ITN
Linköping University February 2, 2020

TNCG15 - Advanced Global Illumination and Rendering
A Monte Carlo ray tracer

Petra Gunnarsdotter (petgu692@student.liu.se)
Leo Nordling (leono184@student.liu.se)

TNCG15 - Advanced Global Illumination and Rendering
Course examiner: Mark Eric Dieckmann

Abstract

This report presents an implementation of a ray tracer with the Monte Carlo techniques. The project is part of the course TNCG15 - *Advanced Global Illumination and Rendering* at Linköping University and develop in the language, C++.

In Global illumination, the goal is to imitate the real light as good as possible. This is one of the biggest challenges in the subject because of the limitations in terms of time, memory and capacity. The calculation is also a complex task since reflections, refractions and shadows can appears in many ways that makes the calculations complex and interdependent.

In this project a scene containing a room with a tetrahedron and a sphere is implemented. The tetrahedron and the sphere has two different material to show how the light behaves on different materials. The tetrahedron has a Lambertian surface that absorb and reflect the light as a diffuse reflector and the sphere is a perfect reflector that reflects all rays. The project does not contain any caustics and is simplified to the previous named materials.

Contents

1 Introduction

In this paper a ray tracer has been created using the Monte Carlo technique. A scene has been implemented in order to test the ray tracer. The scene consists of a hexagon shaped room that contains two objects, one sphere and one tetrahedron. The objects have been implemented with different materials, the sphere has a perfect reflecting material and the tetrahedron has a material of a Lambertian reflector.

1.1 Global illumination

Global illumination can be calculated by the rendering equation which describes the light that is emitted from a point x along a specific viewing direction. The total received radiance is the sum of the emitted light and the reflected light. The reflected light is the sum from all the incoming light over the hemisphere on x. The reflected light is however dependent on other reflected light and this goes on and on making the rendering equation unsolvable. Light therefor has to be approximated.

One way of simulating light is by using ray tracing. This is done by shooting rays from a light source and letting them bounce in the scene until they hit the camera. However the chances of a ray hitting a camera is very small which leads to a large amount of rays has to be used. This makes the method very computationally expensive.

In 1979 , J.Turner Whitted published a paper that presented a ray tracing method that used backwards tracing [7]. The ray tracing described in the paper is instead done by having the rays originate from the camera. Each ray is set to bounce a predetermined amount of times or until they hit a diffuse surface. At each intersection point, the light contribution is calculated using a local lighting model. When a diffuse surface is hit, the algorithm sends out an amount of shadow rays to check if there is any object between the light source and the hit point. The amount of shadow rays that reach the light source will determine how much directly illuminated the hit point will be.

The method works very well for specular and transparent reflections. However the method do not consider the indirect light contribution coming from other diffuse reflections. One method for calculating the diffuse reflections is called the radiosity method. Radiosity works by dividing the scene into smaller patches. For each pair of patches a form factor is calculated. The form factor determines how much light is transferred from one patch to the other and is based on their

sizes, orientation, visibility and the distance between them. Another method for calculation diffuse reflections is called Monte Carlo integration which is also the method used in this project.

1.2 Monte Carlo ray tracing

Ideal diffuse surfaces reflects light uniformly in all directions. This mean that the surface is perceived the same regardless of the viewing direction. By using Monte Carlo integration the indirect light at each diffuse intersection point can be approximated by random rays that are sampled over a hemisphere around the intersection point. Since the samples are random the result will be unbiased and its therefore important to use enough samples to prevent noisy results.

2 Background

The implementation of the project is done in C++ and it is divided into classes to describe different instances. The project also used the OpenGL Mathematics library for handling of vectors.

2.1 Overview of the Method

The ray tracer that is implemented in this project uses a backwards tracing technique with Monte Carlo integration on diffuse surfaces. Initially a camera and a scene is created. From the camera a number of rays are shoot through each pixel in an image plane. Each ray will travel through the room and bounce on objects a predetermined amount of times. If the intersecting object has a perfect reflecting surface a new reflecting ray will be calculated. If the surface is diffuse, a number of shadow rays will be sent out and used to calculate the direct light contribution. Using more shadow rays will increase the smoothness of the shadows, especially if the image is rendered with a low amount samples per pixel. The indirect light will be calculated by shooting new rays into the scene. The new samples will be shot randomly over a hemisphere around the normal of the surface of the intersection point. This will happen recursively a predetermined amount of times. The indirect light is then the average of light contribution returned by the sampled rays.[1][3] [6]

2.2 Vertex, Triangles & Spheres

A vertex represent a point in a 3-dimensional space. The scene consists of two types of objects, triangles and spheres. The triangles consisted of vertices that creates the structure of the objects and the sphere is calculated mathematically.

