significant increase in speed. To this end, it will be assumed that there exists a strong degree of coherence among rays. In particular, rays will be considered to be travelling in parallel to compose beams much in the same way as described by Heckbert and Hanrahan [18]. This assumption, by removing the generality of the ray-surface intersection calculation, will result in a significant increase in speed at the expense of excluding glossy and refractive patches. Since the above assumption does not effect the control flow of the standard ray tracing algorithm, the following discussion will be limited to the beam-surface intersection calculation portion of the algorithm. It should be noted that many of the techniques for reducing the computation time of a ray traced image which were discussed previously are still applicable with bounding volumes and adaptive tree depth being particularly useful.

The procedure for the beam-surface intersection calculation will now be outlined. The patch is first transformed so that the direction of the beam is parallel with the z axis. The view of a patch along the beam direction now appears as a grid with the rays being located at grid intersections. Thus the rays can be identified by their x and y coordinates in the beam coordinate system. Using a modified univariate Newton iteration, the rays which pass closest to the boundary curves are then computed and stored on a stack along with the corresponding patch parameters. After all of the boundary curves have been completed, the algorithm begins processing the stack. Until the stack is empty, the following procedure is performed. First, a ray-patch intersection is popped from the stack and checked against the beam z buffer to see if it is the closest intersection. Next, the modified bivariate Newton iteration is used to compute the intersection of the patch with the current rays four nearest neighbors. If these ray patch intersections have not already been processed then they are placed on the stack.

The main portion of the algorithm amounts to a variant of the seed fill algorithm [2] using the boundary curve ray intersections as seeds. A check on the patch parameters gives a simple test for determining whether the intersection is inside or outside of the patch. By exploiting the coherence of adjacent rays, the intersection calculation is made incremental which results in a significant decrease in the amount of computation time required. The required modification to the iteration procedures will be discussed in the following section.

III. Implementation

This section considers some of the details of the implementation of the above algorithm. For the sake of illustration only the specific example of a bicubic surface will be considered, although this by no means implies any restriction on the generality of the method. A general bicubic surface can be described by the equation

$$S(u,v) = [u^3\ u^2\ u\ 1]\ M\ [v^3\ v^2\ v\ 1]^T \qquad (1)$$

where S(u,v) is a three dimensional point with components X(u,v), Y(u,v), and Z(u,v), M is a matrix of constant coefficients, and the parameters u and v are restricted to be within the interval [0 1]. In the discussion that follows it will be assumed that the transformation to beam coordinates has been applied to the patch description and that X, Y, and Z are the patch coordinates with respect to the beam coordinate system. Therefore, the next step is the computation of the ray-boundary curve intersections.

The four boundary curves of the patch are obtained by substituting u=0, u=1, v=0, and v=1 into the patch description given by eq. 1. All of the resulting curves are univariate and can be expressed in the form

$$C(t) = N_3\ t^3 + N_2\ t^2 + N_1\ t + N_0 \qquad (2)$$

where once again C(t) is a three dimensional point with components X(t), Y(t), and Z(t) with the parameter t in the interval [0 1]. Since the patch has been transformed into beam coordinates, only the x and y components

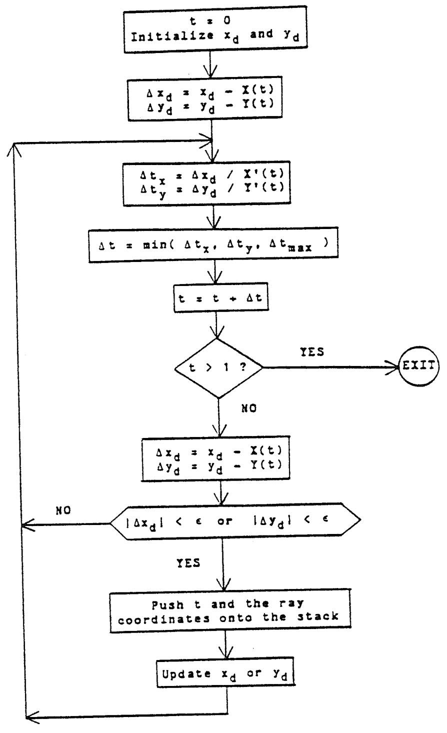
]
p
r
c
R
t
r
w
c
f
ye
pr
ex
du
en
sul
cor
tot
How
be
mea
int
[8,
pro
sur
inte
sim
inte
surf
cons
inte
sign
howe
appl
1
param
also
appli
parti
surfa
rapid

