Ray Tracing

Ray tracing is a global illumination algorithm based on the emission of rays to calculate occlusion, i.e. to determine the visibility of three-dimensional objects from a certain point in space. Ray tracing also refers to several extensions of this fundamental method, which calculate the further path of rays after intersections with surfaces.

Ray tracing is most prominent in 3D computer graphics. Here, the basic ray tracing algorithm is a way of displaying a 3D scene. Extensions that simulate the path of light beams through the scene, as well as the Radiosity method, are used to calculate light distributions.

Other applications of the ray tracing algorithm include auralization (i.e. sound design) and high-frequency technology.

Before the development of ray tracing, computer graphics essentially consisted of a series of "programming tricks" that attempted to imitate the shading of illuminated objects. Ray tracing was the first algorithm in this field to make some physical sense.

Besides early uses of the physical principle by Albrecht Dürer as a painting technique, the first ray tracing image was displayed on an oscilloscope-like screen at the University of Maryland in 1963. [7]. The developers of the ray tracing algorithm are Arthur Appel, Robert Goldstein and Roger Nagel, who published the algorithm at the end of the 1960s [2, 14]. Other researchers working on ray tracing techniques at the time were Herb Steinberg, Marty Cohen and Eugene Troubetskoy [7]. Ray tracing is based on geometric optics, in which light is understood as a group of rays. The techniques used in ray tracing have been used much earlier, e.g. by lens manufacturers and artists. Today, many computer programs for creating images from a 3D scene use ray tracing, usually in combination with other methods.

Simple forms of ray tracing only calculate direct lighting, i.e. the light coming directly from the light sources. Ray tracing has been extended several times since it was first used in computer graphics. Higher-developed versions also take account of indirect light reflected by other objects; this is referred to as global illumination.

Basic principle

The generation of a raster image from a 3D scene is called rendering or image synthesis. This is preceded by the creation of such a scene by the user using a 3D modelling tool.

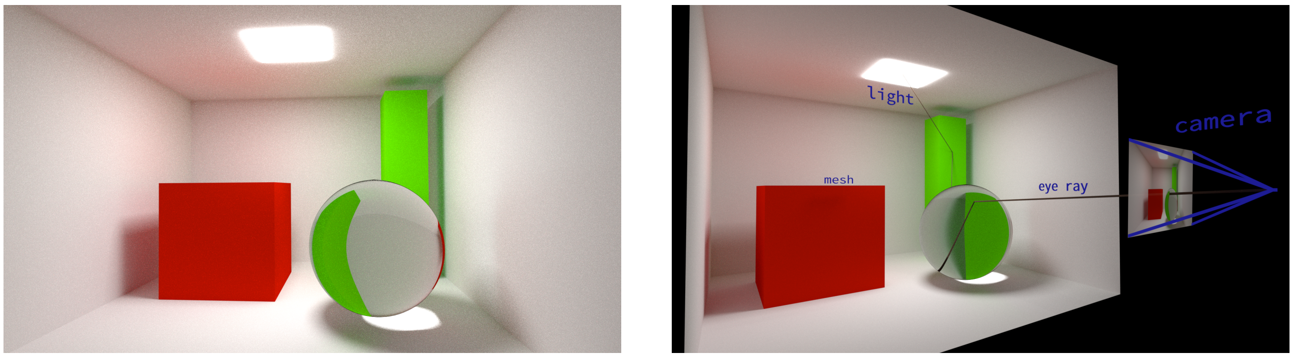
The scene description contains at least the following data:

- the position of the elementary primitives, such as polygons or spheres, from which the objects of the scene are composed;
- the local lighting models and their parameters, which determine the colours and materials of the individual objects in the scene;
- the scene's light sources.

In ray tracing, the position of an eye point and an image plane are also defined, which together indicate the perspective from which the scene is viewed. The eye point is a point in space that corresponds to the position of a virtual camera or a general observer. The image plane is a virtual rectangle located some distance from the eye point. It is the three-dimensional equivalent of the 2S raster image on a monitor to be rendered in space. Points distributed in raster form on the image plane correspond to the pixels of the raster image to be generated. A possible core ray tracing algorithm is outlined in Algorithm 1. The principle of this algorithm is outlined in Figure 1.