Triangles are described by three vertex points and together they create a flat surface between them. The surface has a normal that describes which side of the triangle that is the front side. Together the triangles create the scene objects like walls, roof, etcetera. When a ray is calculated an intersection point between the triangle and the ray are calculated. The intersection point of a triangle is calculated by Möller-Trumbore algorithm.[2] Möller-Trumbore algorithm is calculated by describing the point of the barycentric coordinates u and v.

$$T(u, v) = (1 - u - v)v_0 + uv_1 + vv_2. \tag{1}$$

The equation 1 shows that the intersection point can be described by u and v with the vectors between the vertex points of the triangle, v_0, v_1, v_2. By help of

Cramer's rule the equation 2- 7 can be calculated.

$$T = p_s - v_0 \tag{2}$$

$$E_1 = v_1 - v_0 \tag{3}$$

$$E_2 = v_2 - v_0 \tag{4}$$

$$D = p_e - p_s \tag{5}$$

$$P = D \times E_2 \tag{6}$$

$$Q = T \times E_1 \tag{7}$$

It then leads to equation 8

$$T = uE_1 + vE_2 - tD \tag{8}$$

By these equations we can calculate the intersection point where u and v describe the point at the surface hits and t is then the length between the rays start and intersection point. The equations 2- 7 creates the matrices 9 that we then can calculate the intersection point in u, v and t. The value t is then used to choose which object that is closest by the camera.

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P \cdot E_1} \begin{pmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{pmatrix} \tag{9}$$

Spheres are represented by a vertex point as its centre and a radius. Spheres that are intersected by a ray has two intersection points. One point for entering the sphere and one point for when the ray exits. The intersection points are calculated geometrically with the help of trigonometry and Pythagorean theorem.[4]

$$P = O + tD \tag{10}$$

A ray is given by formula 10 where O is the ray origin and D is the ray direction. Changing the variable t will give any point along the ray. The ray enters the sphere at P when $t = t_0$ and exits at point P' when $t = t_1$ as shown in figure 1.

The length t_{ca} is given by the dot product of the vectors L and D, where L is the vector from the sphere centre to the ray origin. The length of d is then derived by the Pythagorean theorem shown in formula 11. By using formula 11 and formula 10 it is then possible to calculate the points P and P'.

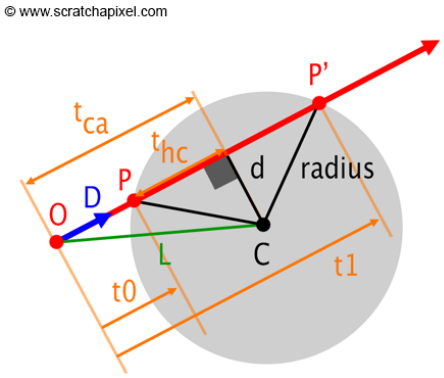

Figure 1

$$d = \sqrt{L^2 - t_{ca}^2} = \sqrt{L \cdot L - t_{ca} \cdot t_{ca}} \tag{11}$$

The sphere has a high glossy material that use a specular-reflection model. The model cast a new ray by the BRDF-model[5] and get the colour where the ray intersect when it has been reflected.

2.2.1 Scene

The scene contains a hexagon shaped room that with a tetrahedron and a sphere. The room is described with 14 vertex points. Of these points twenty-four triangles are formed, twelve of which are the walls of the room and six are the ceiling and six are the floor. In figure 2 a layout of the room with coordinates of the vertex points are shown. The floor is in the z-plan, where z = -5 and the ceiling where z = 5.

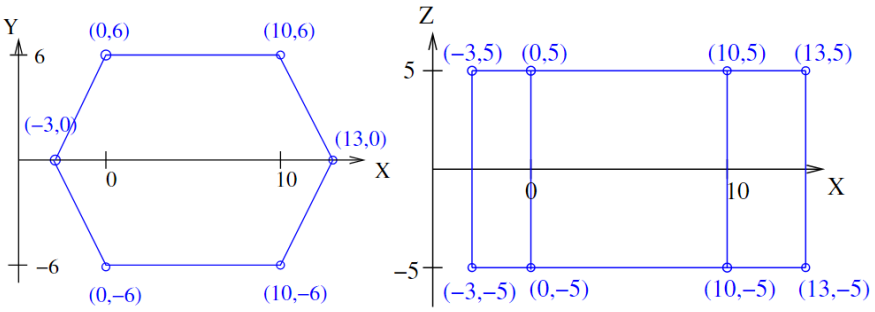

Figure 2: *The hexagon room in the scen with the coordinates.*

2.2.2 Tetrahedron

A tetrahedron is an object described with four vertexes that together form four triangles. The shape of a tetrahedron is shown in figure 3. The tetrahedron is a Lambertian reflector and behaves as a diffuse reflector.

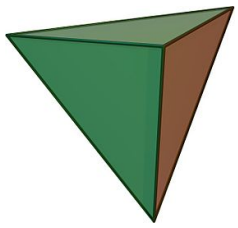

Figure 3: *A tetrahedron*

2.3 Camera

To view an image, a camera is created with pixels. In this project 800x800 pixels are used to create the camera-plane. The camera plane has a width and length of 2 units and it is centre in (0,0,0).

