Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 8961577, 16 pages
https://doi.org/10.1155/2022/8961577

Research Article

Ray Tracing Acceleration Algorithm Based on FaceMap

Jian Wang, Hui Xiao, and Hongbin Wang

Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650504, China

Correspondence should be addressed to Hongbin Wang; whbin2013@kust.edu.cn

Received 7 January 2022; Revised 28 February 2022; Accepted 7 March 2022; Published 25 April 2022

Academic Editor: Wei Liu

Raster method and ray tracing are two important algorithms in computer graphics. The former, raster method, marked by high speed and efficiency, has the disadvantage of an unrealistic rendering effect. By contrast, ray tracing requires considerable computing resources and time despite its advantages of high fidelity and simple structure, thus only suitable for offline rendering. Given such limitations, a ray-tracing acceleration algorithm that combines the raster method and ray tracing is proposed in this study based on FaceMap, a two-dimensional data structure that stores surface distribution under point light or the view of a camera. Firstly, all triangular surfaces of the 3D model are rasterized by linear surface projection to form the surface distribution of a spherical panorama. Secondly, the structure of FaceMap is adopted to store the distribution information of all the surfaces in the scene. Thirdly, the data on the intersecting surfaces in this direction are collected by projecting the ray onto FaceMap in the ray-tracing process, thus reducing the intersection and propagation operations and improving the efficiency. Four object models including chessboard, grass patch, bust, and typewriter are selected for comparative rendering experiments. Our proposed method is used to compare with Rhino and VRay rendering software, respectively. The results show that our proposed method obtained better rendering effects within greatly reduced rendering time (© 2018Optical Society of America).

1. Introduction

Raster method and ray tracing are two branches of computer graphics. Raster method has the advantages of fast speed and high efficiency in scene rendering, but the disadvantage is that the rendering degree is far less than the reality; ray tracing has the advantages of high rendering reality and simple structure, and the disadvantage is that it consumes too much calculation, so it can only be used in offline rendering.

Traditional ray-tracing algorithms adopt the Monte Carlo illumination model [1], which simulates the propagation of light in the real physical environment. After the conduct of a series of operations such as specular reflection, perspective refraction, and diffuse reflection, an image is finally generated. However, it is difficult to determine which surfaces the light may intersect within the propagation process due to the relatively complex rendered scenes. Therefore, the main calculation in the ray-tracing process lies in the intersection operation of the ray and surface.

To narrow the range of surfaces requiring intersection operation, commonly used techniques are ray-tracing acceleration algorithms including the bounding box method, three-dimensional DDA (3D-DDA) algorithm, and octree division method:

(i) Bounding box method: the basic idea of the bounding box is to use geometric objects of regular shapes to wrap each model in the scene, respectively. The adjacent bounding boxes are wrapped in larger boxes to form a tree structure [2]. The intersection operation starts from the root node of the tree and then to the subnodes. It will involve the sub-bounding boxes or the surfaces of inner models of the bounding box if there is an intersection point. Otherwise, the ray will not intersect with any surface of the inner models in the bounding box.

(ii) 3D-DDA algorithm: the low efficiency of ray intersection operation is mainly due to the uncertainty about the surfaces that may intersect and that are closest to the ray source. DDA algorithm evenly

divides the space into three-dimensional grids, within which the corresponding surface is stored [3]. In the process of intersection operation, the ray starts from the grid where the ray source is located and successively intersects with the encountered surfaces inside other grids. If an intersection point appears, it will be the closest to the ray source.

(iii) Octree division method: similar to the 3D-DDA algorithm, the octree division method divides the space into nonuniform grids [4]. A cube that can wrap the whole scene is divided into eight blocks, and the number of surfaces in each block is calculated. The block will be further divided into eight sub-blocks if the number is larger than the threshold; otherwise, the division stops. The above steps are repeated until the surface number of every sub-block is smaller than the threshold.

The above three methods are all based on space dividing, which means dividing the space capable of wrapping the whole scene into different grids that contain some surfaces inside. When the ray propagates between different grids, the intersection operation is conducted between the ray and the surface within a grid.

Although these methods are efficient in scene rendering, they require additional calculation of ray propagation between grids and fail to determine whether there is an intersection between surfaces in a direction before the ray leaves the scene. Such drawbacks are particularly serious when there are numerous light sources in the scene [5]. The reason is that for each intersection on a surface, a shadow test line is required to be sent out to the light source for the purpose of examining whether it would be obscured by other surfaces.

It can be seen from the above analysis that both the raster method and ray tracing have their own advantages and disadvantages. A natural idea is to combine the advantages of the two algorithms to solve the aforementioned problems. As a remedy, we propose FaceMap, a two-dimensional data structure, to store surface distribution corresponding to the point light source or camera. It is by virtue of FaceMap that the surfaces intersected in a certain direction can be rapidly determined, and the efficiency of intersection operation can thus be improved.

The rest of this paper is organized as follows: Section 2 and Section 3 introduce the raster method and the ray-tracing method, respectively. In Section 4, the ray-tracing acceleration algorithm based on FaceMap is illustrated in detail. Experimental results and analysis are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Raster Method

Raster rendering has the advantages of speed and efficiency. Because of its special rendering pipeline flow, raster rendering can be well integrated into the hardware (such as GPU). Besides, customized programming based on particular needs is made possible to obtain the desired picture effect. The pipeline flow mainly consists of vertex conversion, primitive assembly, rasterization, and interpolation coloring [6, 7].

2.1. Vertex Conversion. Each 3D rendering engine involves a space camera, whose projection result of a 3D scene is an image seen by a user. The camera itself is in the 3D space, capable of basic movement, rotation, and other different view transformations.

The basic unit of the 3D model is a triangular surface with three vertices where the 3D coordinate data are stored in accordance with the world coordinate system. The data do not change when the model remains static. However, if the model coordinates cannot be calculated according to the world coordinate system due to changing the camera's angle of view, they should be converted to the camera's space. This is called vertex conversion. When the coordinate data of the model's vertices are transformed into the visual field of the camera, the basic projection transformation can be conducted. There are two kinds of projection transformation, namely perspective projection and orthogonal projection. The former is similar to what is observed by human eyes, that is, an object is big when near and small when far, complying with the perspective principle. By comparison, the latter is similar to the scene when all models are compressed to a plane while retaining the original sizes, which is not in line with the perspective principle. Therefore, raster rendering mainly focuses on perspective projection.

2.2. Perspective Projection. The camera used in the raster method usually contains four parameters, namely viewing angle θ, near clipping plane distance n, far clipping plane distance f, and screen aspect ratio α, with the near clipping plane, serving as the projection plane, as shown in Figure 1.

