JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 18, 541-561 (2001)

An Efficient and Stable Ray Tracing Algorithm for Parametric Surfaces

SHYUE-WU WANG*, ZEN-CHUNG SHIH*
AND RUEI-CHUAN CHANG*+
*Department of Computer and Information Science
National Chiao Tung University
Hsinchu, 300 Taiwan
+Institute of Information Science
Academia Sinica
Taipei, 115 Taiwan

In this paper, we propose an efficient and stable algorithm for finding the ray-surface intersections. Newton's method and Bézier clipping are adapted to form the core of our algorithm. Ray coherence is used to find starting points for Newton iteration. We introduce an *obstruction detection technique* to verify whether an intersection point found by using Newton's method is the closest one. When Newton's method fails to achieve convergence, we use Bézier clipping as the substitution to find the intersection points. This combination achieves significant improvement in tracing primary rays. A similar approach also successfully improves the performance in tracing secondary rays.

Keywords: ray tracing, Newton's method, Bézier clipping, parametric surfaces, ray coherence, Bézier surface

1. INTRODUCTION

Ray tracing is one of the most important techniques for realistic image generation. It is capable of producing very impressive images for highly specular scenes. The kernel of the ray tracing algorithm is an intersection finding routine for computing the ray-surface intersection points. Whitted [1] estimated that a ray tracing algorithm will spend up to 95% of its computation time on intersection tests for scenes of moderate complexity. It takes plenty of time during the whole process, efficient intersection finding is one of the greatest challenges in ray tracing.

Newton's method is a popular numerical method because of its efficiency and convenience in implementation. The critical task when using Newton's method is finding a good initial point. Toth [2] proposed an elegant theoretical results to locate the initial points. Lischinski and Gonczarowski [3] also proposed an improved technique based on Toth's results [2]. Other researchers [4-10] subdivided surfaces into *flat* patches and located the initial points from the intersections of rays and *tighter* bounding volumes that enclosed the patches. However, the primary shortcoming of these methods is that they do not use the ray coherence property finding the ray-surface intersection points. Taking the intersection point of a previous adjacent ray as the initial point, Joy and Bhetanabhotla

Received December 6, 1999; revised May 22, 2000; accepted November 16, 2000.
Communicated by Tomoyuki Nishita.

[11] concluded that the computation cost can be reduced by taking the ray coherence property into consideration. Unfortunately, native utilization of ray coherence may lead to an incorrect closest intersection point [3, 11]. We call this problem as the *incorrect solution problem*. A very interesting but difficult phenomenon induced from this issue is that how can we verify whether an intersection point is the closest one or not without checking every single one of the ray-surface intersection points.

The basic idea of subdivision methods is based on the prune-and-search technique, surface subdivision, and convex hull property. These methods [1, 12, 13] are stable but slow in calculating intersections. Bézier clipping is a well-known subdivision method which was proposed by Nishita et al. [13] and has been applied in many topics [14, 15]. The novel achievement of this method is that the subdivision process can be performed in a 2D space instead of a 3D space. However, the main drawback of subdivision methods is that ray coherence property cannot be used to assist calculation of ray-surface intersection points. Campagna et al. [16] recently pointed out that the use of Bézier clipping might lead to incorrect intersection points, so they have proposed a modification scheme to solve this problem.

Kajiya [17] proposed an algebraic algorithm that reduces the ray-surface intersection problem to that of finding the exact roots of a high-degree polynomial. Manocha and Demmel [18, 19] transformed the problem of finding ray-surface intersections into the problem of finding the eigenvalues of high-order matrices. Both of these methods suffer from high computation cost in dealing with high order matrices. The computation cost of finding eigenvalues for such a matrix is quite high.

Since the performance of ray tracing parametric surfaces is dominated by the way of calculating the ray-surface intersections, various methods have been proposed to improve performance. The first one involves changing the computation base. Transforming the computation base from a 3D space into a 2D space can effectively reduce the computation cost to a certain degree. As for Newton's method, the number of arithmetic operations can be reduced by 33%. As pointed out by Nishita et al. [13], the number of arithmetic operations needed to subdivide a non-rational patch is reduced by 33% and that needed for a rational patch is reduced by 25%. The other consideration is that the method use to find ray-surface intersections should be as *simple* as possible. Numerical methods, especially Newton's method, could be considered as the simplest ones for finding intersections. However, two problems remain to be solved.

The first one is selection of suitable initial points. Ray coherence is often an effective method, but it leads to the *incorrect solution problem*. The second one is that Newton's method is somehow unstable on finding the intersection point. In our practical experiments, it unveils that Newton's method fails to converge even if the initial point that we take is quite near to the intersection point. That is, we may miss some intersection points [7]. Fortunately, subdivision methods, especially Bézier clipping, can solve this problem.

Based on the above considerations, we propose an efficient algorithm based on combining Bézier clipping and Newton's method. To verify whether an intersection point is the closest one, we propose an obstruction detection technique. A regular grid with uniform space subdivision [20] is constructed for each surface. The regular grid is an axis-aligned parallelepiped, which encloses the surface and contains all of the patches obtained by subdividing the surface. We construct the oriented-slab boxes [6] for each patch. Once an intersection point using Newton's method has been found, a ray is traced from the intersection point toward the traced ray's origin through the tested surface's regular grid. According to the intersections between the ray and the oriented-slab boxes of the intersected patches, we can either verify whether an intersection point is the closest one or we can use Bézier clipping to find the true closest intersection point.

An overview of the proposed algorithm is as follows. For a surface that will be tested along a scan line, Bézier clipping is used to locate the first ray (along the scan line) that intersects the surface and to find the closest intersection point between them. Then, Newton's method with ray-to-ray coherence is used to find the other intersection points between the surface and the remaining rays along the scan line. That is, we take the intersection point found by previous ray as the initial point. After an intersection point is found in this way, we use the obstruction detection technique to verify whether this point is the closest one or not. When Newton's method fails to converge, Bézier clipping is applied to find the intersection points instead.

In addition, the regular grid can also be used to improve efficiency in tracing secondary (reflected, refracted and shadow) rays. Since the origin of a secondary ray lies on a surface, the computations for finding the intersections between them should be performed in order to detect self-shadowing or self-reflection. However, if the secondary ray does not intersect the surface except at the origin of the ray, it is not necessary to perform this computation. In this paper, we present a method based on the regular grids to avoid this intersection computation. The idea is as follows. When tracing a secondary ray, we first trace this ray through the regular grid. By examining the intersection conditions between the secondary ray and the intersected patches, we can verify whether or not the secondary ray along the tracing direction will intersect the surface.

Experimental results show that the performance of our algorithm is 2 to 2.8 times faster than that of Bézier clipping [13] on tracing primary rays. The method we use to select the initial point for Newton's method can result in rapid convergence. Furthermore, by reducing the computation needed for secondary rays, our algorithm can reduce the total rendering time by 20% to 50%.

