School of Informatics

Informatics Research Review
Two Ray Tracing Methods in 3D Rendering

[redacted]

January 2021

Abstract

One of the most important tasks for nowadays computer graphic researchers is rendering high-quality photo-realistic imagery. Among all existing rendering algorithms, ray-tracing methods best simulate the behavior of light and are widely implemented in offline rendering tasks. This paper reviews two famous ray-tracing algorithms: the Whitted-style ray-tracing algorithm and the path-tracing algorithm. The classic Whitted-style ray-tracing algorithm is the first mature ray-tracing rendering method. The path-tracing algorithm is developed from Whitted-style ray-tracing and is the most commonly used ray-tracing method in today's game and film industry.

Date: Friday 22nd January, 2021

[redacted]

Contents

1 Introduction **1**

2 Literature Review **2**

2.1 Whitted-style ray-tracing 2

2.1.1 Casting and tracing rays 3

2.1.2 Blinn-Phong reflection model 3

2.1.3 Bounding volumes and BV-Tree 4

2.2 Monte Carlo path tracing 5

2.2.1 The rendering equation 5

2.2.2 Solving the rendering equation and path tracing 6

3 Summary & Conclusion **7**

1 Introduction

Today, physically based rendering[1] technologies are widely used in the 3D game and film CG industry. As customers' demand for the reality and shading quality of computer-generated pictures grows higher and higher, traditional rendering methods like rasterization[2] no longer meet the needs of generating photo-realistic imagery. In this circumstance, ray-tracing methods that simulate the behavior of light accurately become a preferable choice in rendering high-quality 3D scenes. Meanwhile, as the newest graphic hardware starts to provide specialized support for ray-tracing methods (Nvidia's newest GPUs have RT cores to accelerate ray-tracing algorithms), researchers and developers are now able to apply time-consuming ray-tracing methods in both offline and real-time rendering tasks[3]. From now to the next ten years, ray-tracing rendering will be the most popular subject in the computer graphics field.

The basic theory of ray-tracing is casting virtual rays into a 3D scene and trace these rays to gather color information of virtual objects in the scene. In ray-tracing, we assume rays and their transmission obey these laws:

1. rays travel in straight lines
2. rays do not "collide" with each other if they cross
3. rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity)

When doing ray-tracing rendering, we cast rays from the location of the virtual camera into a 3D scene, and each ray represents an image pixel. Among these rays, those who hit virtual objects return the color of the hitting point, and others return the background color of the scene. Using the color information gathered by rays, graphic hardware generate high-quality images.

The idea of casting rays into virtual scenes and using virtual rays to render images was first raised by Appel[4]. Then Goldsein and Nagel refined the theory and raised the idea of "visible surface"[5]. Inspiring as the "visible surface" theory is, it ignored the refraction and reflection of rays. In 1979, Turner Whitted first introduced a mature recursive ray-tracing method in his paper "An Improved Illumination Model for Shaded Display"[6]. Whitted successfully simulated the reflection and refraction of rays in a virtual scene and used his ray-tracing renderer generated photo-realistic images with higher quality than images generated by rasterization methods. We will introduce more details about this method in later chapters of this review.

Though Whitted-style ray-tracing is a rather important ray-tracing algorithm, it is a little bit old-fashioned. The most popular ray-tracing method being widely applied in today's high-quality rendering tasks is Monte Carlo path tracing[7]. Monte Carlo path tracing is developed from Whitted-style ray-tracing and is based on the rendering equation[8], an integral equation that can compute the radiance of a point in a 3D scene. The radiance (luminance) is a physical quantity in radiometry[9] that represents the intensity of light. Details about the rendering equation and related concepts like indirect illumination will be introduced in later sections of this review.