For points that have been converted to the camera space, their 3D coordinates can be transformed accordingly by the projection transformation matrix M. At this point, the coordinates are not clipped in the screen space; the coordinates on the near projection plane can then be obtained. In addition, the transformation matrix can be regarded as the process of the space points being transformed into the near projection plane. Accordingly, the reverse process is to transform the points on the projection plane into the vectors in the camera space and thus is often used to solve the coordinates and rotation angles of camera. The transformation matrix M is expressed as follows:

$$M = \begin{bmatrix} \frac{\cot\theta}{\alpha} & 0 & 0 & 0 \\ 0 & \cot\theta & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & \frac{fn}{n-f} & 0 \end{bmatrix}. \quad (1)$$

Then, the points after perspective projection can be transferred to the screen coordinate system through the screen matrix S, and the matrix S is shown as follows:

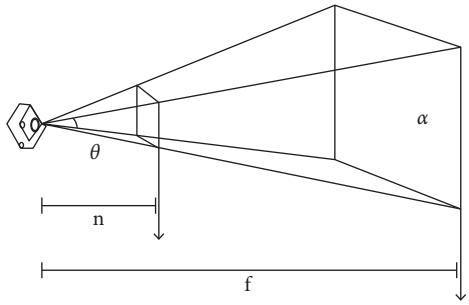


Figure 1: Perspective camera.

$$S = \begin{bmatrix} \frac{\text{width}}{2} & 0 & 0 & 0 \\ 0 & -\frac{\text{height}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{\text{width}}{2} & \frac{\text{height}}{2} & 0 & 1 \end{bmatrix}. \quad (2)$$

2.3. Triangular Rasterization. The smallest polygon in a two-dimensional plane is a triangle, and any polygon can be viewed as a combination of several triangles. Hence, triangles are often used as basic structure for 3D models. For a triangle that has been transferred to screen coordinate system, its three edges and inside pixels need to be filled with progressive scanning being the frequently adopted method [8], as shown in Figure 2.

The pseudo-code of the process is described in Algorithm 1.

The highest and lowest points of a triangle should not exceed the scope of the display screen and, if necessary, should be reduced to 0~(height − 1). Specifically, a triangle edge intersects with a certain line y if and only if one vertex of the edge is below y and the other vertex is not. Assume the two vertices of the edge are (x_1, y_1) and (x_2, y_2), respectively; then the equation that solves the x-coordinates of the vertices is expressed as follows:

$$x = x_2 + (x_2 - x_1)\frac{y - y_1}{y_2 - y_1}. \quad (3)$$

The two intersection points of the triangle edge and the line y can be obtained by the above method. The rasterization of the line is completed after filling the pixels between the two points. However, one drawback of triangle rasterization lies in the serious jagged edges of a triangle. A normal solution to such problem is to increase the sampling points in the edge pixels, as shown in Figure 3.

The edge pixel is divided into several pixels for rasterization, and then, the average value of the pixel color is calculated, serving as the color of the edge pixel. This approach can greatly alleviate the jag effect and meanwhile achieves low consumption since only edges require upsampling. Therefore, it can also be applied to mobile devices such as cell phones [9].

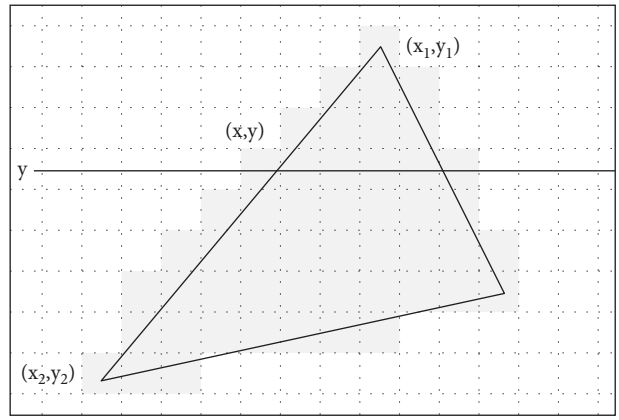


Figure 2: Triangle rasterization.

Given the triangular surfaces of the model, its fineness can be regarded as equivalent to the number of triangles. However, a detail-emphasized model will significantly increase rendering time due to its large number of triangular surfaces. Therefore, existing improved algorithms use a small number of points to simulate a high-accuracy model. After fitting surfaces by voxel rendering (as shown in Figure 4), the numbers of vertices and triangular surfaces are reduced, and the rendering of the model is accelerated [10].

3. Ray Tracing

Scene rendering of ray tracing mainly involves the process of sampling, reconstructing, synthesizing, and resampling the global illumination, for the virtual 3D space that can be viewed as the combination of objects and light sources. Unlike in the raster method, objects in ray tracing are not directly observed by cameras. Instead, certain light distribution forms in space after the light source and objects are illuminated by light, and what the camera captures is the reflected light as a result of the intersection of the ray and surface [11].

Ray tracing mainly computes the intersection of the point-based ray and the triangular surface in the space and then obtains u and v of the intersection point relative to the triangle vertices.

Assume the ray starts from point O, and the unit direction vector is D which intersects with the triangle (V_0, V_1, V_2) at point P. Then, P can be viewed as translated from V_0 corresponding to V_1 and V_2, the distance between P and O being t, as displayed in Figure 5.

The equation obtained is expressed as follows [12]:

$$O + Dt = (1 - u - v)V_0 + uV_1 + uV_2 \begin{cases} u \geq 0 \\ v \geq 0 \\ u + v \leq 1 \end{cases}. \quad (4)$$

By rearranging the above equation and extracting t, u, and v as unknowns, the linear equations can be described as follows:

Input: A triangle vertex information
(1) Find the highest and lowest points of the triangle;
(2) Traverse from the highest point to the lowest point:
    (a) Calculate the two intersection points of the current line and the triangle edge;
    (b) Fill from the left node to the right node;
Output: Results after transfer

Algorithm 1: A triangle transfer to the screen coordinate system.

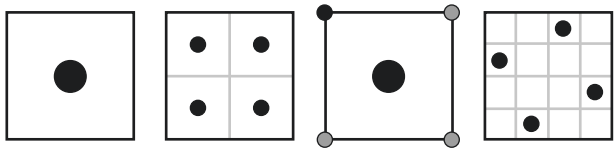


Figure 3: Edge pixel sampling.

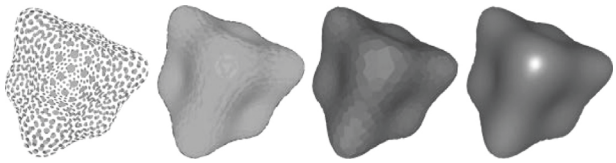


Figure 4: Surface simulation by the sparse point set.

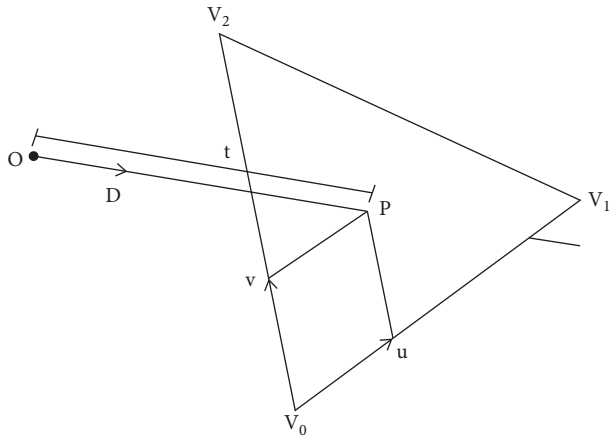


Figure 5: Intersection of the ray and surface.

$$\left[-D\left(V_1 - V_0\right)\left(V_2 - V_0\right)\right]\begin{bmatrix} t \\ u \\ v \end{bmatrix} = o - V_0, \tag{5}$$

$$\begin{bmatrix} -D & E_1 & E_2 \end{bmatrix}\begin{bmatrix} t \\ u \\ v \end{bmatrix} = T. \tag{6}$$

Assume $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, and $T = O - V_0$, and then, equation (5) can be rewritten as equation (6).