The rest of this paper is organized as follows. In section 2, we review the projection process, Bézier clipping, and Newton's method. The obstruction detection technique used to solve the incorrect solution problem is presented in detail in section 3. Our hybrid algorithm is described in section 4. The improvement achieved in tracing secondary rays is presented in section 5, and experimental results are given in section 6. Finally, conclusions are shown in section 7.

2. PROJECTION, BÉZIER CLIPPING AND NEWTON'S METHOD

Transforming the computation from 3D space to 2D space is important technique to reduce the computation cost of finding ray-surface intersection points. Bézier clipping [13] is a well-known method based on this technique. In this paper, we use Newton's method to solve a two-dimensional nonlinear system reduced by means of the projection process. In this section, we will first introduce the projection process and then review the Bézier clipping approach. Finally, we will present our method, which applies Newton's method to solve the two-dimensional nonlinear system.

2.1 Transformation of Ray-Surface Intersections

Let $P(u, v)$ be a rational Bézier surface [21] in three-dimensional Euclidean space, defined as

$$P(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)w_{ij}C_{ij}}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)w_{ij}}, \tag{1}$$

where $B_i^n(s) = \binom{n}{i}(1-u)^{n-i}u^i$ denotes the Bernstein basis functions and $C_{ij} = (x_{ij}, y_{ij}, z_{ij})$, for $i = 0,..., n$ and $j = 0 ,..., m$, represents Bézier control points with corresponding weights w_{ij}. A ray L is defined by the intersection of two orthogonal planes, P_1 and P_2, and its implicit equation is as follows:

$$L\colon a_k x + b_k y + c_k z + e_k = 0, \tag{2}$$

where $a_k^2 + b_k^2 + c_k^2 = 1$, $k = 1, 2$. In order to find the intersections between ray L and $P(u, v)$, we substitute Eq. (1) into Eq. (2) and clear the denominator. The resulting equation is written as follows:

$$\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)d_{ij}^k = 0, \qquad k = 1,2, \tag{3}$$

where

$$d_{ij}^k = w_{ij}(a_k x_{ij} + b_k y_{ij} + c_k z_{ij} + e_k). \tag{4}$$

Since $a_kx_{ij} + b_ky_{ij} + c_kz_{ij} + e_k$ is the distance from the control point C_{ij} to the plane P_k, we can now project the rational Bézier surface into a two-dimensional space by taking the projected control point as $p_{ij} = (\overline{x}_{ij}, \overline{y}_{ij}) = (d_{ij}^1, d_{ij}^2)$. Note that the distances are signed values. From Eq. (3) and Eq. (4), the projected surface is defined as follows:

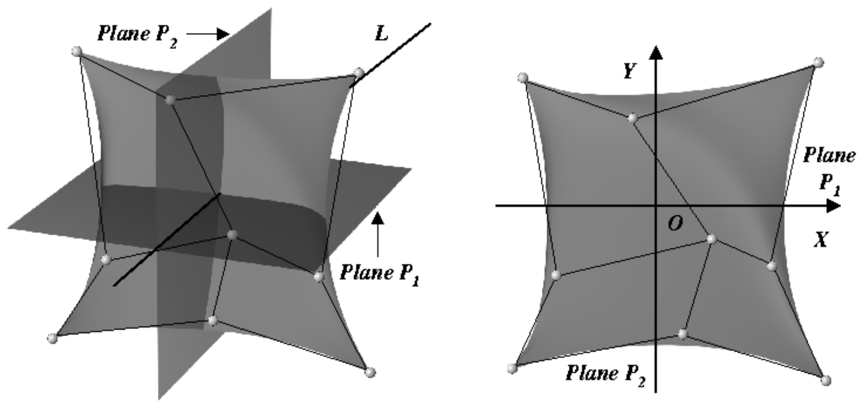
$$\overline{P}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)p_{ij}. \tag{5}$$

In this projection, the two-dimensional coordinate system is formed by these two orthogonal planes, where plane P_1 corresponding to the $\overline{X}$ axis and plane P_2 corresponding to the $\overline{Y}$ axis. The origin O of the coordinate system is corresponding to the ray L. For example, a three-dimensional Bézier surface with 9 control points and a ray L defined by planes P_1 and P_2 are shown in Fig. 1(a). The corresponding projected surface is shown in Fig. 1(b).

By Eq. (3) and Eq. (5), we can transform the ray-surface intersection problem into the following two-dimensional nonlinear system:

$$\overline{P}(u,v) = 0. \tag{6}$$

That is, we need to find all of the (u, v) pairs such that $\overline{P}(u,v) = 0$, where $0 \le u, v \le 1$.



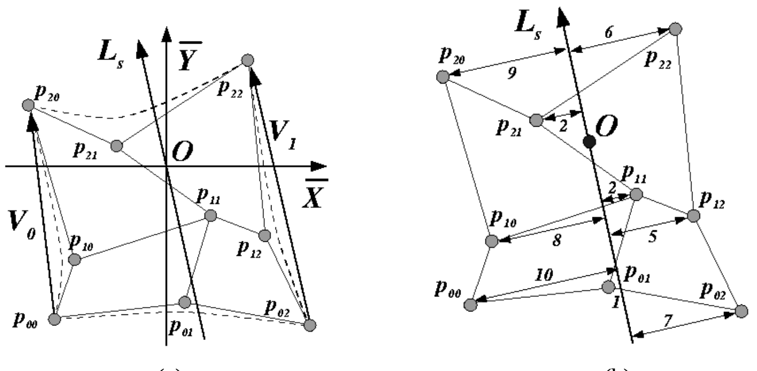
(a) (b)

Fig. 1. The projection of a Bézier surface. (a) shows a surface and a ray in three-dimensional space. (b) shows the corresponding projected surface.

2.2 Bézier Clipping

The idea behind Bézier clipping is to subdivide the parameter space of the non-linear system (Eq. (6)) into regions that may contain solutions and regions that do not contain solutions. All of the intersection points can be found by recursively applying the process to the regions that may contain the solutions.

The first step in solving the problem is to define a line L_s that extends through the origin O parallel to the vector $V_0 + V_1$. The vectors V_0 and V_1 are constructed using the boundary control points of the projected surface $\overline{P}(u,v)$. For example, Fig. 2(a) illustrates these two vectors with the projected surface in Fig. 1(b). Let L_s be defined as $ax + by + c = 0$, where $a^2 + b^2 = 1$.



(a) (b)

Fig. 2. (a) The projected surface with vector V_0, V_1, and line L_s. (b) Control point distances.