After getting the radiance of a point on a virtual object, we can use this radiance value with the object's texture information to compute the color of this point. After getting color information for all pixels, we will be able to render an accurate image of the whole 3D scene. But as we can see, the rendering equation is a recursive integral function and can not be solved by conventional methods. Actually, in path tracing, we often use Monte Carlo integration[10] to solve it. That is the reason why path tracing is also called Monte Carlo path tracing. Path tracing can render images with high physical correctness, and it is widely applied in offline rendering[11] tasks due to its correctness.

Actually, since we link each ray with a pixel in the picture we are going to render, a typical way of rendering an image with 1920*1080 pixels is to cast 1920*1080 rays into the scene and compute the complex light transport process and solve rendering equations at every ray-object intersection point. So, as we can see, ray-tracing methods are really time-consuming and are not often used in real-time rendering. But as graphic hardware becomes more and more powerful, the game industry shows great interest in implementing ray-tracing methods in real-time rendering tasks, such as using ray-tracing methods to render high-quality game scenes.

The last thing to mention is that this review will focus on ray-tracing methods. Basic computer graphics conceptions such as perspective projection and projection transformation[2] and traditional rendering algorithms such as rasterization will not be introduced.

2 Literature Review

2.1 Whitted-style ray-tracing

Whitted-style ray-tracing is one of the most classic ray-tracing methods and it was first raised by Turner Whitted in 1979. Whitted-style ray-tracing provides the basic strategy of casting and tracing virtual rays that influence all later ray-tracing algorithms.

Whitted-style ray-tracing can be divided into three parts: casting and tracing rays, ray-object intersection detection, and shading computing.

2.1.1 Casting and tracing rays

Before introducing rules of casting and tracing rays in Whitted-style ray-tracing, we assume rays and their transmission obey these laws:

1. rays travel in straight lines
2. rays do not "collide" with each other if they cross
3. rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity)

Together with the above three laws, Whitted also defined three kinds of object surfaces: opaque and diffuse surface, mirror-like surface, and transparent surface. Rays will not reflect or refract on diffuse surfaces, only reflect on mirror-like surfaces and will both reflect and refract on transparent surfaces.

The process of Whitted-style ray-tracing is given here:

> Step 1: For each image pixel, cast a ray into the scene from the position of the virtual camera.
>
> Step 2:
>
> > Case 1: If the ray hits an opaque and diffuse surface[11], then compute the color of the intersection point and return. The color computed here will be used in setting the color of the related image pixel.
> >
> > Case 2: If the ray hits an opaque and mirror-like surface, then cast a new reflected ray at the intersection point into the scene and conduct step 2 on this new ray.
> >
> > Case 3: If the ray hits a transparent surface, then cast a new reflected ray and a new refracted ray at the intersection point and conduct step 2 on both two new rays. Here we use Snell law to compute the direction of the refracted ray.
> >
> > Case 4: If the ray does not hit anything object in the scene, then return the preset background color.

After recursively computing the color information for each pixel, we can render an image of the scene.

2.1.2 Blinn-Phong reflection model

To compute the color of a point on the surface of an object, we have to first get the light intensity at this point. Whitted-style ray-tracing adopts Blinn-Phong reflection model[12][13] to compute the light intensity on ray-object intersection points. After getting the light intensity, we will be able compute the color of a surface point using this value together with related texture information.

Blinn-Phong reflection model divides the light intensity at a point into three parts: ambient light intensity, diffuse light intensity and specular light intensity.

The ambient part (I_a) is set to simulate the complex reflection caused by ambient light. In practical applications, the ambient light intensity is often set as a fixed value. Adding this fixed value into the shading model means adding color to account for disregarded complex background illumination in the scene.