Next, according to Cramer's rule and the mixed product formula, t, u, and v are solved, as expressed as follows:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\left|D \times E_2 \cdot E_1\right|}\begin{bmatrix} T \times E_1 \cdot E_2 \\ D \times E_2 \cdot T \\ T \times E_1 \cdot D \end{bmatrix}. \tag{7}$$

To avoid repeated operations, assume $R = D \times E_2$ and $Q = T \times E_1$, and thus, equation (7) is simplified as follows:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\left|R \cdot E_1\right|}\begin{bmatrix} Q \cdot E_2 \\ R \cdot T \\ Q \cdot D \end{bmatrix}. \tag{8}$$

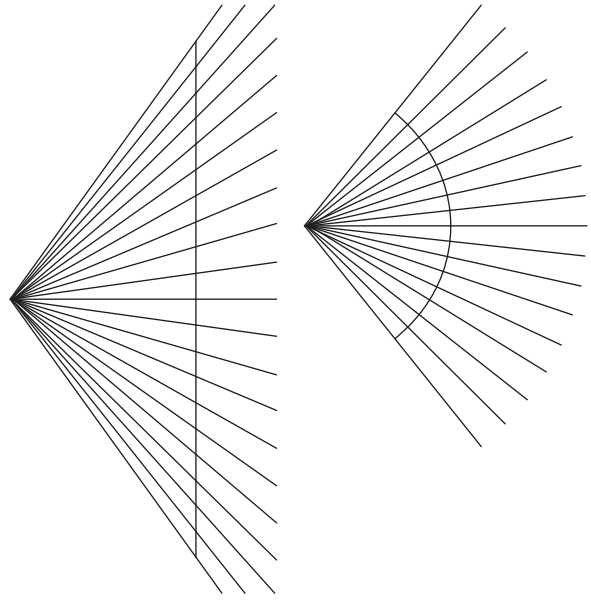
4. Ray-Tracing Acceleration Algorithm Based on FaceMap

The raster method is characterized by high speed and efficiency but has the disadvantage of an unsatisfactory rendering effect. In contrast, the ray-tracing method features in simple mechanism and high fidelity, but it only applies to offline rendering due to its high computational complexity and time complexity. Therefore, the idea of combining the advantages of the two algorithms becomes very natural and intuitive.

4.1. Basic Idea of FaceMap. Conventional raster projection is a kind of linear plane projection, with linear variations of the distance between points relative to the origin. By contrast, the projection of FaceMap belongs to linear surface projection, with linear variations of angles between points relative to the origin. FaceMap consists of four parts as follows: linear surface projection, curve approximation by bisection method, filling of the curved triangle, and reverse solution of light vector.

Figure 6 clearly shows that in terms of ordinary plane projection, the angle between point's narrows with the decrease in the distance between points and the screen edge. Consequently, serious deformation can occur at the screen edge, and the larger the field angle is, the more serious the deformation becomes.

A typewriter model was extracted via Adobe After Effect 2015 (AE for short), and the conversions of the camera's field angle are shown in Figure 7. To go into detail, the upper left field angle was 50°, basically without deformation; the upper



(a) (b)

Figure 6: (a) Plane projection. (b) Surface projection.

right field angle was 90°, with tensile deformation near the bottom corners; the bottom left field angle was 120°, with particularly evident tensile deformation; the bottom right field angle was 170° (because AE is unable to set the field angle as 180°, or in other words, the field angle in plane projection cannot reach 180°), with the whole typewriter model being stretched to a strip.

In comparison, the linear surface projection based on FaceMap is similar to real optical lens in everyday life, with a complete 360-degree visual field. It can generate various image effects including wide-angle, fisheye, and ultra-wide-angle. As shown in Figure 8, the upper left field angle was 50°, basically without deformation (similar to that in plane projection); the upper right field angle was 90°, with an overall bent deformation; the field angles of the lower left and the lower right were 120° and 170°, respectively, both with the effect closer to that obtained by fisheye lens.

FaceMap is a two-dimensional data structure that stores the spherical panorama distribution of the scene. Therefore, after mapping the object in the scene to FaceMap, a spherical panorama can be generated. Figure 9 presents the FaceMap schematic diagram of the interior of a typewriter model with spatial origin (0, 0, 0), the field angle being 360°. Each point in the figure stored the surface data in that direction. A darker color of points indicated a larger number of surfaces in the direction, and a lighter color, a smaller number of surfaces.

In the ray-tracing process, for ray vectors emitted by the camera, surfaces that might intersect can be determined directly by FaceMap. In addition, the shadow test line facing the light source can also find out whether there is any surface as an obstacle, requiring no light propagation operation.

In this study, the scene surface was projected onto FaceMap by the raster method so as to accelerate the generation. Instead of using ordinary linear plane projection whose maximum field angle is approximately 180°, this paper adopted the linear surface projection which has a 360-degree visual field for the generation of FaceMap.

As shown in Figure 10, the central point of FaceMap was O. After projecting a triangular surface in the scene onto the sphere of FaceMap, a curved triangle ABC was formed, and the three corresponding edges, denoted as a, b, and c, were all curves.

The detailed raster process is shown in Figure 11. The three vertices A, B, and C of the triangle were projected onto the two-dimensional structure of FaceMap by virtue of linear surface projection. Then, the bisection method was used to approximate any curved edge of the triangle, and finally, the scanning line filling method was employed to fill the curved triangle.

4.2. FaceMap Algorithm

4.2.1. Linear Surface Projection. The BIT-VBF left-handed 3D coordinate system was adopted in this paper. The vertex coordinate transformation was viewed as converting points from the world coordinate system to the camera coordinate system, by virtue of the transformation matrix [13]. The position coordinates of any point in 3D space were represented by (x, y, z), while the world coordinate axes were described as axis $X(1, 0, 0)$, axis $Y(0, 1, 0)$, and axis $Z(0, 0, 1)$ successively. Similarly, a camera also possessed position coordinates and three axes. The position of the camera was assumed as camPos,

