The Implementation of Ray Tracing Global Illumination Basing on JavaScript

Zhang Hao[1], **Du Xiaorong and Wang Beishan**

Abstract. Global illumination rendering is one of the pivotal techniques necessitated by 3D scenes' simulating; thus the quality of global illumination renderer would significantly affect the reality of such a simulation. Based on ray tracing, one of the major algorithms which aim to achieve great and realistic rendering, a relatively simple global illumination renderer is constructed via JavaScript supplemented by HTML5. Such a renderer can be used to render a comparatively verisimilar scenograph which is capable of displaying the geometrical optical phenomena engendered by reflection, refraction and shadow. Since this renderer is purely constructed via the standard programming language on Web, JavaScript, without any additional plug-ins or libraries, the program, theoretically, is able to fulfill the task of rendition of a given 3D scene in any personal computers installed with a mainstream browser.

Keywords: Realistic rendering · Global illumination · JavaScript · Ray tracing

1 Introduction

Ray tracing is a specific algorithm in order to achieve global illumination whose lights are radiated, for improving the efficiency of ray casting, from a camera rather than a light source, thereby externalizing a well-designed model of 3D-scenes. Ray tracing algorithm, one of the major rendering algorithms in the global-illumination realm, is of relatively simple theories, easy accomplishment and ability to generate diverse types of verisimilar pictures with high quality. In the former studies, when investing the algorithm to eliminating hidden surfaces in 1968, Appel. A proposed the description of ray tracing algorithm[1]. In 1979, Kay and Greenberg furthered this description to embrace the consideration of light refraction. One year later, Whitted became the first one to propose a practical model of global illumination, named Whitted model, and presented the general paradigm of how to achieve ray tracing algorithm including consideration of reflection, refraction as well as transmission and shadow.

The fact that the advent and boom of information high ways avails huge amount of the data with convenience and speediness becomes a common impetus that motivates many Internet enterprises to conduct their research around 3D-images. A JavaScript API for rendering interactive 3D graphics, WebGL[2], is formulated by their efforts.

[1]Zhang Hao
School of Physics and Engineering, Sun Yat-sen University, Guangzhou Guangdong, China

Du Xiaorong (✉)
Institute of Power Electronics & Control Technology, Sun Yat-Sen University, Zhuhai Guangdong, China
e-mail: duxr@mail.sysu.edu.cn

Wang Beishan
School of Physics and Engineering, Sun Yat-sen University, Guangzhou Guangdong, China

As a web language, JavaScript not only provides a friendly renderer in different mobile terminals, but also surmounts the traditional weaknesses of graphics delineated by XML documents with the help of the strong Canvas tag[3]. In this essay, the implementation of ray tracing algorithm is represented and recapitulated to convey some beneficial discussions about global illumination[4].

2 Theories of renderer Establishment

In this section, we first investigate the fundamental way to achieve a ray tracing algorithm, and then recapitulate the shading model we adopted, Phong illumination model. A description of a spot light model is followed by a specific approach towards improving the efficiency of ray casting. In the final part, a way to achieve camera positioning is presented and further discussed.

2.1 Achievement of Ray Tracing

A fundamental ray tracing algorithm needs to take into account refraction and reflection affecting interactive 3D images[5]. Light rays emitted from the source intersect the surface of an object and then are bent by refraction or deflected by reflection. Consequently, light rays would follow their new paths until they intersect another object and repeat the similar process mentioned above. As only a percentage of rays will be observed by cameras, the tracking direction distinguishes itself from natural ray direction with an inverse orientation, which means the origin of rays is the camera instead of a light source. This process can be clearly described in details in **Fig. 1.**

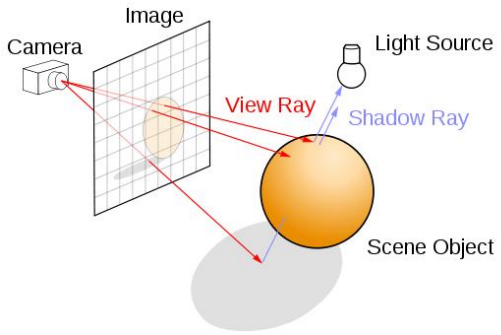


Fig. 1 A light ray is emitted from the camera, across a pixel on the image surface, travelling until it intersects a scene object. After this process, another three types of rays, reflective, refractive and shadow rays, are potentially produced

If the shading model of intersection points is reflective, tracing the new reflective ray is needed and so applied to refractive one. Differentiated the rays mentioned before, shadow rays are from intersection points and ends at light sources. If the path shadow rays intersect with an object, the shading of intersection points can be modified by transparency of those objects that block rays from light sources.