Algorithm 1 An example for a ray tracing algorithm.

```
for every pixel $\in$ raster image do
    Ray ray;
    ray.origin = fragment position in world coordinates
    ray.depth = 0
    ray.direction = interpolated value coming from vertex shader
    output.colour = 0
    for each intersection $<$ ray trace depth do
        intersection = intersect(ray,every object);
        if intersection.hit != 0 then
            output.colour += intersection.colour
            computeShadow(intersection); {extension}
        else
            break; {ray left scene}
        end if
        ray.origin = intersection.position
        ray.origin += epsilon * intersection.normal {avoid self shadow artefacts}
        ray.direction = reflect(ray.direction, intersection.normal);
    end for
end for
```



(a) (b)

Figure 1: (a) shows an adoption of a famous test scene (the Cornell box [4]) rendered using a ray tracing variant called path tracing. (b) shows the camera setup for this scene and one example viewing ray path corresponding to one pixel in the image plane. (images rendered by G. Jiang)

Computation of occlusion

Ray tracing is primarily a method for the calculation of occlusion, i.e. for determining the visibility of objects from the viewpoint.

Ray tracing works with a data structure, called ray, which indicates the starting point and the direction of a semi-straight line in space. The direction of the ray is calculated for each pixel, which points from the eye point to the corresponding pixel of the image plane. For each primitive of the scene, a geometric method is used to determine the possible intersection point at which the ray hits the primitive. If necessary, the distance from the eye point to the intersection point is calculated. The ”winner”, i.e. the primitive visible from the eye point, is the one with the smallest distance.

The principle of transmitting the rays from the eye point is similar to that of a pinhole camera, in which an object is imaged on a film. In ray tracing, however, “film” (picture plane) and “hole” (eye point) are reversed. Similar to the pinhole camera, the distance between image plane and eye point determines the “focal length” and thus the field of view.

Since the rays do not emanate from the light sources, as in nature, but from the eye point, we also speak of *Backward Ray Tracing*. Ray tracing deals with the question of where the light comes from. However, some publications refer to the procedure as *Forward Ray Tracing* or *Eye Ray Tracing*.

Intersection tests

The above mentioned test for a possible intersection of ray and primitive is the heart of the algorithm. Such tests can be formulated for a variety of primitive types. In addition to triangles and spheres, cylinders, quadrices, point clouds or even fractals are possible.

For spheres, the intersection point test is a relatively short and simple procedure, which explains the popularity of these objects for ray tracing test images. For reasons of simplicity, however, many rendering programs only allow triangles as primitives, from which any object can be approximately composed.

For Spheres any point on the surface is defined as

$$|q - p_s|^2 - r^2 = 0, \tag{1}$$

where r is the radius of the sphere. To test whether a ray intersects a surface we can substitute for q using the ray equation,

$$|p_0 + \mu d - p_s|^2 - r^2 = 0. \tag{2}$$

Setting $\Delta p = p_0 - p_s$ and expanding the dot product produces

$$\mu^2 + 2\mu(d \cdot \Delta p) + |\Delta p|^2 - r^2 = 0 \tag{3}$$

The quadratic equation has the solution

$$\mu = -d \cdot \Delta p \pm \sqrt{(d \cdot \Delta p)^2 - |\Delta p|^2 + r^2} \tag{4}$$

If the quadratic equation 4 has no solution, the ray does not intersect the sphere. If it has two solutions ($\mu_1 < \mu_2$): μ_1 corresponds to the point at which the rays enters the sphere, μ_2 corresponds to the point at which the rays leaves the sphere. This is further illustrated in Figure 3.