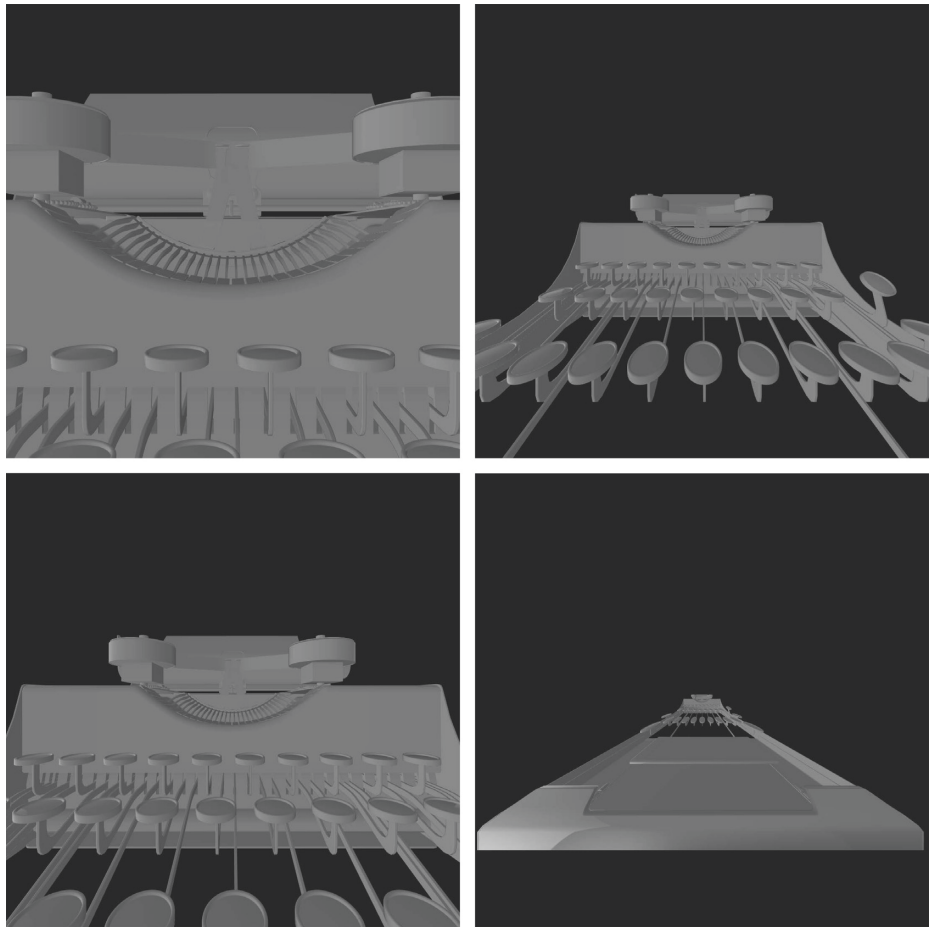


Figure 7: Plane projection with field angles of 50°, 90°, 120°, and 170°.

with the three axes being camAX, camAY, and camAZ, respectively, and the axis coordinates would change according to the rotation of the camera.

The angle changes relative to the screen center are linear in linear surface projection. In other words, if the distance between a point and the central origin is two times that of another point, then the angle between this point and the line of sight is also two times that of the other point.

The field angle of a surface was denoted as θ, the center as camPos, and the three axes as camAX, camAY, and camAZ. For a certain point v in the space, its vector cv relative to the camera was calculated by

$$\begin{aligned} \mathrm{cv}.x &= v.x - \mathrm{camPos}.x, \\ \mathrm{cv}.y &= v.y - \mathrm{camPos}.y, \\ \mathrm{cv}.z &= v.z - \mathrm{camPos}.z. \end{aligned} \tag{9}$$

Vector operation was expressed as

$$\mathrm{cv} = v - \mathrm{camPos}. \tag{10}$$

The calculated cv was the converted coordinates when the camera remained static, while the axis coordinates changed when the camera rotated, and the correspondingly converted vertex coordinates tv were calculated by

$$\begin{aligned} \mathrm{tv}.x &= \mathrm{cv}.x \times \mathrm{camAX}.x + cv.y \times \mathrm{camAX}.y \\ &\quad + \mathrm{cv}.z \times \mathrm{camAX}.z, \\ \mathrm{tv}.y &= \mathrm{cv}.x \times \mathrm{camAY}.x + cv.y \times \mathrm{camAY}.y \\ &\quad + \mathrm{cv}.z \times \mathrm{camAY}.z, \\ \mathrm{tv}.z &= \mathrm{cv}.x \times \mathrm{camAZ}.x + cv.y \times \mathrm{camAZ}.y \\ &\quad + \mathrm{cv}.z \times \mathrm{camAZ}.z. \end{aligned} \tag{11}$$

After simplification, the transformation matrix was expressed as

$$\mathrm{tv} = \begin{bmatrix} \mathrm{camAX} \\ \mathrm{camAY} \\ \mathrm{camAZ} \end{bmatrix} [\mathrm{cv}]. \tag{12}$$

The transformation matrix can operate a series of transformations of the model, such as translation, rotation, and scaling. Only one point in the space is needed to move and rotate, and then, all the vertices of the model experience coordinate transformation relative to that point, thereby resulting in the converted model. Besides, scaling could be realized merely by multiplying one factor in the matrix [14].

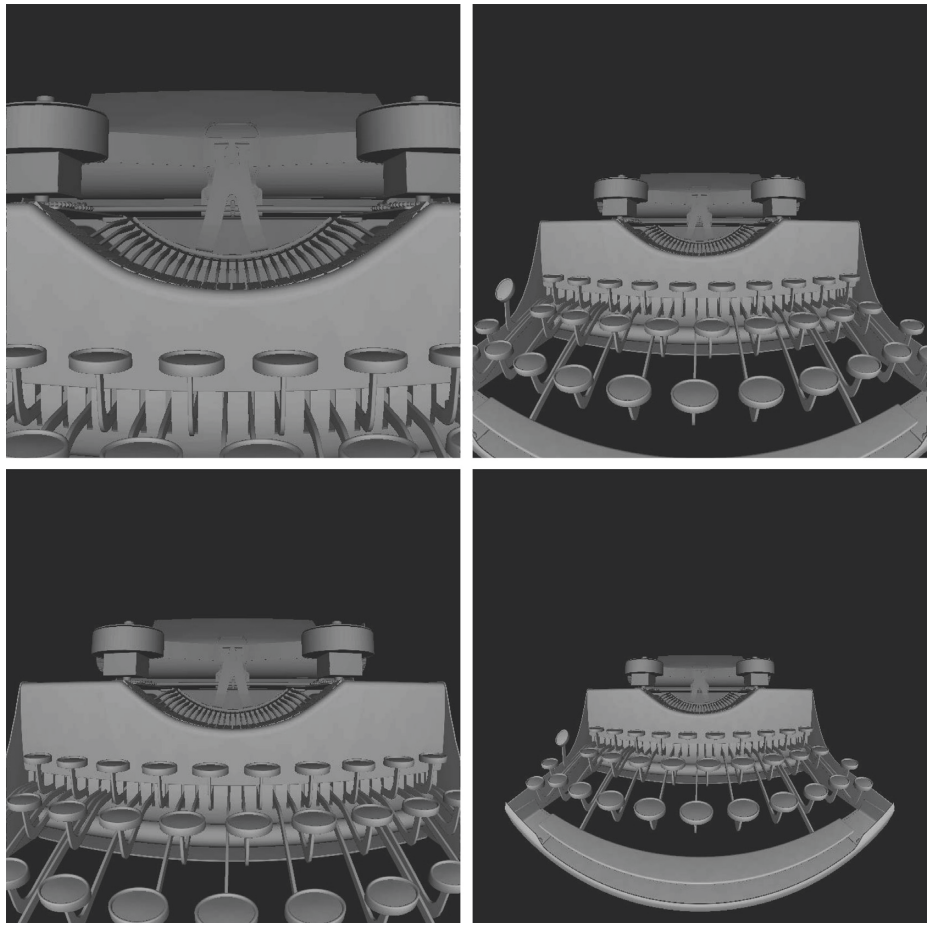


Figure 8: Surface projection with field angles of 50°, 90°, 120°, and 170°.