The projected surface $\overline{P}(u,v)$ can now be transformed into an *explicit surface* [13] by replacing the control points p_{ij} with the corresponding distance from p_{ij} to L_s (see Fig. 2(b)). Since this explicit surface is also a Bézier surface, the convex hull of the explicit surface bounds the explicit surface and the intersections between the convex hull and the parametric space can be found. Hence, the parametric space can be classified into regions that may contain the solutions and regions without solution. Then, the *de Casteljau subdivision algorithm* [21] is applied to clip away the regions without solution. This is the subdivision process of Bézier clipping. All of the solutions of the non-linear system (Eq. (6)) can be found by recursively applying the process to the regions that may contain the solutions.

2.3 Newton's Method

The ray-surface intersection problem, as in the above discussion, can be rewritten as a two-dimensional nonlinear system (Eq. (6)). We apply Newton's method to solve this two-dimensional nonlinear system instead of a three-dimensional nonlinear system, like Toth [2], Lischinski and Gonczarowski [3], and Barth and Stürzlinger [7].

Let $f(x) = 0$ be a nonlinear system in n-dimensional Euclidean space (in practical, $n = 2$ or 3) and Y be a nonsingular matrix; the Newton iteration is presented using

$$x_{k+1} = x_k - Yf(x_k). \tag{7}$$

The vector $-Yf(x_k)$ denotes the *Newton step* and Y is the inverse Jacobian matrix of f at x_k. For example, consider $n = 2$ and apply Newton's method to solve Eq. (6). Let $f(X) = (f_1(X), f_2(X)) = \overline{P}(u, v) = 0,$ where $f_k(X) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) d_{ij}^k, k = 1,2,$ and $X = (u, v)$. The Jacobian matrix $J(X)$ for f can be written as:

$$J(X) = \begin{bmatrix} \partial f_1(u,v)/\partial_u & \partial f_1(u,v)/\partial_v \\ \partial f_2(u,v)/\partial_u & \partial f_2(u,v)/\partial_v \end{bmatrix}, \tag{8}$$

and $Y = J^{-1}(X)$. Then the Newton iteration can be rewritten as:

$$\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} u_k \\ v_k \end{bmatrix} - \begin{bmatrix} \partial f_1(u,v)/\partial_u & \partial f_1(u,v)/\partial_v \\ \partial f_2(u,v)/\partial_u & \partial f_2(u,v)/\partial_v \end{bmatrix}^{-1} \begin{bmatrix} f_1(u_k, v_k) \\ f_2(u_k, v_k) \end{bmatrix}. \tag{9}$$

Newton's method yields quadratic convergence while the matrix Y is updated at each iteration. If the matrix Y is fixed during Newton's iterations, then Newton's method is referred to as *simple* Newton's method [22] with only linear convergence.

3. INCORRECT SOLUTION PROBLEM

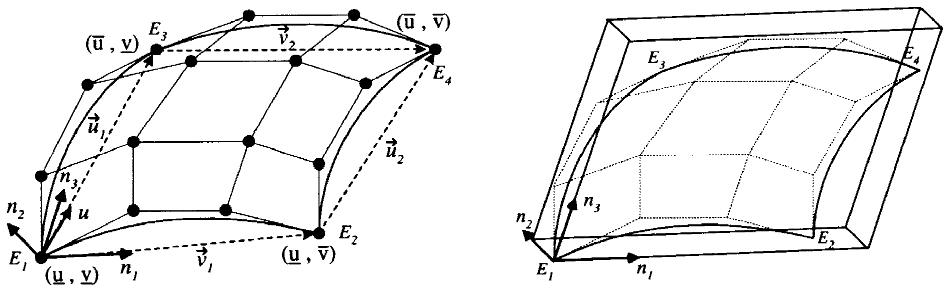
As pointed out by Joy and Bhetanabhotla [11], we can use ray coherence to accelerate the speed of locating a suitable initial point for Newton's method. Unfortunately, we will face the incorrect solution problem during applying this property. Taking the entire view of the incorrect solution problem into our consideration very carefully, we find out that the key to solving this problem is to determine whether there do exist obstructions (intersection points) between the intersection point found by Newton's method and the origin of the traced ray. If we use any known algorithms to calculate the ray-surface intersections, the computation time can be no more be reduced.

To solve this problem efficiently, we use regular grids to enclose the *flat* patches obtained by subdividing the surfaces and the construct the oriented-slab boxes [6] for each patch. By examining the intersections of a ray and the oriented-slab boxes of the tested patches, we can either verify whether an intersection point is the closest one or use Bézier clipping to find out the exact closest intersection point. This is the core idea behind the obstruction detection technique. In the following, we shall first introduce the property of intersections between a ray and an oriented-slab box. Secondly, we will discuss the criterion of deciding whether there exists one (or more) intersection point(s) between a ray and a patch. Finally, we will discuss the details of the obstruction detection technique.

3.1 Intersection With an Oriented-Slab Box

The oriented-slab box was proposed by Yen et al. [6]. The main feature of the oriented-slab box is a very tighter bounding volume for flat patches. This property is very informative and necessary for us to determine the intersection conditions between a ray and a patch. This is why we take the oriented-slab boxes as the bounding volumes.

Let a Bézier surface be adaptively subdivided into flat patches. Each patch belongs to two parameter intervals $[\underline{u}, \overline{u}]\times[\underline{v}, \overline{v}]$. The oriented-slab box is constructed by calculating the min-max bounding volume along each of the three orthogonal slab normals n_1, n_2, and n_3, as shown in Fig. 3.


$(\overline{u}, \underline{v})$
E_3
$\vec{v}_2$
$(\overline{u}, \overline{v})$
E_4
$\vec{u}_1$
$\vec{u}_2$
n_3
n_2
u
E_1
$(\underline{u}, \underline{v})$
n_1
$\vec{v}_1$
$(\underline{u}, \overline{v})$
E_2
E_3
E_4
n_3
n_2
E_1
n_1
E_2

(a) (b)

Fig. 3. An example of the oriented-slab box. (a) Shows the three orthogonal slab normals. Slab normal n_1 is equal to $\vec{v}_1 + \vec{v}_2$. u is equal to $\vec{u}_1 + \vec{u}_2$ and slab normal n_2 is obtained from the equation of cross product of n_1 and u. Slab normal n_3 is obtained from the cross product of n_2 and n_1. These three vectors are then normalized. (b) Shows the corresponding oriented-slab box.

Let the oriented-slab box be defined by

$$E_1 + \beta_1 n_1 + \beta_2 n_2 + \beta_3 n_3, \tag{10}$$

where E_1 is a control point of the patch. The coefficients β_1, β_2 and β_3 are intervals. These three intervals can be obtained by transforming the control points onto the local coordinate system formed by these three slab normals. A traced ray intersected with the oriented-slab box yields an interval along the ray. Let a traced ray be defined by

$$O + t\vec{d}, \tag{11}$$

where O is the origin and $\vec{d}$ is the direction vector. To compute this interval, we intersect the ray with the two planes containing opposite faces of the box. The solution of this intersection in terms of the ray parameter t is computed by using Eq. (10) and Eq. (11). After some transformations were performed, we obtain

$$t_k = \frac{(E_1 - O)\cdot n_k + \beta_k}{\vec{d}\cdot n_k}, \quad \text{for } k = 1, 2, 3 \tag{12}$$

The intersection of the three intervals t_k states the parameter subspace that the ray may intersect with the patch. If this subspace is empty, the ray misses the oriented-slab box completely. If the denominator in Eq. (12) is zero, the ray is parallel to the pair of opposite faces. Instead of using complicated rules for distinguishing the cases that the ray lies either between or outside these two opposite faces, we use a very small value for the denominator. If the ray is between these two faces, we get a very large interval t_k containing the interesting domain of t. In another case, t_k is far away and intersection of the three intervals t_k will yield an empty subspace.