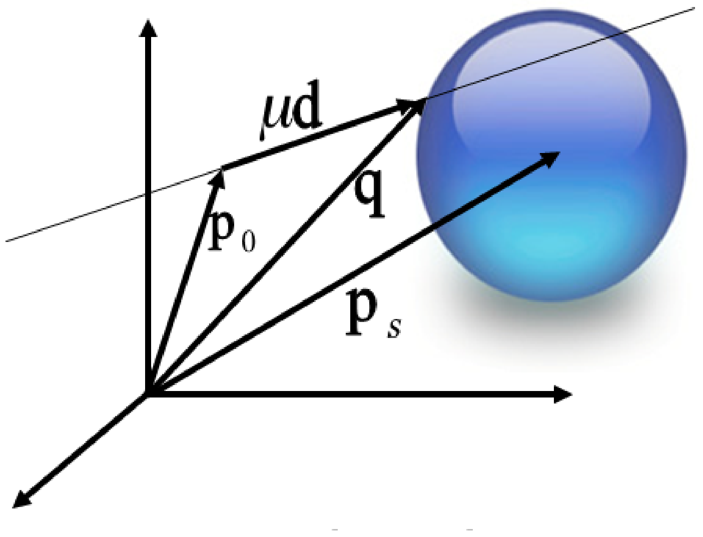


Figure 2: Intersection calculation with a sphere.

Triangles are most common for 3D scene definitions. Their intersections with viewing rays can be efficiently computing with algorithms similar to the intersection algorithm as proposed by Möller and Trumbore [].

More complex geometries such as NURBS have recently been used for intersection testing [1]. A maximum of precision is advantageous, since the surface is not divided into triangles as usual. The disadvantage is an increased render time, because the intersection point test with complex free-form surfaces is much more complex than with simple triangles. A sufficient approximation to the accuracy of NURBS is also possible with triangles, but in this case a very large number must be selected.

Shading

When determining the next primitive, not only the intersection point and its distance to the eye point is calculated, but also the normal of the primitive at the intersection point. Thus all information is available to determine the "luminous intensity" reflected to the eye point and thus the colour. The definition of the scene's light sources are also used. The calculations are based on local lighting models that simulate the material properties of an object. This part of the renderer that is responsible for determining the colour is called a shader.

Acceleration techniques

When determining the first primitive which a ray hits, each primitive of the scene can be tested against the ray, as described in Algorithm 1. However, this is not necessary if it is known that certain primitives are not in the vicinity of the ray and therefore cannot be hit. Since point of intersection tests require the longest runtime for ray tracing, it is important to test as few primitives as possible against the ray in order to keep the overall runtime low.

During several acceleration procedures, the scene is split up automatically in some form and the primitives are assigned to these subdivisions. When a ray passes through the scene, it is not tested against the primitives, but first against the subdivisions. This means that the ray only has to be tested against the primitives of the subdivision that crosses the ray.

A variety of such acceleration techniques have been developed for ray tracing. Examples of subdivision schemas are voxel grids, BSP trees, and bounding volumes that enclose the primitives and form a hierarchy. Mixed forms of these techniques are also popular. There are special acceleration techniques for animations. The complexity of these techniques allows a ray tracer to grow quickly into a larger project.

No technology is generally optimal; efficiency is scene-dependent. Nevertheless, every acceleration procedure reduces the runtime enormously and makes ray tracing a practicable algorithm. For most non-animated scenes, K-D tree based subdivisions are the most efficient or near optimal techniques, as they can be optimized using heuristics cite9,10. It has been found that the asymptotic runtime of ray tracing is logarithmic depending on the number of primitives.

It has been shown that on modern computers the speed of ray tracing is limited not by processor performance but by memory access. Through careful use of caching by the algorithm it is possible to reduce the runtime considerably. It is also possible to use the single instruction multiple data (SIMD) capability of modern processors, which enables parallel computations, as well as specially optimized subdivision schemes [13, 9] This allows simultaneous tracking of multiple rays combined in "packets". The reason for this is that the rays emitted from the eye point are usually very similar, i.e. mostly cut the same objects. With the SSE instruction set, for example, four rays can be tested simultaneously for an intersection point with a primitive, which speeds up this calculation many times. On corresponding hardware implementations - for example on GPUs - larger packages with more than 10000 rays can also be tracked.