2.2 Phong Illumination

In this essay Phong illumination[6] is adopted to make a renderer. In this model, the shading of intersection points is jointly dependent on three types of rays generated by ambient, diffuse reflection of rough surfaces as well as the specular reflection of shiny surfaces, as shown in **Fig. 2**.

(a)Diffuse reflection (b)Specular reflection

Fig. 2 It illustrates how Phong reflection model describes the way a surface reflects light as a combination of the diffuse reflection and specular reflection

The Phong model can be described as:

$$\begin{cases} I_r = I_{ar} K_{ar} + I_{Pr} K_{dr} \cdot (\boldsymbol{L} \cdot \boldsymbol{N}) + I_{Pr} K_{sr} \cdot (\boldsymbol{N} \cdot \boldsymbol{H})^n \\ I_g = I_{ag} K_{ag} + I_{Pg} K_{dg} \cdot (\boldsymbol{L} \cdot \boldsymbol{N}) + I_{Pg} K_{gr} \cdot (\boldsymbol{N} \cdot \boldsymbol{H})^n \\ I_b = I_{ab} K_{ab} + I_{Pb} K_{db} \cdot (\boldsymbol{L} \cdot \boldsymbol{N}) + I_{Pb} K_{sb} \cdot (\boldsymbol{N} \cdot \boldsymbol{H})^n \end{cases} \quad (1)$$

Generally, the simple illumination only needs to consider light intensity of red, green and blue components to render pixel shading. This approach is appropriately called trichromatic system rendering which is adopted in this essay.

As usual, precisely considering the effects of shadow requires pre-light pass techniques[7] in which the effects of light sources ought to be rendered separately. When merely taking into account the direct shadow generated by light sources, yet, the transparency of the object, or objects if the shadow ray intersects with multiple elements, is the only factor contributing the shading of the intersection point. Therefore, the product of the transparency of the object and the original shading determines the real shading of the pixel.

2.3 Light Sources

Common light sources include collimated light, pointolite, spotlight and area light. Since it is not difficult to produce a model of these light sources[8], we only briefly introduce the model adopted of spotlight. On the basis of a pointolite, using a corresponding optical structure to guide and limit the lights towards a cone shape we get a special light source, called spotlight. There are diverse models describing spotlight and one of them is shown as **Fig. 3**.

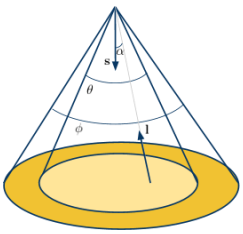


Fig. 3 A model of spotlight

As for the model mentioned above, the light cone consists of two different areas, called an inner cone and outer cone respectively. The interior angle of the inner cone is indicated by θ and Φ. The range of a coefficient describing the gap areas is from zero to one. The value, one, represents the lightest intensity. This coefficient dwindles

from 1 to 0 when approaching the outer boundary of the cone. Another parameter, p, is used to determine the rate of attenuation of the light intensity.

$$spot(\alpha) = \begin{cases} 1 & cos\alpha \geq cos\frac{\theta}{2} \\ (\frac{cos\alpha - cos\frac{\Phi}{2}}{cos\frac{\theta}{2} - cos\frac{\Phi}{2}})^p & cos\frac{\Phi}{2} < cos\alpha < cos\frac{\theta}{2} \\ 0 & cos\alpha \leq cos\frac{\Phi}{2} \end{cases} \quad (2)$$

2.4 The Algorithm of Basic Element Intersection

The general way to calculate intersecting points of a light ray and a quadric surface is already fully discussed[9] and so are the specific ways targeting on any of them[10]. In this essay, we narrow our focus on how to achieve a simple and effective algorithm to calculate the intersecting point of light rays and polyhedral surfaces, in which the more complex bounding box technology is unnecessary.

When an object is made up of several polygons, then calculating the intersection of rays and polyhedron is to be transformed into calculating the intersection of rays and polygons. Traditionally, in order to avoid the sampling in empty voxel and greatly improve the efficiency of ray casting, boundary box techniques are often utilized[11]. In this essay, we just apply what we learn from linear algebra to the same problem instead of using the traditional techniques and we will get an easy path towards our goal.

Since each polygon can be divided into several triangles, we only need to discuss the way adopted with a triangle $\Delta P_1P_2P_3$. First, consider a situation that the ray origin, O point (a light source or a intersection point where new rays are generated), is not on $\Delta P_1P_2P_3$(the excluded situation need some further but brief discussion and therefore are not mentioned in this essay).

The expression of the ray is indicted as:

$$\boldsymbol{d} = l\boldsymbol{OP_1} + m\boldsymbol{OP_2} + n\boldsymbol{OP_3} \quad (3)$$

In the expression, l, m and n are the components of the vector $\boldsymbol{d}$ using the basic vectors $\boldsymbol{OP_1}$, $\boldsymbol{OP_2}$, $\boldsymbol{OP_3}$.