There are three main reasons for us to adopt the oriented-slab box. First of all, we can pre-calculate the numerator of Eq. (12) during the preprocessing stage for the cause of all the primary rays starting from the view point. During the rendering step, there are only 3 dot product operations needed for the denominator, 6 division operations for the t_k intervals, and the comparisons for Eq. (12) will be performed when the ray intersects the oriented-slab box. Secondly, it is easily found that the computation will save three arithmetic operations of dot product while calculating the numerator of Eq. (12) compared with the calculation of a secondary ray and an enclosing parallelepiped [7].

Thirdly, the original purpose of the design of oriented-slab box is for parametric surfaces. The characteristic of oriented-slab box is a very tighter bounding volume for flat patches. Such tighter bounding volume provides useful information to determine how many intersection points between a ray and a patch. We can consider a critical case: a ray is nearly tangential to the patch, as shown in Fig. 4. It is likely that more than one intersection point exists. This critical case can be easily recognized because it appears only when the ray passes through the box approximately parallel to the larger face.

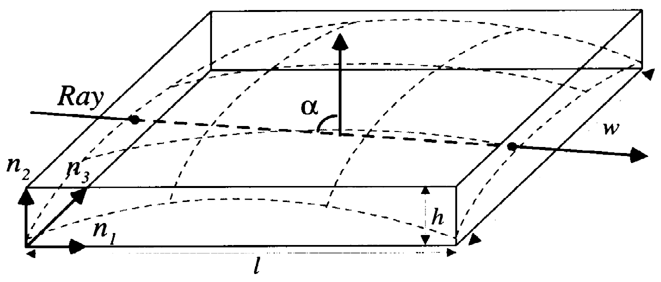


Fig. 4. A critical case that a ray is nearly tangential to the patch. Assume the length of each side of the oriented-slab box are w, h, and l, where $h << w \le l$.

To provide an efficient and precise measure of when this case may happen, we issue out a criterion as follows:

$$\tan\alpha \geq \frac{w}{h}. \tag{13}$$

That is, it is likely that more than one intersection point exists. Hence, we assume that there are more than one intersection point of a ray and a patch when Eq. (13) holds. Otherwise, we assume that a ray intersect with a patch at only one intersection point. This criterion is similar to the rule proposed by Barth and Stürzlinger [7].

3.2 The Obstruction Detection Technique

The theoretical groundwork of obstruction detection technique is based on the surface subdivision [7], the uniform space subdivision [20], the oriented-slab box [6], and Bézier clipping. During preprocessing step, we construct a regular grid, which is the core of this technique and is designed to ensure that the ray-patch tests can be done efficiently. Each surface is adaptively subdivided into *flat* patches according to the rules proposed by Barth and Stürzlinger [7]. For each patch, we construct an oriented-slab box (as mentioned in the previous paragraph) instead of the enclosing parallelepiped. Then a regular grid associated with each surface is constructed to organize these patches. The regular grid is created by applying the uniform space subdivision technique [20] on the bounding volume of each surface. The bounding volume that we used is an axis-aligned parallelepiped.

In rendering step, assume that P is an intersection point between the traced ray L and the surface S. Let P be found by Newton's method. To verify whether the point P is the closest one, we trace a detecting ray BL through the regular grid of the surface S. The detecting ray is started from the intersection point P toward the origin of ray L. Hence, a list of patches will be visited by the ray BL. We could immediately locate the patch containing the point P and calculate Eq. (13) for the ray BL and the patch. If this equation does not holds, we have no need to compute the intersection points between BL and the patch for any further. Otherwise, we use Bézier clipping to find the intersection points between them.

For other patches in the visited list, we use Bézier clipping to find the intersection points between the ray BL and the patches whose oriented-slab boxes are intersected with BL. Using this detecting process, we can either determine whether an intersection point is the closest one or find out the closest intersection point for current traced ray.

For example, as shown in Fig. 5, the dotted grids represent the uniform space subdivision and the curve segments represent the patches. Assume that P_1 is the point found by Newton's method. The detecting ray BL_1 starting at P_1 is traced and SP_2 is the tested patch. We can conclude that P_1 is the closest intersection point because BL_1 intersects the patch SP_2 at only one intersection point and BL_1 does not intersect with other patches' oriented-slab boxes. No matter what, if P_4 is the intersection point found by Newton's method, we will obtain the closest intersection point P_3 by applying Bézier clipping on BL_2 and SP_0, because of the intersection of BL_2 and oriented-slab box of the patch SP_0. On tracing ray L_3, we will obtain that Eq. (13) does hold no matter what P_6 (or P_7) is found by Newton's method. The closest intersection point P_6 will be obtained by applying Bézier clipping on BL_3 and SP_{10}.

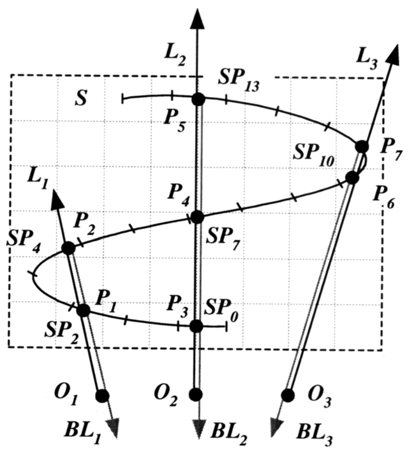


Fig. 5. The demonstration of obstruction detection technique.

The major problem of implementing the obstruction detection technique is the execution efficiency when tracing the detecting rays. The number of ray-patch intersection tested and the number of voxels traveled play important roles in efficiency. The voxel size is the dominating factor because large voxels will induce more ray-patch tests and small voxels will increase the voxel traveling time. In our implementation, the voxel size is chosen as follows.

Given a regular grid, let P_{num} be the number of patches of the surface. X_{len}, Y_{len}, and Z_{len} are the length of the regular grid in X-axis, Y-axis, and Z-axis respectively. Our criterion is that the number of patches should be less than or equal to the number of voxels. Let V_{len} be the length of voxel width. We obtain the following inequality.

$$P_{num} \le \frac{X_{len}}{V_{len}} \times \frac{Y_{len}}{V_{len}} \times \frac{Z_{len}}{V_{len}}.$$

Solving for V_{len} in the above equation, we have

$$V_{len} \le \sqrt[3]{\frac{X_{len} \times Y_{len} \times Z_{len}}{P_{num}}}.$$

In our implementation, the voxel width that we used is $\sqrt[3]{\frac{X_{len} \times Y_{len} \times Z_{len}}{P_{num}}}$.