It is also possible to parallelize the entire ray tracing process. This can be achieved trivially by rendering different parts of the image or individual pixels with different processors or machines. Only certain acceleration techniques or extensions need to be adapted in order to be suitable for parallel processing.

Memory consumption

The basic ray tracing method requires very little memory. However, the scene itself, which nowadays often consists of several million primitives, occupies a lot of memory and can comprise several gigabytes. In addition, there is the more or less high additional memory requirement of acceleration techniques. Since such large scenes do not fit completely into the memory of the computer, swapping is often necessary.

In the case of larger objects that are present several times in the scene and differ only in their position and size (for example, in a forest full of trees), the entire geometry does not have to be saved. Instancing or procedural on-the-fly geometry generation saves considerable space in certain scenes.

Extensions

One of the reasons for the success of the ray tracing algorithm is its natural extensibility. The naïve method described above is insufficient for today's requirements of image synthesis. With increasing computing power and increasing inspiration from physics - above all optics and radiometry - several extensions and variants emerged, some of which will be briefly discussed here.

Basically, the attainable quality of the rendered images as well as the relative time requirement increased considerably with each extension and reached the maximum with *path tracing*. Only subsequent developments aimed to reduce the time needed for path tracing without sacrificing quality.

Shadows

Due to the flexibility of the ray tracing algorithm, it is possible to transmit light rays not only from the eye point, but also from any other point in the room. As Arthur Appel demonstrated in 1968, this can be used to simulate shadows.

An arbitrary point of a surface is located in the shadow exactly when there is an object between it and the light source. By transmitting a shadow ray from the intersection point at the surface towards the light source, it is possible to determine whether an object crosses its path. If this is the case, the intersection point is located in the shadow and the brightness of the ray is returned as 0. Otherwise, normal shading is used.

Recursive ray tracing

Recursive ray tracing was developed around 1980 by Kay and Whitted [6, 15].

Ray tracing can be applied not only to simple opaque objects, but also to transparent and reflective objects. Further light rays are emitted from the intersection points. In the case of reflective surfaces, for example, only the direction of the ray emanating from the surface must be taken into account in accordance with the law of reflection (incidence angle is equal to the angle of reflection) and a corresponding reflection ray must be calculated.

In the case of translucent objects, a ray is emitted in accordance with the Snellius' refraction law, this time into the interior of the object in question. In general, transparent objects also reflect part of the light. The relative colour proportions of the reflected and refracted ray can be calculated using Fresnel's formulas. These rays are also called secondary rays.

Since the secondary rays can fall on other objects, the algorithm is recursively invoked to allow multiple reflections and refractions. The hierarchical totality of the calls is also called the render tree.

Path Tracing

Although diffuse ray tracing allows numerous effects, it is still unable to simulate global illumination with effects such as diffuse inter-reflection and caustics (bright spots of light produced by bundling light). This is due to the fact that secondary rays are emitted from reflections, but not from diffuse surfaces.

In his 1986 publication James Kajiya described the render equation, which forms the mathematical basis for all methods of global illumination [5]. The “brightness” contributed by a ray is interpreted radiometrically correctly as radiometric density. Kajiya showed that secondary rays have to be emitted from all surfaces for global illumination. In addition, he also pointed out that a render tree has the disadvantage that too much work is wasted on the calculations in a large hierarchy and it is better to send out a single ray at a time. This method is nowadays known as path tracing, because a ray is searching its way through the scene from the eye point. Path tracing has a rigorous mathematical and physical basis.

In path tracing, if the secondary ray emitted by a diffuse surface directly hits a light source, this brightness component is usually ignored. Instead, the proportion of direct lighting continues to be calculated using a shadow ray. Alternatively, direct lighting can be calculated by transmitting only one secondary ray according to the local lighting model and, if it directly hits a light source, returning its radiance. Which of these two methods is more efficient depends on the local illumination model of the surface and the spatial angle of the light source viewed from the surface [12]. The conceptually simpler variant of path tracing, in which no shadow rays are emitted, is known as adjoint photon tracing. [8].