of the curve need to be considered. The algorithm incrementally advances along the curve computing the rays which pass closest to it. A flowchart describing the procedure is given in fig. 1 with fig. 2 illustrating the physical significance of the variables involved. The procedure is initialized by setting t=0 and determining Δx_d and Δy_d which are the displacements to the nearest ray grid line in the x and y direction, respectively. The rate of change of the x and y components with respect to t, denoted by X' and Y' are easily computed by differentiating eq. 2 to obtain

$$C' = 3\ N_3\ t^2 + 2\ N_2\ t + N_1 \qquad (3)$$

Univariate Newton iteration is used to determine the t parameter satisfying the desired x and y positions, x_d and y_d, which represent the curves intersection with the ray grid lines. The minimum change in t is chosen at each iteration to insure that no grid intersections are missed. A maximum change in t, Δt_{max} is included in the iteration scheme to avoid any difficulty when both X' and Y' go to zero, i.e. the boundary curve is proceeding parallel to the z axis. The procedure continues to output all of the rays which pass nearest the curve until t exceeds 1 at which time it halts. Additional seed values may be placed on the stack by including u and v parameter seed values in the patch description and then iterating to find the closest ray intersection.

This completes the stack initialization and the program now begins processing the information on the stack. As ray-patch intersections are popped from the stack, they are first checked against the beam z buffer to determine if they are visible. The beam z buffer is of conventional design being composed of a two dimensional array where the



t = 0
Initialize x_d and y_d
$\Delta x_d = x_d - X(t)$
$\Delta y_d = y_d - Y(t)$
$\Delta t_x = \Delta x_d / X'(t)$
$\Delta t_y = \Delta y_d / Y'(t)$
$\Delta t = \min(\Delta t_x, \Delta t_y, \Delta t_{max})$
$t = t + \Delta t$
t > 1 ?
YES
EXIT
NO
$\Delta x_d = x_d - X(t)$
$\Delta y_d = y_d - Y(t)$
NO
$|\Delta x_d| < \epsilon$ or $|\Delta y_d| < \epsilon$
YES
Push t and the ray coordinates onto the stack
Update x_d or y_d

Fig. 1. Flowchart of the modified univariate Newton iteration procedure used to find which rays pass closest to the patch boundary.

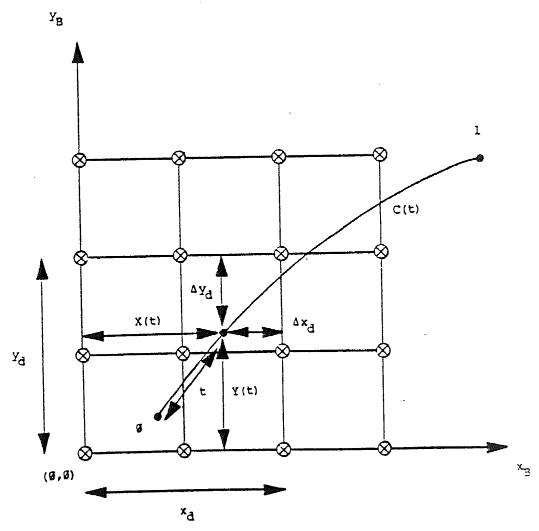


Fig. 2. Physical significance of the variables used in the modified univariate Newton iteration procedure.