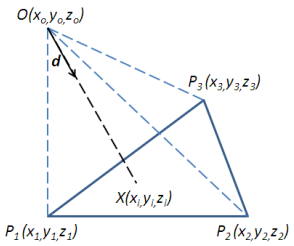


Fig. 4 Linear expression of $\boldsymbol{d}$

Combined with **Fig. 4**, we can expand the expression to:

$$\begin{cases} (x_1 - x_o)l + (x_2 - x_o)m + (x_3 - x_o)n = d_1 \\ (y_1 - y_o)l + (y_2 - y_o)m + (y_3 - y_o)n = d_2 \\ (z_1 - z_o)l + (z_2 - z_o)m + (z_3 - z_o)n = d_3 \end{cases} \quad (4)$$

Using Cramer's rule we have:

$$\Delta = \begin{vmatrix} x_1 - x_o & x_2 - x_o & x_3 - x_o \\ y_1 - y_o & y_2 - y_o & y_3 - y_o \\ z_1 - z_o & z_2 - z_o & z_3 - z_o \end{vmatrix}$$

$$\text{Also: } \Delta_x = \begin{vmatrix} d_1 & x_2 - x_o & x_3 - x_o \\ d_2 & y_2 - y_o & y_3 - y_o \\ d_3 & z_2 - z_o & z_3 - z_o \end{vmatrix} \quad \Delta_y = \begin{vmatrix} x_1 - x_o & d_1 & x_3 - x_o \\ y_1 - y_o & d_2 & y_3 - y_o \\ z_1 - z_o & d_3 & z_3 - z_o \end{vmatrix}$$

$$\Delta_z = \begin{vmatrix} x_1 - x_o & x_2 - x_o & d_1 \\ y_1 - y_o & y_2 - y_o & d_2 \\ z_1 - z_o & z_2 - z_o & d_3 \end{vmatrix} \qquad (5)$$

Because we consider a situation that O is not on the surface where $\Delta P_1P_2P_3$ is located, Δ would not be equal to zero. By discussing the interrelation of the signs of Δ, Δ_x, Δ_y and Δ_z, we can demonstrate that the rays intersect $\Delta P_1P_2P_3$ if and only if Δ has a different sign from all of Δ_x, Δ_y and Δ_z. Therefore, we can avoid calculating the coordinates of those intersection points that are not on $\Delta P_1P_2P_3$ by initially making a judgment on the signs of Δ, Δ_x, Δ_y and Δ_z.

2.5 Positioning Camera

One of the advantage of combining HTML5 and JavaScript to write a program is its convenience of achieving data exchange—users only need to click a labeled button if they want to change the position of the camera or other parameters of the object elements. In order to keep the focus upon the scenes, it is required to rotate the directional axis of the camera when it has been moved. Using the rotational matrix of Euler Angle we are able to achieve this goal[12], as indicated in (6) and **Fig. 4**.

$$R_{xyz}(\alpha,\beta,\gamma) = R(Z,\alpha)\cdot R(Y,\beta)\cdot R(X,\gamma) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}\cdot$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\alpha\cos\beta & -\sin\beta \\ \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\beta\sin\gamma \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \qquad (6)$$

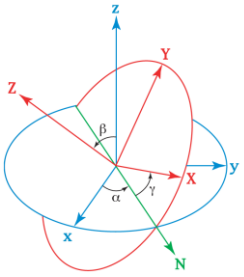


Fig. 4 An Euler Angle illustration

3 the Renderer Interface and Illustrations

The renderer interface on the basis of JavaScript with several labelled buttons, available to set or change the optical parameters of objects and the positions of cameras, is presented as **Fig. 5**.

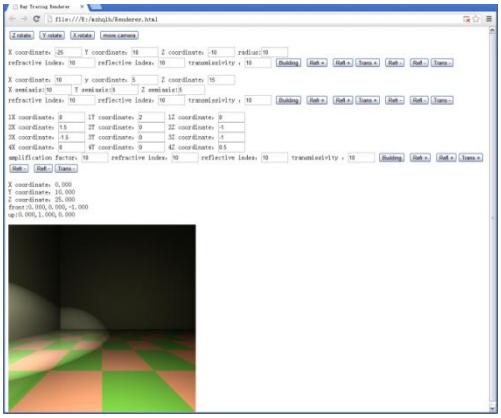
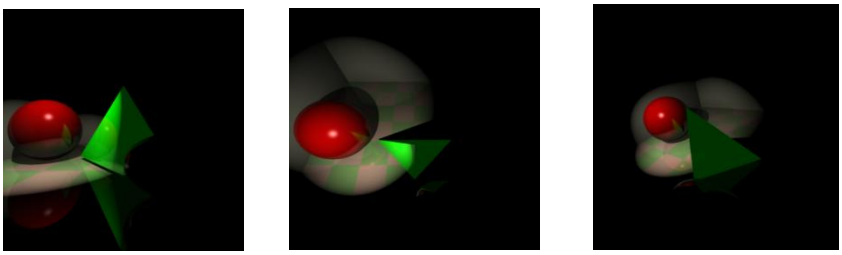