4. AN EFFICIENT RAY TRACING ALGORITHM

The basic idea of our approach is to combine Bézier clipping and Newton's method to find the ray-surface intersection points. Consider Fig. 6. The dash lines represent rays passing through the current scan line. The black dots denote ray-surface intersection points that obtained by either Bézier clipping (B) or Newton's method (N). The crosses represent the rays that do not intersect with the surface and $\overline{B}(\overline{N})$ denotes that the ray-surface intersection test is performed by Bézier clipping (Newton's method).

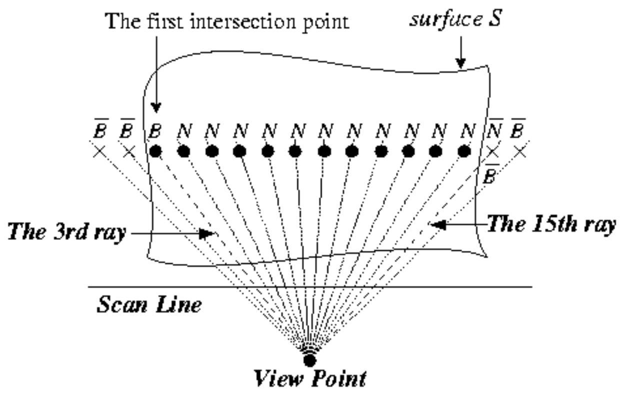

The first intersection point
surface S
$\bar{B}$ $\bar{B}$ B N N N N N N N N N N N N $\bar{B}$
$\bar{B}$
The 3rd ray
The 15th ray
Scan Line
View Point

Fig. 6. The general process for finding the ray-surface intersection points.

Initially, we use Bézier clipping to locate the first ray that intersects the surface *S* and to find the intersections. In this example, the closest intersection point between the third ray and *S* is found by Bézier clipping. Then, we apply Newton's method with the previous intersection point as the initial point to find the next intersection point. After an intersection point is found in this way, we use the obstruction detection technique to either verify whether this point is the closest one or obtain the closest intersection point for current traced ray. If Newton's method fails to converge, for instance, the fifteenth ray in this case, Bézier clipping is performed again to find another intersection point. In the following paragraphs, we will discuss only primary rays if not noted otherwise.

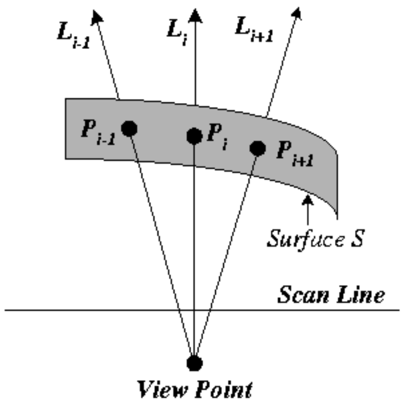


Fig. 7. An improved method for selecting the initial point.

To accelerate the speed of convergence of Newton's method, we use the previously found intersection point as the initial point. Consider Fig. 7, given a scan line on the view plane and a surface *S* that will be tested along the scan line. Assume that $L_{i\text{-}1}$ is the first ray that intersects *S* and the intersection point $P_{i\text{-}1}$ is found by Bézier clipping. Let L_i be the next ray along the scan line. Newton's method with $P_{i\text{-}1}$ as the initial point is performed to find the intersection point between L_i and S. Assume that Newton's method converges to point P_i. For the next ray, say L_{i+1}, the selection rule of the initial point will be different, since the ray coherence property provides another useful meaning: a *change of locations for adjacent intersection points on a surface will not be so apparent*. It seems to be a better choice that taking $P_i + (P_i - P_{i-1})$ as the initial point with the comparison of P_i. Obviously, the advantage is that this new chosen initial point can give rapidly convergence of Newton's method.

In general, the selection of the initial point for the next ray L_{i+1} will be dependent on the method of finding P_i. If P_i is found by Newton's method, $P_i + (P_i - P_{i-1})$ will be the initial point. If P_i is found by Bézier clipping, P_i is the initial point.

In order to identify the convergence of Newton's method, two criteria should be adopted: maximal number of iterations M_n and convergence tolerance value C_v. According to Eq. (7), Newton's method will not stop until one of the following conditions is satisfied:

(a) $|u_{k+1} - u_k| + |v_{k+1} - v_k| \leq C_v$ and $k \leq M_n$;

(b) $k > M_n$;

(c) Y is ill-conditioned;

(d) u or v are outside the parametric intervals (i.e. $u < 0$ or $u > 1$ or $v < 0$ or $v > 1$).

If the iteration is terminated by case (a), Newton's method will be converged and x_{k+1} will be the solution. If the iteration is terminated by case (b) or (c) or (d), we do not find any solution.

5. THE IMPROVEMENT IN TRACING SECONDARY RAYS

Plenty of computational effort has been invested in tracing secondary (reflected, refracted and shadow) rays on ray tracing parametric surfaces. The more light sources in the environment or more transparent and specular surfaces, the higher computational work shall be offered. Since the origin of a secondary ray must lie on a surface, the computations for finding the intersections between them must be performed in order to detect self-shadowing or self-reflection. Here, we assume that the shadow rays are starting from the intersection points toward the light sources. If the secondary ray along the tracing direction does not intersect the surface at the other intersections except the origin of the ray, we have no need to perform further computation anyway.

If we can determine this intersection condition, the unnecessary verification and computation could be saved and performance could be improved. This is not only the key point of this section but also the centrality of our original thought. In this section, we present a method based on the regular grids to detect this intersection condition. The main advantage of this method is that it can be applied to almost all the existing ray tracing algorithms nowadays to improve their performance. The details of this method are presented in the following.

Given a secondary ray L and a surface S containing the origin, we trace the secondary ray through the regular grid of S. Hence, a list of patches will be visited by the ray L. We first compute Eq. (13) of L and the patch containing the origin. If this equation does not hold (which means there is only one intersection point between the ray and the patch), we continue the tracing process. Otherwise, we use Bézier clipping to find the intersection points of L and the patch.

For other patches in the visited list, we use Bézier clipping to find the intersection points between the ray L and the patches whose oriented-slab boxes are intersected with L. Through this process, we can obtain either the condition which the secondary ray along the tracing direction does not intersect the surface at other intersections or the conditions of self-shadowing and self-reflection.