The diffuse part of the shading model is given by:

$$I_d = k_d(I/r^2)max(0, n \cdot l) \tag{1}$$

where

k_d = diffuse coefficient

I = the intensity of the light source

r = euclidean between the shading point and the light source

n = shading point's normal vector

l = the direction of the light source

The specular part of the shading model is given by:

$$I_s = k_s(I/r^2)max(0, n \cdot h)^p \tag{2}$$

where

k_s = specular coefficient

h = unit vector in the direction halfway between the viewer and the light source[6]

p = an index to control the reflection lobe

The complete Blinn-Phong shading equation is given by:

$$I = I_a + I_d + I_s \tag{3}$$

Blinn-Phong reflection model only takes the intensity generated by light sources into account and ignores indirect illumination (for example, the light reflected between objects). Strictly speaking, Blinn-Phong model can only roughly estimate the light intensity at a space position and its output is just an approximate value. Although it is not physically correct, Blinn-Phong model is simple enough and performs well in practice applications. Apart from early ray-tracing methods, this model is also widely used in other rendering algorithms like rasterization.

2.1.3 Bounding volumes and BV-Tree

The most time-consuming process in the whole Whitted-style ray-tracing algorithm is ray-object intersection detection. In modern 3D rendering tasks, a virtual scene often has tens of thousands of triangle meshes and it is impossible to compute ray-object intersection detection on each triangle mesh. To accelerate the detection process, a structure named bounding volume is widely applied in ray-tracing algorithms.

Instead of directly detecting intersections between rays and triangle meshes, we can first divide all meshes into several subsets and build a bounding volume for each subset. The bounding volume fully contains every mesh inside it and before detecting intersections between rays and meshes, we can first detect intersections between rays and simple bounding volumes. If a ray does not hit the bounding volume, it will never hit inside meshes. Therefore, we can skip all meshes in the bounding volume and do much less ray-mesh intersection detection.

AABB[14], OBB[15], and k-dops[16] are three kinds of the most commonly used bounding volumes. Based on experience, AABB is always a good choice.

To further partition the scene and accelerate ray-mesh intersection detection, we can build a tree structure named BV-Tree[16] to store bounding volumes in the scene. Every leaf node of a BV-Tree stores a set of meshes and their bounding volume, and each internal node in BV-Tree stores the bounding volume of its child nodes. The root node of the tree is just associated with the full set of meshes in the scene. BV-Tree is an efficient search tree. By applying BV-Tree, we first compute the intersection between rays and bounding volumes of internal nodes. If a ray fails to hit the bounding volume of an internal node, it will also never hit the bounding volumes of this node's child nodes. Bounding volume structures and BV-Tree greatly simplify the intersection detecting process in ray-tracing algorithms and enable ray-tracing algorithms to run faster.

2.2 Monte Carlo path tracing

Although Whitted-style ray-tracing is a simple model and can generate better reflection and refraction effects in images than traditional render methods, it is still not accurate enough. Blinn-Phong shading model can only make rough assumptions to the light intensity and cannot take the influence of indirect illumination into account. In 1986, James Kajiya raised the rendering equation that can compute faithful radiance based on the law of the conservation of energy. After being introduced into computer graphics, the rendering equation was implemented in a new ray-tracing method called path tracing. Since path tracing uses Monte Carlo integration to solve the rendering equation, it is also called Monte Carlo path tracing.

With the help of the rendering equation, path tracing can generate complex global illumination and render photo-realistic images. Path tracing is often implemented in offline rendering tasks and is the most widely used ray-tracing method in today's film and CG industry. But one of its main drawbacks is that path tracing is even more time-consuming than traditional ray-tracing methods. Therefore, path tracing is still not often implemented in real-time rendering tasks.

2.2.1 The rendering equation

The rendering equation is given by[8]:

$$I(x,x^{'}) = g(x,x^{'})[\epsilon(x,x^{'}) + \int_S \rho(x,x^{'},x^{''})I(x^{'},x^{''})\mathrm{d}x^{''}] \tag{4}$$

where

$I(x,x^{'})$ is the radiance of the light passing from point $x^{'}$ to point x. Radiance is a physical quantity in radiometry[9] that represents the intensity of light.