Although path tracing can simulate global illumination, the efficiency of the process decreases for small light sources. Especially caustics and their reflections are very noisy with path tracing, unless a lot of rays are emitted. Therefore, other path tracing-based procedures or extensions are usually used.

Light Ray Tracing is a rare variant in which the light rays are not emitted from the eye point but from the light sources. The pixels that are hit by the ray on the image plane are coloured. This makes it possible to simulate certain effects such as caustics well, but other effects are very inefficiently simulated because many rays miss the image plane.

Diffuse ray tracing

In addition to light refraction and reflection, recursive ray tracing enables the simulation of hard shadows. In reality, however, light sources have a certain size, which makes shadows appear soft and blurry.

This effect, as well as anti aliasing, shiny reflection and more, can be simulated with diffuse ray tracing (also known as stochastic ray tracing or distributed ray tracing), which Cook published in 1984 [3]. The idea is to send out several rays in different situations instead of one ray and to form the mean value from the calculated colours. For example, soft shadows can be created with core shadows and partial shadows by randomly perturbation the shadows of the light source’s surface [11]. The disadvantage is that image noise is generated when too few rays are used. However, there are possibilities such as importance sampling that reduce noise.

Ray tracing applications

Ray tracing calculations are considered to be very time-consuming. Ray tracing is therefore used primarily in the creation of representations where quality is more important than computing time. Ray tracing can take theoretically infinitely long, depending on the technology used, the scene complexity, the hardware used and the desired quality - in practice often seconds to several hours, in some cases even several days. In areas such as computer games and virtual reality, where spatial representations have to be computed in real time, ray tracing has not been able to established so far. Computer animation films are mainly produced with the REYES system, which avoids ray tracing calculations as much as possible. Occasionally, ray tracing was used by the demo scene.

However, ray tracing has several advantages over conventional real-time renderers based on Z-buffering: a simple implementation with manageable complexity, high flexibility in contrast to the graphics pipeline as well as the easier interchangeability of the shaders and thus an easier implementation of new shaders. The speed of ray tracing must therefore be set in relation to the achieved image quality. For the demanding quality requirements of realistic image synthesis, there is no alternative to ray tracing, especially for complex scenes with any material.

Efforts are being made to implement real-time capable ray tracers for complex scenes, which has already been achieved under certain conditions with processor and memory optimized software solutions (e.g. Nvidia's OptiX). Hardware-optimized implementations of ray tracing show that the future widespread use of ray tracing in real-time is conceivable. These applications are used in projects such as the OpenRT programming interface and various implementations for programmable GPUs (GPGPU). In addition, special architectures have been developed for hardware-accelerated ray tracing [10].

The raytracing principle can be extended to any application where the propagation of waves in a scene is to be simulated. Rays always represent the normal vectors to a wavefront. In auralization and high-frequency technology, attempts are made to simulate the effects of a scene on acoustics or an electromagnetic field. The goal is to calculate the energy share for certain frequencies, which is transmitted from a transmitter to a receiver via the different possible paths through the scene.

One way of finding the transmission paths is to send radiation from one source isotropically (in all directions), possibly reflecting it to the objects with a loss of energy, and to determine the total energy of the radiation impinging on the receiver. This method is called Ray launching. Rays can also have a certain "shape" - for example, that of a tube - in order to simulate point-shaped receivers. The disadvantage of this method is its sluggishness, as many rays never reach the receiver and a high number is required for precise statistics. Another problem arises from the fact that the wavelength is often not negligible compared to the dimensions of the bodies within a scene. If diffraction of rays is not taken into account, noticeable errors may occur in the simulation.

The pretty ray tracing picture

Figure 3 is a screen shot from a student's implementation of the ray tracing task during the course work of this lecture.