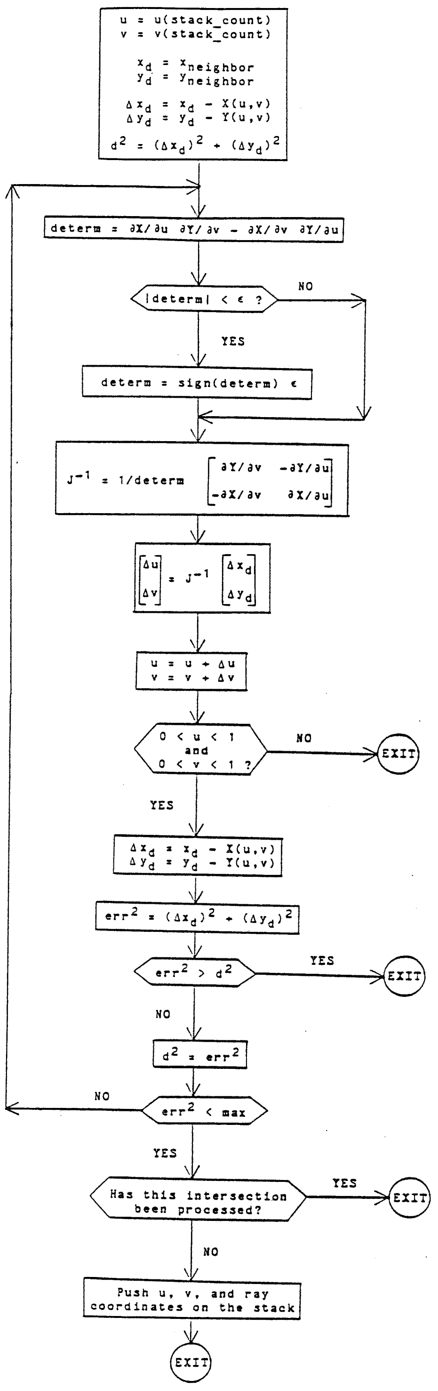
address is given by the x and y coordinates of the ray in question. The minimum contents of the buffer are the u and v parameters of the ray patch intersection since position and surface normal information can be easily computed. The z value of the intersection may also be stored in order to speed up the depth check at the expense of additional memory storage.

After checking and possibly updating the z buffer, the patch intersections of this rays four nearest neighbors, if any exist, need to be calculated. The first step is to simply compute the neighbor rays x and y coordinates and determine if they lie within the boundary of the beam. If they do then iteration starts from the current intersection to find the neighboring intersection. A flowchart of the modified bivariate Newton iteration procedure is given in fig. 3.

The procedure begins by setting the u and v parameters of the patch to those that were popped from the stack. The desired x and y positions, denoted x_d and y_d, are then set to the coordinates of the popped rays nearest neighbor. The desired x and y displacements from the current patch position, denoted Δx_d and Δy_d, are then obtained by subtracting the desired position from the current position. The squared length of the desired displacement is then stored in order to measure the progress of the Newton iteration scheme. This completes the initialization and the procedure is now ready to start the actual iteration.

The rate of change of the patch parameters u and v is related to the rate of change in x and y through the equation

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = J \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} \qquad (4)$$



u = u(stack_count)
v = v(stack_count)
$x_d = x_{neighbor}$
$y_d = y_{neighbor}$
$\Delta x_d = x_d - X(u,v)$
$\Delta y_d = y_d - Y(u,v)$
$d^2 = (\Delta x_d)^2 + (\Delta y_d)^2$
determ = $\partial X/\partial u\ \partial Y/\partial v - \partial X/\partial v\ \partial Y/\partial u$
$|determ| < \epsilon$?
NO
YES
determ = sign(determ) ϵ
J^{-1} = 1/determ $\begin{bmatrix} \partial Y/\partial v & -\partial Y/\partial u \\ -\partial X/\partial v & \partial X/\partial u \end{bmatrix}$
$\begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = J^{-1} \begin{bmatrix} \Delta x_d \\ \Delta y_d \end{bmatrix}$
$u = u + \Delta u$
$v = v + \Delta v$
$0 < u < 1$ and $0 < v < 1$?
NO
EXIT
YES
$\Delta x_d = x_d - X(u,v)$
$\Delta y_d = y_d - Y(u,v)$
$err^2 = (\Delta x_d)^2 + (\Delta y_d)^2$
$err^2 > d^2$
YES
EXIT
NO
$d^2 = err^2$
NO
$err^2 < max$
YES
Has this intersection been processed?
YES
EXIT
NO
Push u, v, and ray coordinates on the stack
EXIT

Fig. 3. Flowchart of the modified bivariate Newton iteration procedure used to compute the neighboring ray-patch intersections.

where J denotes the Jacobian matrix given by

$$J = \begin{bmatrix} \partial X/\partial u & \partial X/\partial v \\ \partial Y/\partial u & \partial Y/\partial v \end{bmatrix} \quad (5)$$

If it is assumed that a linear approximation is valid, then the required change in the patch parameters which will result in the neighboring ray intersection is given by

$$\begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = J^{-1} \begin{bmatrix} \Delta x_d \\ \Delta y_d \end{bmatrix} \quad (6)$$