$g(x,x^{'})$ represents the distance between x and $x^{'}$, and is usually computed as $1/||x-x^{'}||^2$.

$\epsilon(x,x^{'})$ is the radiance of the light emitted by the surface at point $x^{'}$ (if $x^{'}$ is on the surface of a light source). If the surface at point $x^{'}$ cannot emit light then $\epsilon(x,x^{'})$ will be set as zero in the equation.

$\rho(x, x^{'}, x^{''})$ is also called bidirectional reflectance distribution function, BRDF[19]. "$\rho(x, x^{'}, x^{''})$ is related to the intensity of light scattered from $x^{''}$ to x by a patch of surface at $x^{'}$."[8] In practice, ρ is always a set of preset values and is stored as a part of scene information.

The rendering equation is an integral equation. It divides the radiance at point x into two parts: direct illumination(the $\epsilon(x, x^{'})$ part) and indirect illumination(the integral part). The integral in the indirect illumination part takes over set S, which is the union of all surfaces in the scene. Therefore x, $x^{'}$ and $x^{''}$ in equation (4) are among all surfaces and all objects.

2.2.2 Solving the rendering equation and path tracing

Apparently, it is impractical to compute integral over all surfaces in a scene. Therefore we use Monte Carlo method to solve the integral equation and get an approximate solution.

Monte Carlo integration:

$$\int_a^b f(x)\mathrm{d}x \approx \frac{1}{N}\sum_{k=1}^{N}\frac{f(X_k)}{p(X_k)} \tag{5}$$

$$X_k \sim p(x)$$

Using Monte Carlo integration, we can change the original the original integral part:

$$\int_S \rho(x, x^{'}, x^{''}) I(x^{'}, x^{''})\mathrm{d}x^{''}$$

into a computable form:

$$\frac{1}{N}\sum_{i=1}^{N}[\frac{\rho(x, x^{'}, x^{''}) I(x^{'}, x^{''})}{p(x^{''})}] \tag{6}$$

Now the only task is to solve $I(x^{'}, x^{''})$ in equation (4). As we can see, the rendering equation is a recursively defined equation, and it takes countless indirect illumination into account. In practice we often set a maximum recursion depth D of the $I(x^{'}, x^{''})$ part in the equation. With a preset maximum depth D, we can write the pseudocode of our tracing algorithm:

```
shade(x, dir, depth)
{
    I_total = 0.0 //black
    if depth <= D:
        trace the ray r(x, dir)
        if r hits x_prime on a light source:
            I_total += epsilon(x, x_prime)
            return I_total;
        else if r hits x_prime on an object:
            randomly choose N directions dir_r ~ pdf
            for each dir_r:
                I_total += (1/N) * shade(x_prime, dir_r, depth+1) / pdf(dir_r)
        else if r hits nothing:
            return 0.0
    else if depth > D:
        return I_total
}
```

But this is still not standard path tracing. Standard path tracing typically only sample and trace one ray at every shading point(if N is greater than one, the method will be called as distributed ray-tracing[17]). Therefore, the pseudocode should be modified as:

```
shade(x, dir, depth)
{
    I_total = 0.0 //black
    if depth <= D:
        trace the ray r(x, dir)
        if r hits x_prime on a light source:
            I_total += epsilon(x, x_prime)
            return I_total;
        else if r hits x_prime on an object:
            randomly choose one direction dir_r
            I_total += shade(x_prime, dir_r, depth+1)
            return I_total
        else if r hits nothing:
            return 0.0
    else if depth > D:
        return I_total
}
```

The reason behind the modification is that if we sample and trace $N(N > 1)$ rays at each shading point, we will encounter an explosion of rays as the recursion goes on. Taking efficiency into consideration, it is reasonable to sample only one ray. Though the result may not be very numerically accurate, path tracing is still able to render images with considerably high quality.