Figure 9: FaceMap schematic diagram.

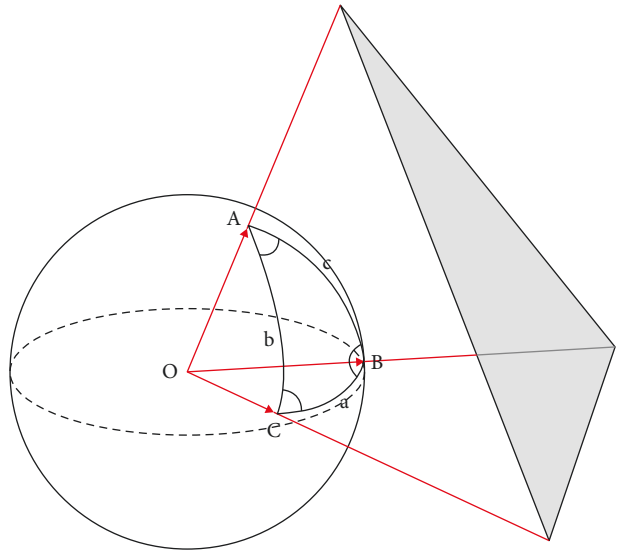


Figure 10: Surface projection.

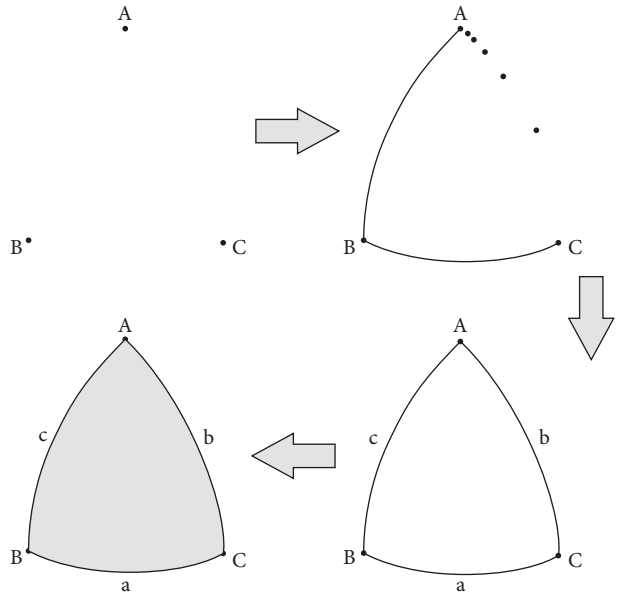


Figure 11: Curved triangle rasterization.

The angle β between tv and the sight axis was calculated by

$$\beta = \text{acos}\left(\frac{\text{tv} \cdot \text{camAZ}}{|\text{tv}||\text{camAZ}|}\right). \tag{13}$$

Next, the screen radius was assumed to be R, so the distance t between point sv after mapping and the screen center was obtained by the following equation:

$$t = \frac{2\beta}{\theta} R. \tag{14}$$

Given the x-coordinate and y-coordinate of tv, the direction of its projection on the XOY plane of FaceMap could be determined. The distance between tv on the XOY plane and the central origin was denoted as Dist, and then, the ratios kx and ky of components in the OX and OY directions were obtained, as calculated by

$$\begin{aligned} \text{Dist} &= \sqrt{x^2 + y^2}, \\ kx &= \frac{x}{\text{Dist}}, \\ ky &= \frac{y}{\text{Dist}}. \end{aligned} \tag{15}$$

At last, the sx – coordinate and sy – coordinate of point sv projected onto FaceMap were calculated by

$$\begin{bmatrix} sx \\ sy \end{bmatrix} = t \begin{bmatrix} kx \\ ky \end{bmatrix}. \tag{16}$$

4.2.2. Curve Approximation by the Bisection Method. A curve can be seen as a collection of countless points, while a line in screen space is a collection of finite points. Therefore, curves in a space can be approximated by calculating finite pixels. Given the method of perspective projection, the curve mapping method in screen space is described in Algorithm 2.

4.2.3. Filling of the Curved Triangle. The curved triangle needs to be filled after all its points are obtained. Similar to the rasterization of regular triangles, the filling of the curved triangle also employed the progressive scanning method is described in Algorithm 3.

The number of intersections per row, though might more than two due to the curved triangle edges, could only be even, so the filling was conducted in pairs.

For a curved triangle, the intersection of each line and the curved edge is described in Algorithm 4.

This method is capable of conflict detection and the recognition of multipoints on the same line, thus avoiding the occurrence of odd number of intersection points on a line.

4.2.4. Reverse Solution of the Light Vector. In the reverse ray tracing, each pixel on the screen needs a reverse solution of the ray vector.

By the nature of surface projection, the distance from a point to the screen center is linearly correlated with the angle between the vector corresponding to the point and the sight axis camAZ. Accordingly, for a point (sx, sy) on the screen, the radius of the screen was assumed as R, and the field angle as θ. Then, the relation between the coordinates of the point and those of the screen center was $dx = sx - R$ and $dy = R - sy$. The angle β between its vector and the sight axis was calculated by

$$\beta = \frac{\sqrt{dx^2 + dy^2}}{2R}\theta. \tag{17}$$

After the normalization of both the ray vector and sight axis camAZ, its module became 1. The characteristics of the vector inner product indicated that the axis Z's component Lz of the ray vector could be calculated by

$$Lz = \cos(\beta). \tag{18}$$

The module of the normalized ray vector was 1, as mentioned above. Next, the components of axis X and axis Y of the ray vector were assumed as Lx and Ly, respectively. According to the calculation of vector modulus, (19) could be expressed as

$$\sqrt{Lx^2 + Ly^2 + Lz^2} = 1. \tag{19}$$

With the ratio of Lx to Ly being denoted as k, (20) could be derived based on projection characteristics as

$$k = \frac{Ly}{Lx} = \frac{dy}{dx}. \tag{20}$$

Then, (19) was embedded into (20) to form (21), described as

$$Lx = \sqrt{\frac{1 - Lz^2}{1 + k^2}}. \tag{21}$$

In summary, the ray vector $L(Lx, Ly, Lz)$ was calculated by

$$\begin{cases} Lx = \sqrt{\dfrac{1 - \cos^2\beta}{1 + (dy/dx)^2}}, \\ Ly = \dfrac{dy}{dx}Lx, \\ Lz = \cos\beta. \end{cases} \tag{22}$$

Nonetheless, this calculated ray vector L was still within FaceMap space, thus needed to be converted to a world coordinate system. Therefore, the ray vector of the world coordinate system was denoted as T, and (23) was obtained as

$$\begin{bmatrix} \text{camAX} \\ \text{camAY} \\ \text{camAZ} \end{bmatrix} [T] = [L]. \tag{23}$$