Since the Jacobian is only a 2 x 2 matrix, its inverse is easily computed using

$$J^{-1} = \begin{bmatrix} \partial Y/\partial v & -\partial Y/\partial u \\ -\partial X/\partial v & \partial X/\partial u \end{bmatrix} / \text{(det.)} \quad (7)$$

provided that J is of full rank (i.e. the determinant is not equal to zero). The required partial derivatives are easily obtained by differentiating eq. (1) with respect to u and v. This results in

$$\partial S/\partial u = [3u^2 \; 2u \; 1 \; 0] \; M \; [v^3 \; v^2 \; v \; 1] \quad (8)$$
$$\partial S/\partial v = [u^3 \; u^2 \; u \; 1] \; M \; [3v^2 \; 2v \; 1 \; 0] \quad (9)$$

The difficulties usually cited when discussing Newton iteration arise when J is ill-conditioned. This occurs when J is approaching a singularity and is associated with the determinant approaching zero. Physically this will occur in the neighborhood of a silhouette edge which is analagous to the boundary curve proceeding parallel to the z axis in the univariate case discussed earlier. The method of dealing with this case is also similar to that used earlier. One can consider the solution for the vector $[\Delta u \; \Delta v]^T$ to be composed of two parts, one which specifies a direction (although not a unit vector) and the other being a scale factor. This is clearly illustrated by substituting eq. (7) into eq. (6) to obtain

$$\begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \text{(1/det.)} \begin{bmatrix} \partial Y/\partial v \, \Delta x_d - \partial Y/\partial u \, \Delta y_d \\ \partial X/\partial u \, \Delta y_d - \partial X/\partial v \, \Delta x_d \end{bmatrix} \quad (10)$$

Thus the direction of the vector $[\Delta u \; \Delta v]^T$ is specified by the elements of the Jacobian and the desired x and y displacements. The only effect of the determinant is to appropriately scale the vector specifying direction. Since the procedure is iterating toward a solution, it is concerned with always improving its position and not necessarily getting there in one step. This is particularly true when the exact solution of eq. (6) results in a solution which represents a large change since this generally means that the linear approximation was a poor one. Therefore, the algorithm puts a threshold on the value of the determinant to limit the maximum change in the patch parameters u and v.

Once the vector $[\Delta u \; \Delta v]^T$ is computed, the patch parameters are updated and checked to see if they still lie within the patch boundary. If they do then the x and y coordinates of the current patch position, X(u,v) and Y(u,v), are computed along with an updated version of the required changes in x and y to arrive at the ray intersection. At this point an error term is computed and compared to the previous error term. If there is not an improvement towards the desired solution then the procedure assumes that no solution exists and it terminates. This easily handles the case where one ray just intersects a silhouette edge but its neighbor misses. The error is then checked against a maximum threshold to determine if another iteration is necessary. If the solution is sufficiently close then the new ray patch intersection is checked to see if it has been previously processed. If not then it is pushed on the stack and the procedure terminates. At this point the algorithm moves to the rays next nearest neighbor. If all neighbors have been processed then the next ray-patch intersection is popped from the stack.

IV. Results

The resulting images for various surfaces computed using the above algorithm are presented in figures 4 thru 8 with timing data given in table 1. All computations were done on a Hitachi M200H computer equipped with a Ramtek display. All images are computed to a resolution of 1024 x 1024 using the illumination model presented in [1]. Due to insufficient color resolution (256 colors) dithering is used to improve the effective number of gray scales. Fig. 4 is a single patch similar to one presented in [13] which was generated to provide a timing comparision. After normalization of image resolution and CPU speed, the current algorithm represents an order of magnitude decrease in computation time. Fig. 5 is an example of a patch with a rather complex silhouette edge which was generated to illustrate the robustness of the algorithm when dealing with regions where a standard Newton iteration scheme would be unstable. Fig. 6 illustrates the importance of transparency and shadows in determing the three-dimensional characteristics of a surface from an otherwise ambiguous image (the vertical ridges are an artifact of the dithering process). Fig. 7 consists of 43 bicubic patches including one reflective planar patch. A comparison of the computation time for fig. 7 with that of the single patch images illustrates an important advantage of the algorithm. The computation time for an image is not proportional to the number of objects in the scene as with many ray tracing algorithms, but is a function of the cross-sectional area of the actual beam-patch intersection. Fig. 8, which is composed of only 11 bicubic patches, was generated to illustrate the flexibility of modelling with parametric surfaces as well as to provide further evidence for the preceding statement (compare the computation times for fig. 6 with fig. 8).