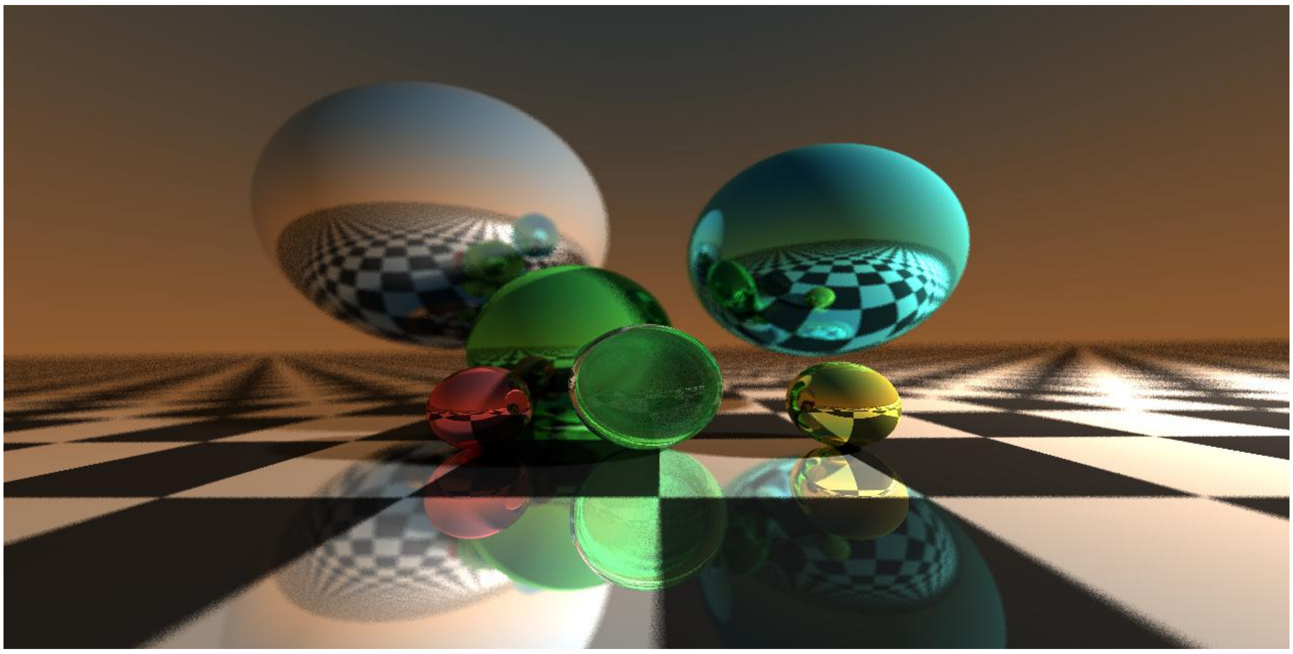


Figure 3: Example ray tracing from G. Jiang's course work solution. Can you identify all implemented visual effects?

References

[1] O. Abert, M. Geimer, and S. Muller. Direct and fast ray tracing of nurbs surfaces. In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 161–168, Sept 2006.

[2] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[3] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[4] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18(3):213–222, January 1984.

[5] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[6] Douglas Scott Kay and Donald Greenberg. Transparency for computer synthesized images. *SIGGRAPH Comput. Graph.*, 13(2):158–164, August 1979.

[7] Terrence Masson. *CG 101: A Computer Graphics Industry Reference*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.

[8] R. Keith Morley, Solomon Boulos, Jared M. Johnson, David Edwards, Peter Shirley, Michael Ashikhmin, and Simon Premoze. Image synthesis using adjoint photons. In *Proceedings of the Graphics Interface 2006 Conference, June 7-9, 2006, Quebec, Canada*, pages 179–186, 2006.

[9] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, July 2005.

[10] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. Saarcor: A hardware architecture for ray tracing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02, pages 27–36, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[11] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Trans. Graph.*, 15(1):1–36, January 1996.

[12] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.

[13] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 2001.

[14] Alan Watt. *3D Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1993.

[15] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.