The value of the determinant was calculated according to Cramer's rule, as shown as follows:

$$\begin{aligned} D &= \begin{vmatrix} \text{camAX}.x & \text{camAX}.y & \text{camAX}.z \\ \text{camAY}.x & \text{camAY}.y & \text{camAY}.z \\ \text{camAZ}.x & \text{camAZ}.y & \text{camAZ}.z \end{vmatrix}, \\ D1 &= \begin{vmatrix} L.x & \text{camAX}.y & \text{camAX}.z \\ L.y & \text{camAY}.y & \text{camAY}.z \\ L.z & \text{camAZ}.y & \text{camAZ}.z \end{vmatrix}, \\ D2 &= \begin{vmatrix} \text{camAX}.x & L.x & \text{camAX}.z \\ \text{camAY}.x & L.y & \text{camAY}.z \\ \text{camAZ}.x & L.z & \text{camAZ}.z \end{vmatrix}, \\ D3 &= \begin{vmatrix} \text{camAX}.x & \text{camAX}.y & L.x \\ \text{camAY}.x & \text{camAY}.y & L.y \\ \text{camAZ}.x & \text{camAZ}.y & L.z \end{vmatrix}. \end{aligned} \tag{24}$$

The calculation of ray vector T in the world space was shown as follows:

Input: A curve information
(1) Plot function of space curve (Input the two vertices of the curve):
    (1) Project the two vertices onto the screen space as two endpoints
    (2) Add the first endpoint
    (3) **if** the distance between the two endpoints is longer than one pixel:
        (i) Find the midpoint of the two vertices
        (ii) Curve approximation function (the first vertex and the midpoint)
        (iii) Add the endpoint after the midpoint projection
        (iv) Curve approximation function (the midpoint and the second vertex)
    (4) **else** Add the second endpoint
(2) Curve approximation function (input the two 3D vertices):
    (1) **if** the distance between endpoints after vertex projection is longer than one pixel:
        (i) Find the midpoint of the two vertices
        (ii) Curve approximation function (the first vertex and the midpoint)
        (iii) Add the endpoint after the midpoint projection
        (iv) Curve approximation function (the midpoint and the second vertex)
    (2) **else** null
Output: Results after mapping

Algorithm 2: Screen space curve mapping algorithm.

Input: All point information of curve triangle
(1) Find the highest and lowest points of the triangle;
(2) Traverse from the highest point to the lowest point:
    (a) Traverse from the highest point to the lowest point: Calculate all intersection points of the current line and the triangle edge;
    (b) For every two intersect points: Fill from the left node to the right node;
Output: Results after filling

Algorithm 3: Filling algorithm of the curve triangle.

Input: A triangle vertex information
    Traverse all points on a curve:
        if the current point is online y and the next point is below line y:
            Add the x-coordinate of the current point
        else null
        if the next point is online y and the current point is below line y:
            Add the x-coordinate of the next point **else** null
Output: Intersection information

Algorithm 4: Algorithm for calculating the intersection of each line and curve edge of curve triangle.

$$\begin{bmatrix} T.x \\ T.y \\ T.z \end{bmatrix} = \frac{1}{D} \begin{bmatrix} D1 \\ D2 \\ D3 \end{bmatrix}. \tag{25}$$

In accordance with Cramer's rule, only one solution exists when D is greater than 0; innumerable solutions exist when D equals 0; no solution exists when D is less than 0. Geometrically, (23) is similar to the intersection operation of three planes, with the normal vectors of the three planes being camAX, camAY, and camAZ, respectively, serving as three sight axes of FaceMap. Therefore, the three normal vectors would certainly intersect at a point. Moreover, since the sight axes were normalized unit vectors with modules of 1 and the value of D was also calculated as 1, (25) was simplified as

$$T = \begin{bmatrix} D1 \\ D2 \\ D3 \end{bmatrix}. \tag{26}$$

4.3. Renderer Design Based on FaceMap Algorithm. FaceMap, being abstract in the program, can be inherited by other components to obtain the panorama distribution of the scene model. Then, FaceMap can be used or modified

according to specific needs. The structure of Renderer based on the FaceMap algorithm is shown in Figure 12:

(1) Vector class contains attribute coordinate values x, y, and z and basic vector operations such as addition, subtraction, point multiplication, and cross multiplication

(2) Face class is used for storing basic triangular surfaces, involving vertex arrays with a length of 3 and normal arrays with a length of 3

(3) OBJ class stores model data and helps to load Obj model files which are then converted to face object arrays

(4) Space vector class represents a basic space point, with attributes including space position, rotation angle, and self-space coordinate axis

(5) FaceMap class, deriving from space vector, is also abstract. However, compared with space vector, FaceMap contains additional attributes including field angle and field radius

(6) Camera inherits FaceMap. While generating FaceMap structure, it only stores the information of surfaces closest to itself, and then, the closest intersection point can be directly calculated during rendering

(7) Point Light also inherits FaceMap. While generating the FaceMap structure, it stores the distribution data of all surfaces. Furthermore, the point light class provides the method of get_lighten_up (). It judges by virtue of its FaceMap whether an entered space point will be blocked by other surfaces, namely whether the point is in shadow

(8) Color class is used for color operation and contains the values of three components R, G, and B. While rendering a certain point, shadow test of multiple light sources should be performed. If can be irradiated, the value of illumination color should be accumulated. Color class provides several basic methods including addition, subtraction, numerical multiplication, and conversion to 24 bit color

(9) Image class is a two-dimensional image buffer, storing scene graphs for ray-tracing rendering; [10] Renderer class controls the whole Renderer, which contains OBJ model, Camera, Point Light array, and Image buffer.

5. Experimental Results and Analysis

5.1. Experimental Environment. The computer configuration used in the experiments is as follows:

System: Windows 7 Ultimate
Processor: Intel Core i7-4710 HQ 2.50 GHz
Memory: 12 GB

5.2. Comparison Methods Introduction. In order to reflect the superiority of the performance of this method, this paper selects the more mature products on the market for comparative experiments. Apart from the proposed Renderer, Rhino 5 and VRay 2.00.02 were also chosen for the comparative experiment. Rhino, introduced by American company Robert McNeel in 1998, is a 3D modeling software based on NURBS (nonuniform rational B-spline). It has been widely used in 3D animation, industrial manufacturing, scientific research, mechanical design, and other fields [15]. VRay, produced by chaos group and ASGVIS Company, is high-quality rendering software and one of the most popular rendering engines in the industry [16].

5.3. Selection of the Rendering Model in Comparative Experiment. A total of four object rendering scenes was used in the comparative experiment, with detailed data of the scene as shown in Table 1.

5.4. Experimental Results and Analysis. Rhino 5, VRay 2.00.02, and Renderer presented in this paper were employed successively to render the four models, and the corresponding effects are shown in Figures 13–16, mainly from the rendering effect for comparison.

Judging from the rendering effects as reflected in Figures 13–16, the effects achieved by the proposed Renderer were similar to those by Rhino and VRay, with main differences lying in the deformation at the edge of the visual field and the brightness. Firstly, in terms of the deformation at the field edge, since surface projection was used in the proposed Renderer, the reserve resolution of the ray vector was also based on surface distribution. As shown from the effects of rendering chessboard in Figure 13 and typewriter in Figure 16, lines close to the camera were slightly bent when using Renderer, while they were straight in the results obtained by Rhino and VRay. Consequently, the rendering effects by the proposed Renderer were closer to those by a real camera. The second inference consisted of the brightness of illumination. Distinct illumination models used by different renderers resulted in diverse calculations of brightness and propagation attenuation of the light source, thus causing differences in brightness and contrast degree of the whole image. Nonetheless, the difference in brightness only led to various displaying effects, with a negligible impact on computational cost.