Fig. 4. A simple patch used for timing comparisons.

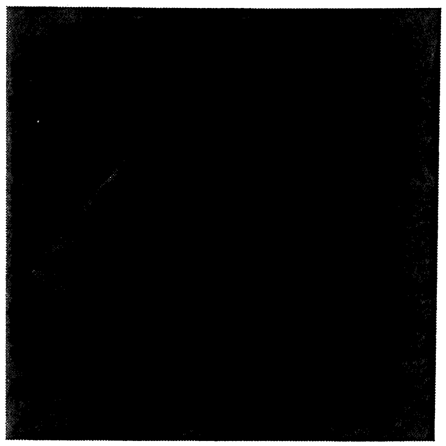


Fig. 5. A patch with a complex silhouette edge.

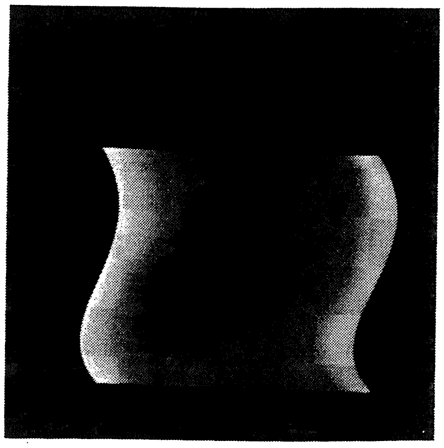


Fig. 6. A transparent patch over a checkerboard.

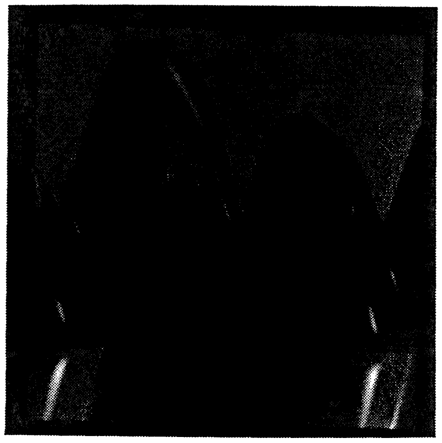


Fig. 7. A safe landing on a frozen lake in front of plastic hills.

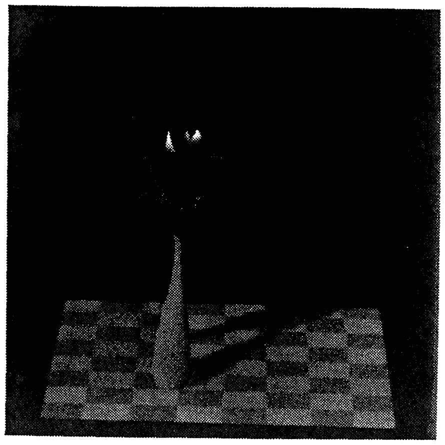


Fig. 8. Still life.

Table 1. CPU Time Requirements

Figure Number	Number of Patches	CPU Time (sec.)
4	1	187
5	1	184
6	2	1326
7	43	1231
8	11	313

V. Discussion

This section discusses some of the capabilities and restrictions inherent to the algorithm presented above. First, the algorithm is useful because it is a general algorithm which can be used for a variety of parametric surfaces. Since there is no approximation involved with modelling the surface, one can obtain a precise image of the object. In addition, the algorithm is comparatively simple, thus making it easy to implement and accessible to a variety of users.

A second advantage of the algorithm is that, unlike most ray tracing procedures, it is incremental. In this respect it is much like Blinn's scan line algorithm for displaying curved surfaces [15]. However, unlike scan line algorithms, the algorithm presented here is capable of utilizing a global illumination model for more realistic images. In addition, there is no need for edge tracking (silhouette or otherwise) or heuristic procedures. The modified iteration scheme presented here smoothly handles all cases, including those that are singular or ill-conditioned.