3 Summary & Conclusion

Both Whitted-style ray-tracing and Monte Carlo path tracing have been widely applied in rendering 3D CG videos, films, and 3D games. Those two methods are the most famous ray-tracing methods. As the first mature ray-tracing framework, Whitted-style ray-tracing is a milestone in computer graphics. Whitted-style ray-tracing first implemented an algorithm of tracing light transport to complete high-quality rendering tasks. Compared with traditional scanline rendering methods like rasterization, Whitted-style ray-tracing is more precise and enables computers to render images in a physically correct way. But on the other hand, ray-tracing methods are typically much slower than those traditional rendering methods. The major contribution of Whitted-style ray-tracing is that it first provided a whole set of methods to trace rays and detect ray-mesh intersections in 3D scenes, simulate the reflection and refraction process, and compute color information for each image pixel.

Path tracing is developed from Whitted-style ray-tracing and is the first method that can compute indirect illumination and generate complex visual effects like soft shadow. With the help of the rendering equation and the theory of global illumination, path tracing achieved to render images in a faithful and accurate way. Path tracing is a popular choice in completing high-quality offline rendering tasks.

Powerful as it is, path tracing is still not a perfect rendering method. It is far more complex than Whitted-style ray-tracing and traditional rasterization methods. Therefore, building an accurate path tracing renderer will be a challenging task. Another main drawback is that path tracing is a time-consuming method and it is hard to be implemented on mobile platforms or handheld game consoles like Nintendo Switch. These devices' graphic hardware is too weak to support complex path tracing renderers. Therefore, many researchers are now trying to simplify path tracing algorithm and make it suitable for mobile devices.

Apart from the basic path tracing, researchers have raised refined ray-tracing models like light tracing[18] and bidirectional path tracing[19][20]. Other popular subjects include building BV-Tree structure in a faster way, designing proper bounding boxes that are both simple and space-saving (AABB bounding boxes are simple but waste a lot of space resources while k-dop bounding boxes are space-saving but rather complex), and finding better Monte Carlo sample strategies to solve the rendering equation more accurately (one refined strategy is sampling on the light instead of randomly sample the whole scene[25]).

Generally speaking, the main topic in this field is accelerating these slow algorithms. Most modern ray-tracing methods have already met the needs of correctness and versatility, and our goal is to make them faster and more efficient.

References

[1] Donald P Greenberg, Kenneth E Torrance, Peter Shirley, James Arvo, Eric Lafortune, James A Ferwerda, Bruce Walter, Ben Trumbore, Sumanta Pattanaik, and Sing-Choong Foo. A framework for realistic image synthesis. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 477–494, 1997.

[2] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics, Fourth Edition.* A. K. Peters, Ltd., USA, 4th edition, 2016.

[3] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering.* Crc Press, 2019.

[4] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45, 1968.

[5] Robert A Goldstein and Roger Nagel. 3-d visual simulation. *Simulation*, 16(1):25–31, 1971.

[6] Turner Whitted. An improved illumination model for shaded display. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, page 14, 1979.

[7] Eric Lafortune. Mathematical models and monte carlo algorithms for physically based rendering. *Department of Computer Science, Faculty of Engineering, Katholieke Universiteit Leuven*, 20:74–79, 1996.

[8] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[9] William Ross McCluney. *Introduction to radiometry and photometry.* Artech House, 2014.

[10] Russel E Caflisch et al. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 1998:1–49, 1998.

[11] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28, 1997.

[12] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[13] James F Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, 1977.

[14] Martin Held, James T Klosowski, and Joseph SB Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conference on Computational Geometry*, pages 205–210. Citeseer, 1995.

[15] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996.

[16] James T Klosowski, Martin Held, Joseph SB Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[17] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, 1984.

[18] Philip Dutre and Yves D Willems. Importance-driven monte carlo light tracing. In *Photorealistic Rendering Techniques*, pages 188–197. Springer, 1995.

[19] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. 1993.

[20] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, pages 145–167. Springer, 1995.