The specific time consumption of each rendering shown in Figures 13–16 is listed in Table 2.

Table 2 demonstrates that the proposed Renderer spent the least time in total, less than half the total time spent by either of the other two renderers. The computational efficiency of Renderer was 2.34 times that of Rhino and 2.15 times that of VRay 2.0. It is worth noting that the rendering by Renderer was completed in a single-thread environment, indicating an even lower CPU resources occupancy and a higher speed, while Rhino and VRay 2.0 used multithread rendering.

In summary, in this study, the proposed FaceMap-based ray-tracing algorithm, Rhino, and VRaywere successively used to render four object models including chessboard, grass patch, bust, and typewriter. The comparison in terms of rendering effect and rendering time indicated that the

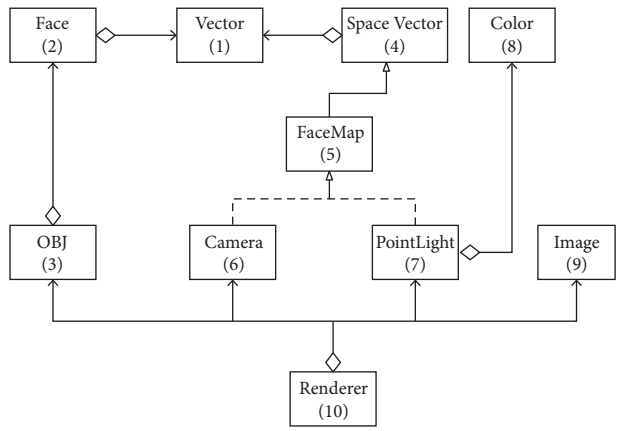
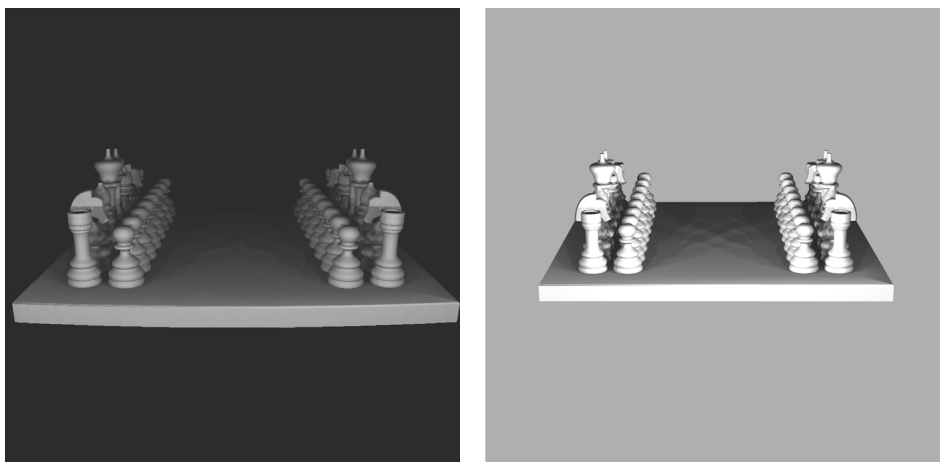


Figure 12: Renderer structure based on FaceMap algorithm.

Table 1: Model data.

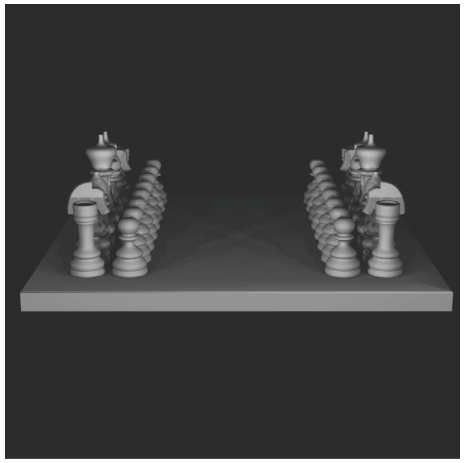
Scene name	No. of vertices	No. of surfaces	Size (MB)
Chessboard	61749	123314	11.4
Grass patch	52600	86864	6.93
Bust	22299	44590	4.01
Typewriter	73920	145558	11.3

Each scene was rendered in white mode with the same camera view, and eight point light sources were added to the scene.



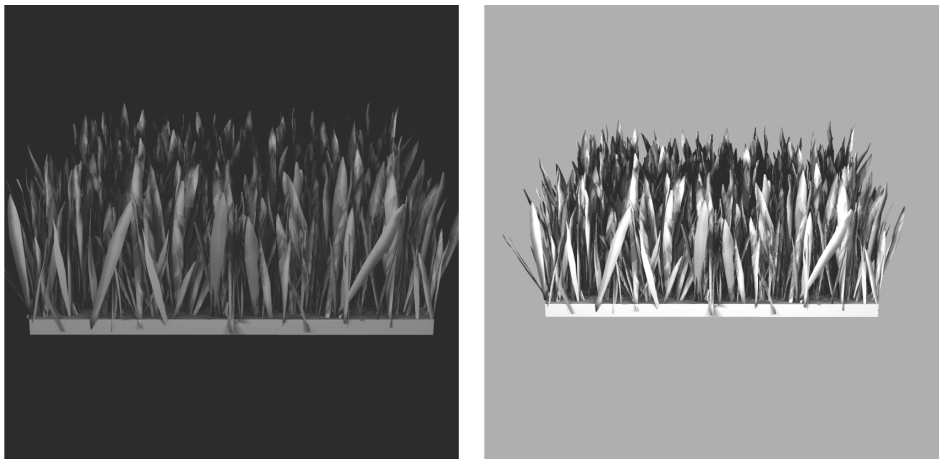
(a) (b)

Figure 13: Continued.

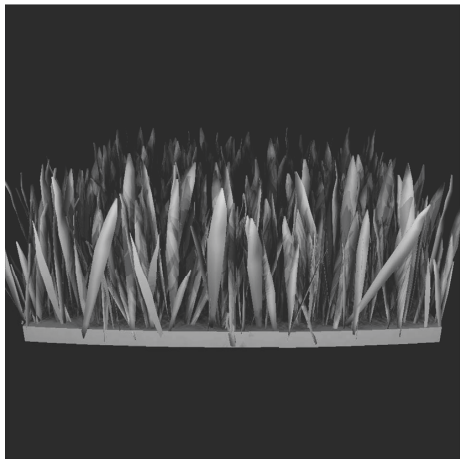


(c)

Figure 13: Effects of rendering chessboard. (a) Rendering effect by Rhino. (b) Rendering effect by VRay. (c) Rendering effect by Renderer (our method).