Fig. 5 The renderer interface constructed

Using rotating labels to shift the perspective of cameras, we can get different pictures as **Fig. 6**.



(a) front view (b) 60° top view (c) 45° side view

Fig. 6 Pictures produced from different perspectives

Using geometric optics labels to change the refractive index of a rendered picture, we can get another picture as presented in **Fig. 7**.

(a) $n_{ellipse} = 1$ (b) $n_{ellipse} = 1.2$

Fig. 7 Two pictures with different elliptical refractive indexes

When mirrors are added as the two walls, we get an interactive 3D image as **Fig. 8.**

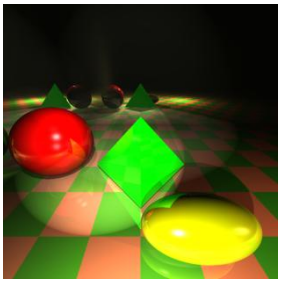


Fig. 8 One example of interactive 3D images

4 Conclusion

In this thesis, a ray-tracing renderer is successfully constructed basing on the combination of JavaScript and HTML5 without any additional plug-ins and libraries, which is capable of simulating basic optical phenomena including refraction, reflection and shadow. In addition to Canvas, a label providing great support to graphics, HTML5 also affords many other labels of friendly interfaces and convenient interaction. Therefore, it is extremely handy to change the optics parameters or shift the camera after the 3D graphics has been rendered, if users desire so.

However, such a renderer only succeeds in some basic and fundamental 3D figures and it is because of this that we can bypass the complex data structure. For that reason, expanding the shape pool of this renderer needs a radical change of the data structure. In the final analysis, although JavaScript programs are both user-friendly and programmer-friendly, when considering the number-crunching of ray tracing algorithm, we might have to resort to another language, such as C++ and Java to render an immersive image with innumerable objects.

5 Acknowledgment

This paper was supported by “The Guangdong Provincial Production Education and Research Major Projects under Grant No.201210558026” and “Zhuhai Science and Technology Plan for Production and Research under Grant No.2012D0501990030”.

References

1. Appel, A. (1968) Some techniques for shading machine rendering of solids. *AFIPS Spring Joint Computing conference*, 37-45

2. Xu Hui, Wei Lihao, L et al (2012) WebGL based HTML5 Application Performance Analyzer. *Journal of Convergence Information Technology*, 7(23), 280-289

3. Marshall, Brandeis. (2010) Taking the tags with you: Digital photograph provenance, *2nd International Symposium on Data, Privacy and E-commerce*. doi: 10.1109/ISDE.2010.18

4. Antal, G., Martinez, R., Csonka, F,. Sbert, M., Szimay-Kalos, L. (2003) Combining global and local global-illumination algorithms. *Spring Conference on Computer Graghics*, 185-192

5. Milo, Yip. (2010) Learning more about computer graphics by JavaScript: Ray Tracing. http://www.cnblogs.com/miloyip/archive/2010/03/29/1698953.html. Cited 15 March 2010

6. Phong, B.T. (1975) Illumination for Computer-generated Pictures. *Comm.ACM*, 18(6), 311-317

7. Michael, J. (2009) Light Pre Pass in XNA: Basic Implementation. http://mquandt.com/blog/2009/12/light-pre-pass-in-xna-basic-implementation. Cited 21 May 2009

8. Milo, Yip. (2010) Learning more about computer graphics by JavaScript: Basic light sources. http://www.cnblogs.com/miloyip/archive/2010/04/02/1702768.html. Cited 3 April 2010

9. Sun, J.G., Hu, S.M. (2009) *Basics of Computer Graphics, 2rd edn*. Tsinghua University Press, Beijing

10. Cheng, Y., Li, Y., Cong, W. (2000) the Analysis of Ray Tracing Algorithms. *Shenyang Institute of Aeronautical Engineering Journal*, 17(2), 10

11. Rubin, S. M., Whitted, T. A. (1980) A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3), 110-116

12. Shi, M. (2004) the Analysis of basic movement of a virtual human in three-dimensional terrain environment. PHD thesis, North China Electric Power University