An additional advantage of the algorithm presented here is that, again unlike most ray tracing procedures, it has excellent information for implementing antialiasing. Some of this information stems from the fact that three-dimensional beams are being traced instead of one-dimensional rays as discussed in [18]. Furthermore, due to the modified iteration procedure, when a silhouette edge lies in between two ray, at the point where the procedure exists due to no improvement in the iteration, it contains information on exactly how close the patch was to the ray in question.

On the negative side, the algorithm requires a much larger amount of memory storage capability than other algorithms. Since an array of rays is being traced at one time, all of the related data must be stored. For beams with a large cross-sectional area this can present a problem. In particular, for the initial viewpoint beam there are 1024 x 1024 rays and the related data structures which must maintain the patch id and u and v parameters of the surface intersections. Thus in the current implementation, beam areas are restricted to an area of 10,000 rays with larger beams being subdivided. Using this scheme the current program uses a maximum of 4 M bytes of memory.

In addition to the sizable memory requirements, the exclusion of curved reflective and refractive patches from the object database restricts the type of images which can be displayed. If reflection or refraction from non-planar surfaces is essential then other techniques must be employed.

VI. Conclusion

The algorithm presented here is based on exploiting object coherence together with assumptions which guarantee a high degree of ray coherence in order to utilize incremental calculations. Standard graphics procedures and a modified Newton iteration scheme are combined to produce a reliable and efficient means of computing the intersection of rays with an arbitrary parametric surface. Thus images using precise surface representations together with the realism of global illumination models are obtained at a reasonable computational expense.

References

1. T. Whitted, "An Improved Illumination Model for Shaded Display," Comm. ACM, Vol. 23, No. 6, June 1980, pp. 343-349.
2. D. F. Rogers, Procedural Elements for Computer Graphics, McGraw-Hill, New York, 1985.
3. R. L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," Computer Graphics, Vol. 18, No. 3, 1984, pp. 137-145.
4. H. Weghorst, G. Hooper, and D. P. Greenberg, "Improved Computational Methods for Ray Tracing," ACM Trans. Graphics, Vol. 3, No. 1, Jan. 1984, pp. 52-69.
5. A. S. Glassner, "Space Subdivision for Fast Ray Tracing," IEEE Computer Graphics and Applications, Vol. 4, No. 10, Oct. 1984, pp. 15-22.
6. A. Fujimoto and K. Iwata, "Accelerated Ray Tracing," Proc. Computer Graphics Tokyo '85, April 23-26, 1985.
7. R. A. Hall and D. P. Greenberg, "A Testbed for Realistic Image Synthesis," IEEE Computer Graphics and Applications, Vol. 3, No. 8, Nov. 1983, pp. 10-20.
8. J. T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," Computer Graphics, Vol. 17, No. 3, 1983, pp. 91-102.
9. J. T. Kajiya, "Ray Tracing Volume Densities," Computer Graphics, Vol. 18, No. 3, 1984, pp. 165-174.
10. P. Hanrahan, "Ray Tracing Algebraic Surfaces," Computer Graphics, Vol. 17, No. 3, 1983, pp. 83-90.
11. T. W. Sederberg and D. C. Anderson, "Ray Tracing of Steiner Patches," Computer Graphics, Vol. 18, No. 3, 1984, pp. 159-164.
12. J. T. Kajiya, "Ray Tracing Parametric Patches," Computer Graphics, Vol. 16, No. 3, 1982, pp. 245-254.
13. D. L. Toth, "On Ray Tracing Parametric Surfaces," Computer Graphics, Vol. 19, No. 3, 1985, pp. 171-179.
14. A. Yajima, H. Jonishi, J Tsuda and N. Osada, "MDM-I: AComputerAidedMold Design and Manufacturing System," Proc. Graphics Interface '82, Toronto, May 17-21, 1982.
15. J. M. Lane, L. C. Carpenter, T. Whitted, and J. F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," Comm. ACM, Vol. 23, No. 1, Jan. 1980, pp. 23-34.
16. D. Schweitzer and E. S. Cobb, "Scanline Rendering of Parametric Surfaces," Computer Graphics, Vol. 16, No. 3, 1982, pp. 265-271.
17. P. Deuflhard, "A Modified Newton Method for the Solution of Ill-Conditioned Systems of Nonlinear Equations with Applications to Multiple Shooting," Numerical Math., Vol. 22, 1974, pp. 289-315.
18. P. S. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects," Computer Graphics, Vol. 18, No. 3, 1984, pp. 119-127.