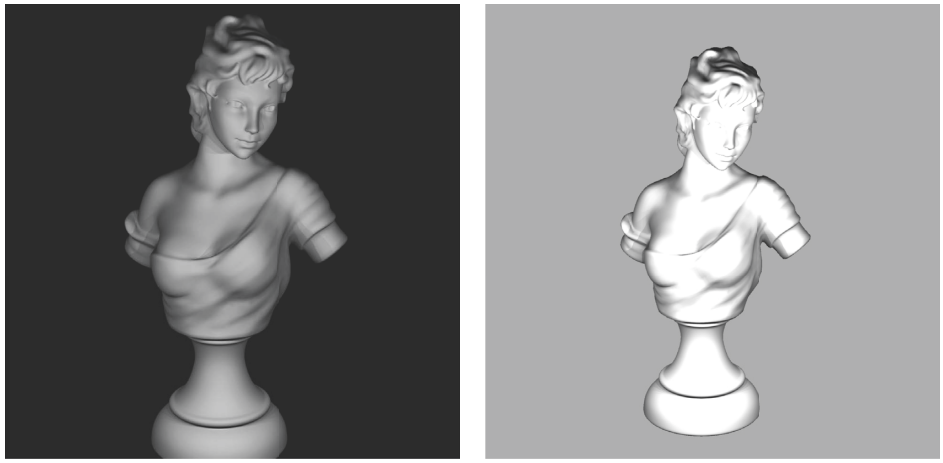


(a) (b)



(c)

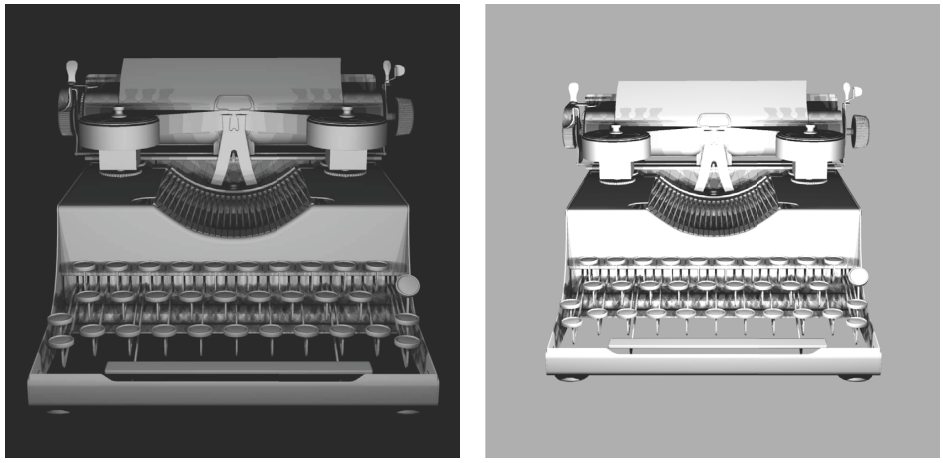
Figure 14: Effects of rendering grass patch. (a) Rendering effect by Rhino. (b) Rendering effect by VRay. (c) Rendering effect by Renderer.



(a) (b)

(c)

Figure 15: Effects of rendering a bust. (a) Rendering effect by Rhino. (b) Rendering effect by VRay. (c) Rendering effect by Renderer (our method).



(a) (b)

Figure 16: Continued.

(c)

Figure 16: Effects of rendering typewriter. (a) Rendering effect by Rhino. (b) Rendering effect by VRay. (c) Rendering effect by Renderer (our method).

Table 2: Time consumption of each rendering.

Renderer	Rendering time (s)				Total time
	Chessboard	Grass patch	Bust	Typewriter	
Rhino 5	2.2	25.2	7.1	45.2	79.7
VRay 2.0	4.7	25.6	9.2	33.7	73.2
Renderer	7.3	11.0	4.8	11.0	34.1

proposed method achieved relatively better rendering effects and significantly reduced the rendering time in the meanwhile.

6. Conclusion

This study intends to solve the shortcomings of the poor rendering effect of the raster method and the high computational complexity caused by the ray-tracing method. We propose a ray-tracing acceleration algorithm based on FaceMap by combining the raster method and ray-tracing method. FaceMap can be defined as a data structure that stores the surface distribution corresponding to a point light source or a camera. With the help of FaceMap, one can quickly determine the surfaces that may intersect in a certain direction, which can help improve the efficiency of intersection operation and thus speed up the overall rendering task and reduce computational complexity.

The novelty of this study and the advantages of the proposed method can be described as follows: (i) it can directly determine the data of a surface in a certain direction, requiring no calculation of light propagation process; (ii) the camera can rapidly determine intersection points based on FaceMap, which in turn greatly improved the speed of rendering operation; (iii) once generated, the FaceMap corresponding to a specific point light source can be reutilized, thus further reduces computational complexity.

To sum up, based on the theory of rendering acceleration and optimization, this paper proposes a new structure representation and corresponding projection algorithm, which provides a new research angle and an efficient technology for the actual rendering requirements and has a promising application prospect in the field of computer graphics. In spite of what stated, the present study is not without limitations, which can be addressed in future research: (i) FaceMap consumes relatively large memory and considers compression as a feasible solution; (ii) the texture and the material of rendered object have not yet been considered by the current model; (iii) the shadow processing caused by light conditions also needs to be further improved.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61462054) and the Science and Technology Plan Projects of Yunnan Province (2015FB135).

References

[1] H. W. Jensen and N. J. Christensen, "Photon maps in bidirectional Monte Carlo ray tracing of complex objects," *Computers & Graphics*, vol. 19, no. 2, pp. 215–224, 1995.

[2] J. Gao, K. Xu, and J. Cui, "An algorithm based on bounding box technology to improve the light and the objects intersection efficiency," *Journal of Traffic information and security*, vol. 22, no. 6, pp. 65–68, 2004.

[3] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Eurographics*, vol. 87, no. 3, pp. 3–10, 1987.

[4] W.-x. Wang, S. Xiao, M. Wen, and H. Dong, "Ray tracing algorithm based on octree space partition method," *Journal of Computer Applications*, vol. 28, no. 3, pp. 656–658, 2008.

[5] H. Weghorst, G. Hooper, and D. P. Greenberg, "Improved computational methods for ray tracing," *ACM Transactions on Graphics*, vol. 3, no. 1, pp. 52–69, 1984.

[6] A. Fujimoto and K. Iwata, "Accelerated ray tracing," *Computer Graphics*, vol. 85, pp. 41–65, 1985.

[7] A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: accelerated ray-tracing system," *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16–26, 1986.

[8] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4, pp. 269–278, 1986.

[9] J. G. Cleary, B. Wyvill, G. M. Birtwistle, and R. Vatti, *Multiprocessor Ray Tracing*, Deptartment of Computer Science University of Calgary, Calgary, Canada, 1983.

[10] A. S. Glassner, "Space subdivision for fast ray tracing," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 15–24, 1984.

[11] M. A. J. Sweeney and R. H. Bartels, "Ray tracing free-form B-spline surfaces," *IEEE Computer Graphics and Applications*, vol. 6, no. 2, pp. 41–49, 1986.

[12] E. Haines and D. Greenberg, "The light buffer: a shadow-testing accelerator," *IEEE Computer Graphics and Applications*, vol. 6, no. 9, pp. 6–16, 1986.

[13] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[14] S. M. Rubin and T. Whitted, "A 3-dimensional representation for fast rendering of complex scenes," *ACM SIGGRAPH Computer Graphics*, vol. 14, no. 3, pp. 110–116, 1980.

[15] https://baike.baidu.com/item/rhino/4065831?fr=aladdin.

[16] https://baike.baidu.com/item/vray/894350?fr=aladdin.