6. EXPERIMENTAL RESULTS

We implement our proposed algorithm in C programming language. All the experiments are executed on a Pentium II with a 300 MHz CPU and 128 MB RAM operating in the Windows NT 4.0 environment. We created four scenes (Figs. 8-11) to demonstrate the computational efficiency for our algorithm. Each scene represents a typical environment for testing the performance. The resolution of the image in each scene was 1024×1024. The number of surfaces and the number of patches are indicated in the caption for each figure. In addition, all of the images shown in this paper are generated using our algorithm with anti-aliasing (four rays per pixel) and ray depth three. The spectral sampling approach [23] is also employed to produce a more realistic color in metals. Nine spectral samples are used in our implementation.

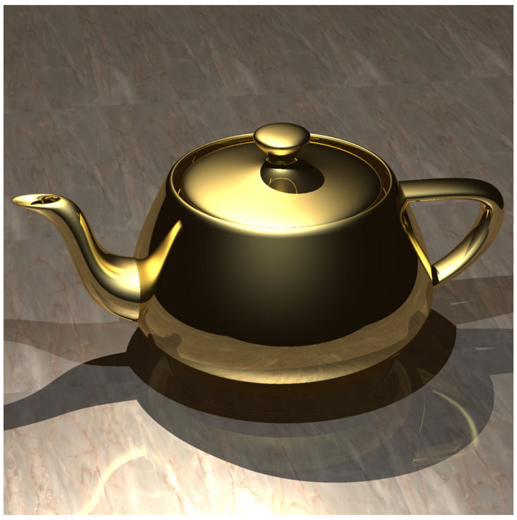


Fig. 8. Newell's teapot with 33 surfaces. These surfaces are subdivided into 4434 patches.

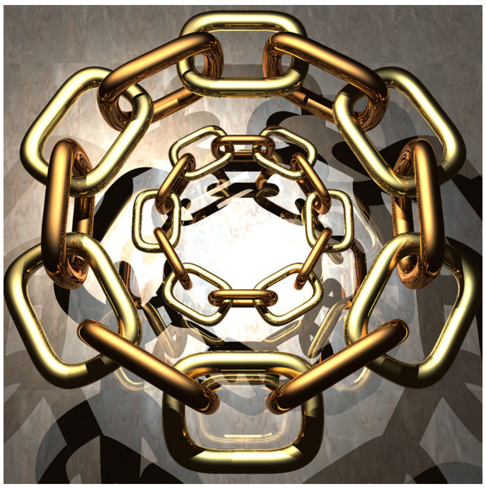


Fig. 9. Twenty-four rings with 298 surfaces, where each ring is constructed using 12 Bézier surfaces. These surfaces are subdivided into 37104 patches.

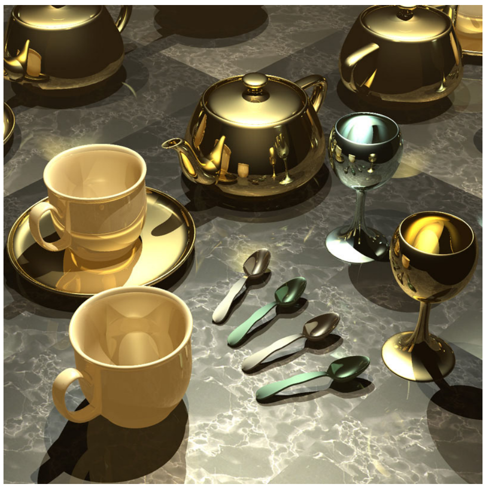


Fig. 10. A highly reflective environment with 255 surfaces. Two flat mirrors and one specular floor are perpendicular to each other. These surfaces are subdivided into 38422 patches.

6.1 Performance Comparisons

It is difficult to make quantitative comparisons with existing ray-surface intersection algorithms, because the performance of these algorithms are dependent on programming style and the organization data structures, for example, the dimension of arrays particularly. Since Bézier clipping [13] is a well-known algorithm, we will compare the performance of our algorithm with that of Bézier clipping.

In our implementation we have modified Bézier clipping according to the schemes proposed by Campagna et al. [16] and we use simple Newton's method to find the ray-surface intersection points. When Newton's method was used, a simple look-up table

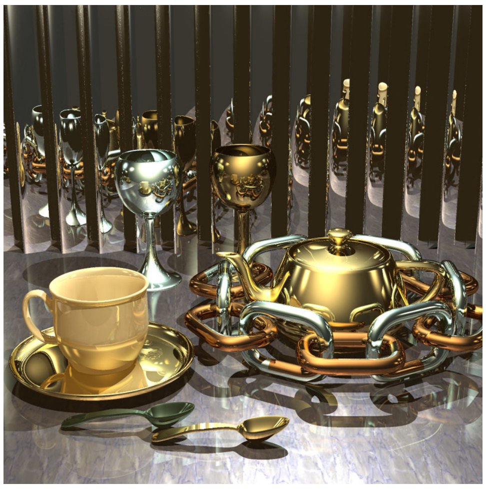


Fig. 11. A more complex scene that contains 343 surfaces. The backdrop consists of fourteen curved mirrors. Two mirrors perpendicular to the floor and located on opposite sides of the floor are not visible in the rendered image. These surfaces are subdivided into 49506 patches.

scheme was employed. For each Bernstein polynomial $B_i^n(u)$ and $B_j^m(v)$, the values of $C_i^n, C_j^m, u^i, (1-u)^{n-i}, v^j$ and $(1-v)^{m-j}$ are stored in separate arrays. Hence, the elements in Newton's method, such as $\partial f_1(u,v)/\partial_u$ or $f_1(u_k, v_k)$ in Eq. (9), could be obtained with a for-loop by using *n*, *i* and *j* as the indices to these arrays.

Both Bézier clipping and Newton's method require a tolerant value to check the convergence of their algorithm. We choose the same tolerant value (10^{-4} in our implementation) for both methods to provide a fair level of comparison. Moreover, we adopt the uniform space subdivision [20] and the simple *min-max* bounding volume (axis-aligned parallelepiped) for both algorithms to reduce the unnecessary ray-surface intersection tests. Tables 1 and 2 present the statistical characteristics for the performance of our algorithm and Bézier clipping on rendering these four scenes. The data included are the sum of rendering a whole scene with 512 × 512 resolutions with ray depth two, but the method for improving the secondary rays is not employed. Nevertheless, this data does not include the computations for finding the intersection points between a ray and a flat surface such as a floor and a flat mirror. The timing unit is given in seconds.

Table 1. The experimental results obtained when our algorithm was used to render the test scenes.

		Fig. 8		Fig. 9		Fig. 10		Fig. 11	
Algorithms	Ray Depth	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Newton's Method	Conv./Call	97%	94%	92%	72%	93%	90%	95%	85%
	Closest	91089	44812	129964	28878	131316	108164	223081	126121
	Not Closest	1352	15967	2050	16589	1285	25238	3999	49904
	Av. Iterations/Pt.	1.7	1.8	2.2	2.9	2.2	2.3	1.9	2.4
Bézier clipping		26226	188517	68154	471200	68097	221076	349829	845348

Table 1 presents the experimental results of our algorithm on rendering these four scenes in ray depth two. Let *sn* be the number of Newton's method terminated successfully and *tn* be the number of Newton's method that have been called. The row *Conv./Call* is the ratio of *sn* to *tn*. For the intersection points found by Newton's method, the row *Closest* indicates the number of points that are verified to be the closest points and the *Not Closest* indicates the number of points are verified not the closest point. The row *Av. Iterations/Pt.* indicates the average number of iterations that Newton's method to converge to an intersection point. The row *Bézier clipping* indicates the number of intersection points found by Bézier clipping.

According to Table 1, the method that we used to select the initial points enable Newton's method to converge rapidly. The most convincing finding is that an intersection point could be obtained in less than two iterations and that about 95% of Newton iteration steps could be terminated successfully.

Table 2 presents a performance comparison between our algorithm and Bézier clipping on rendering these four scenes. Our algorithm achieves a significant improvement on tracing primary rays. The main reasons why our algorithm can achieve such significant improvement are explained in the following. The first reason is the number of inter-

Table 2. A comparison of our algorithm and that of Nishita et. al. on rendering Figs. 7-10.

Fig. 8	Ray Depth	1st		2nd	
	Algorithms	Tot. Pts	Times	Tot. Pts	Times
Our Algorithm	Newton's method	92441	14.9	60779	34.6
	Bézier clipping	26226		188517	
Nishita et al.	Bézier clipping	335335	32.9	288456	38.9
Speed Up			**2.21**		**1.12**
Fig. 9	Ray Depth	1st		2nd	
	Algorithms	Tot. Pts	Times	Tot. Pts	Times
Our Algorithm	Newton's method	132014	23.6	45467	69.7
	Bézier clipping	68154		471200	
Nishita et al.	Bézier clipping	439421	66.5	585129	72.0
Speed Up			**2.82**		**1.03**
Fig. 10	Ray Depth	1st		2nd	
	Algorithms	Tot. Pts	Times	Tot. Pts	Times
Our Algorithm	Newton's method	132601	29.3	133402	53.5
	Bézier clipping	68097		220176	
Nishita et al.	Bézier clipping	297401	61.5	443766	69.3
Speed Up			**2.10**		**1.30**
Fig. 11	Ray Depth	1st		2nd	
	Algorithms	Tot. Pts	Times	Tot. Pts	Times
Our Algorithm	Newton's method	227080	73.5	176025	145
	Bézier clipping	349829		845348	
Nishita et al.	Bézier clipping	1170938	145	1443755	192
Speed Up			**1.97**		**1.32**

section points that needed to be found in our algorithm is obviously less than that of Bézier clipping. The more high curvature surfaces in the environment, the more significant improvement will be. The second reason is that if we take the computation cost of finding an intersection point as a measure, Newton's method is potentially lower in cost than Bézier clipping. Hence, the more intersection points found using Newton's method, the more the execution time saved. For example, when the image shown in Fig. 9 was traced, most of the traced rays intersected the ring surfaces at two intersection points. The number of intersection points found by our algorithm are at about half that found by Bézier clipping. The performance of our algorithm is 2.82 times faster than that of Bézier clipping. This is the most noticeable result of our algorithm.

The performance of our algorithm on tracing secondary rays depends heavily on the coherence property. If an environment provides good coherence in tracing secondary rays, our algorithm can achieve a very significant improvement. For example, Fig. 10 and Fig. 11 are created to provide such environments, our algorithm did improve the performance of Bézier clipping by 25%. Mirror or highly reflection effects are used in many practical applications, such as movie film production and music television video (MTV). To animate parametric surfaces in such environments, our algorithm will be very helpful in improving the performance efficiency.

To determine the performance of our algorithm in tracing secondary rays, we applied it to the above four scenes. The experimental results are presented in Tables 3 and 4. For secondary and third ray depths, Table 3 presents a comparison of the total number of intersection points found by using Bézier clipping and Table 4 presents the execution time spent. For shadow rays, the experimental results counted for ray depth three. Table 3 shows the percentage of ray-surface pairs that did not require the determination of intersection points and Table 4 presents the execution time. Note that this data does not include the computations for dealing with flat surfaces such as floors and flat mirrors.

Table 3. Statistics for before and after the improvement of tracing secondary rays. The row *Avoid/Call* indicates the percentage of shadow ray-surface pairs that did not require the determination of intersection points with our proposed method.

	Fig. 8		Fig. 9		Fig. 10		Fig. 11	
Ray Depths	2nd	3rd	2nd	3rd	2nd	3rd	2nd	3rd
Before Improvement	249296	111526	516667	247669	353589	266263	1021373	622589
After Improvement	82510	11484	153661	75821	197254	110713	483597	223039
Shadow Rays	Scene 1		Scene 2		Scene 3		Scene 4	
Avoid/Call	91%		85%		91%		84%	

The occurrence of the circumstances of self-shadowing or self-reflection is quite seldom for the general case. Avoiding the unnecessary computation of ray-surface intersections can be achieved by some simple and straight forward testing in our algorithm. Obviously, the advantage of our proposed strategy is that the number of ray-surface intersections that need to be found can be decreased. This is why our proposed algorithm can achieve a very significant improvement on tracing secondary rays and shadow rays.

Table 4. The speedup of our proposed method on tracing secondary rays.

Fig.	Ray Depth	2nd	3rd	Shadow Rays
8	Before Improvement	34.6	21.9	193
	After Improvement	23.6	10.2	113
	Speed Up	**1.47**	**2.15**	**1.71**
9	Before Improvement	69.7	37.6	284
	After Improvement	44.8	22.6	125
	Speed Up	**1.56**	**1.66**	**2.27**
10	Before Improvement	53.5	54.4	326
	After Improvement	44.2	40.1	211
	Speed Up	**1.21**	**1.36**	**1.55**
11	Before Improvement	145	135	1027
	After Improvement	114	91.3	363
	Speed Up	**1.27**	**1.48**	**2.83**

6.2 The Utilization of Newton's Method on High Accuracy Environments

According to our experimental results, the method we used to select the initial points enable Newton's method to converge rapidly. In our implementation, the tested environments are rendered by the simple Newton's method with 10^{-4} as the tolerance value. To evaluate the behavior of our algorithm when the high accuracy was requirement, we render the tested environments using both the simple and the original Newton's method with different tolerance values. Table 5 presents the results obtained when our algorithm was used to trace primary rays with tolerance values of 10^{-4}, 10^{-6}, 10^{-8}, and 10^{-10}.

Some constructive and meaningful results reveal to us. The method we used to select the initial points provided stable convergence for the original Newton's method as more significant digits are required. In addition, the performance of Newton's method is better than that of the simple Newton's method. Hence, our algorithm is useful when a very high level of accuracy is required. Under such environments, Newton's method is a better choice than the simple Newton's method.

7. CONCLUSIONS

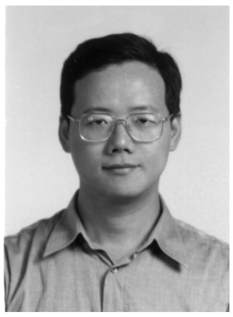
We have proposed an efficient and stable ray tracing algorithm for finding the ray-surface intersection points. The experimental results indicate that the combination of Bézier clipping and Newton's method can provide faster speed in finding the ray-surface intersections. The improvement of our algorithm on primary rays is remarkable. Another significant contribution is that we proposed a novel method to improve the performance for tracing secondary rays. This approach introduces a 20% to 50% reduction in total rendering time for tracing secondary rays.

Our algorithm is very helpful in curved surface design and animation of curved surfaces. For curved surface designed environment [24, 25], our ray tracing algorithm can provide true view-dependent highlights efficiently. For animation of curved surfaces, our algorithm can save lots of computation time in simulating the reflections between curved surfaces and the environment.

REFERENCES

1. T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, Vol. 23, 1980, pp. 343-349.
2. D. Toth, "On ray tracing parametric surfaces," *Computer Graphics* (*SIGGRAPH* '85 *Proceedings*), Vol. 19, 1985, pp. 171-179.
3. D. Lischinski and J. Gonczarowski, "Improved techniques for ray tracing parametric surfaces," *The Visual Computer*, Vol. 6, 1990, pp. 134-152.
4. M. A. J. Sweeney and R. H. Bartels, "Ray tracing free-form B-spline surface," *IEEE Computer Graphics & Application*, Vol. 6, 1986, pp. 41-49.
5. C. G. Yang, "On speeding up ray tracing of B-spline surfaces," *Computer Aided Design*, Vol. 19, 1987, pp. 122-130.
6. J. Yen, S. Spach, M. Smith, and R. Pulleyblank, "Parallel boxing in B-spline intersection," *IEEE Computer Graphics & Applications*, Vol. 11, 1991, pp. 72-79.
7. W. Barth and W. Stürzlinger, "Efficient ray tracing for Bézier and B-spline surfaces," *Computer & Graphics*, Vol. 17, 1993, pp. 423-430.
8. W. Barth, R. Lieger, and M. Schindler, "Ray tracing general parametric surface using interval arithmetic," *The Visual Computer*, Vol. 10, 1994, pp. 363-371.
9. K. Qin, M. Gong, Y. Guan, and W. Wang, "A new method for speeding ray tracing NURBS surfaces," *Computer & Graphics*, Vol. 21, 1997, pp. 577-586.
10. W. Stürzlinger, "Ray-tracing triangular trimmed free-form surfaces," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, 1998, pp. 202-214.
11. K. Joy and M. Bhetanabhotla, "Ray tracing parametric surface patches utilizing numerical techniques and ray coherence," *Computer Graphics* (*SIGGRAPH* '86 *Proceedings*), Vol. 20, 1986, pp. 279-285.
12. C. Woodward, "Ray tracing parametric surfaces by subdivision in viewing plane," *Theory and Practice of Geometric Modeling*, Spring-Verlag, New York, 1989, pp. 273-290.
13. T. Nishita, T. W. Sederberg, and M. Kakimoto, "Ray tracing trimmed rational surface patches," *Computer Graphics* (*SIGGRAPH* '90 *Proceedings*), Vol. 24, 1990, pp. 337-345.
14. Y. Dobashi, T. Nishita, H. Yamashita, and T. Okita, "Modeling of clouds from satellite images using metaballs," in *Proceedings of Pacific Graphics* '98, *Sixth Pacific Conference on Computer Graphics and Applications*, 1998, pp. 53-60.
15. T. Nishita, H. Iwasaki, Y. Dobashi, and E. Nakamae, "A modeling and rendering method for snow by using metaballs," *Computer Graphics Forum*, Vol. 16, 1997, pp. 357-364.
16. S. Campagna, P. Slusallek, and H. Seidel, "Ray tracing of spline surfaces, Bézier clipping, chebyshev boxing, and bounding volume hierarchy — a critical comparison with new results," *The Visual Computer*, Vol. 13, 1997, pp. 265-282.
17. J. T. Kajiya, "Ray tracing parametric patches," *Computer Graphics* (*SIGGRAPH* '82 *Proceedings*), Vol. 16, 1982, pp. 245-254.

18. D. Manocha, "Algebraic pruning: A fast technique for curve and surface intersections," Technical report TR93-062, Department of Computer Science, University of N. Carolina, Chapel Hill., 1993.
19. D. Manocha and J. Demmel, "Algorithms for intersecting parametric and algebraic curves I: Simple intersection," *ACM Transactions on Graphics*, Vol. 13, 1994, pp. 73-100.
20. J. G. Cleary and G. Wyvill, "Analysis of an algorithm for fast ray tracing using uniform space subdivision," *The Visual Computer*, Vol. 4, 1988, pp. 65-83.
21. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, San Diego, 1988.
22. R. L. Burden and J. D. Faires, *Numerical Analysis*, Prindle, Weber and Schmidt, Boston, 1985.
23. R. Hall, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1988.
24. E. Kaufmann and R. Klass, "Smoothing surfaces using reflection lines for families of splines," *Computer Aided Design*, Vol. 20, 1988, pp. 312-316.
25. A. F. Lennings, J. C. Peters, and J. S. M. Vergeest, "An efficient integration of algorithms to evaluate the quality of freeform surfaces," *Computer & Graphics*, Vol. 19, 1995, pp. 861-872.



Shyue-Wu Wang (王學武) received the M.S. degree in Computer Science from National Tsing Hua University, Taiwan, R.O.C., in 1992, and is currently a doctoral candidate at National Chiao Tung University. His research interests include global illumination, real-time rendering, image-based rendering and geometric modeling.

Zen-Chung Shih (施仁忠) was born on 10th February 1959, in Taipei, Republic of China. He received his B.S. degree in Computer Science from Chung-Yuan Christian University in 1980, M.S. degree in 1982 and Ph.D. degree in 1985 in Computer Science form the National Tsing Hua University. Currently, he is a professor in the Department of Computer and Information Science at the National Chiao Tung University in Hsinchu. His current research interests include procedural texture synthesis, non-photorealistic rendering, global illumination, and virtual reality.

Ruei-Chuan Chang (張瑞川) was born on 30th January 1958, in Keelung, Taiwan, Republic of China. He received his B.S. degree in 1979, M.S. degree in 1981, and Ph.D. degree in 1984, all in Computer Science from the National Chiao Tung University. Currently he is a professor in the Department of Computer and Information Science at National Chiao Tung University in Hsinchu. He is also an associate research fellow at the Institute of Information Science, Academia Sinica, Taipei. His current research interests include design and analysis of algorithms, computer graphics, and system software.