2.4 Pixel

The pixels of the camera are created to cast rays through it. Each pixel contains a colour that it gets when the image is rendered by the rays bouncing in the room.

2.5 Light and shadow-rays

Two different light sources are implemented in the project to compare the different behaviour, one area light and one point light. The area light is created by two triangle with the vertices, (6.0, 1.0, 4.9), (8.0, 1.0, 4.9), (6.0, -1.0, 4.9) and (8.0, -1.0, 4.9). The point light source is placed in the coordinates (4.9, 0, 4.9). The point light source is placed near the roof. When calculating the ray, a shadow-ray is sent out from the intersection point to the light point to check if the intersection point is in the direct light or in the shadow. If the intersection point is in the shadow the light contribution is zero and the point is only affected by indirect light also known as diffuse reflections.

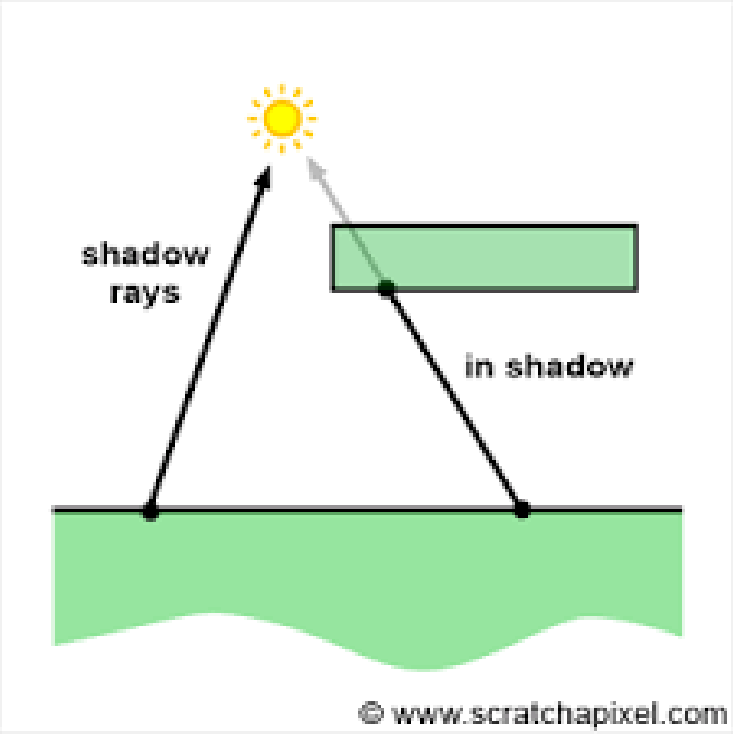

Figure 4: *A visualization of how the shadow-rays works.*

2.6 Ray

The ray is created to simulate how a light-ray moves in a room. The ray is represent by a direction as a vector and a start point. The ray contains a colour depending on what it hits.

When the ray is cast, it emits from the eye through each of the pixels into the room. The ray then bounces through the room and calculates the colour value of the pixel that is then displayed. Depending on whether it intersect with a sphere or a triangle, it is calculated differently as described previously.

3 Results

The result of the his project is a path tracer that uses Monte Carlo integration on diffuse surfaces. The ray tracer has been tested on a scene containing two objects with different materials. One tetrahedron with Lambertian surface and one sphere with perfect reflecting surface. The scene has been rendered using varying amount of samples per pixel and shadow rays per pixel. The ray tracer has also been tested using a point light and a square light. The stopping condition for the Monte Carlo integration has been set to 3 bounces. The different renderings can be shown in the figures 5 to 13. The images have been rendered with the same amount of shadow rays as samples unless stated otherwise. The rendering time from the images below ranged from a minute for figure 9 to 7 hours for figure 13.

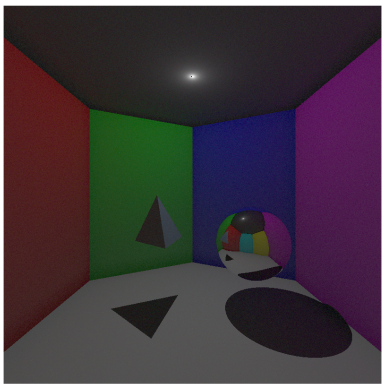

Figure 5: *Using 5 samples per pixel (point light).*

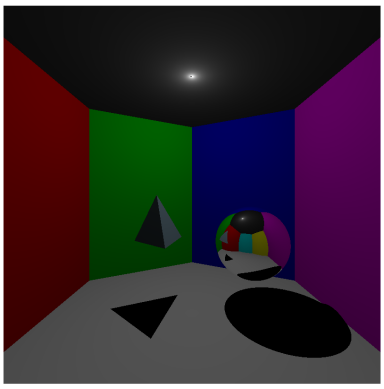

Figure 6: *Direct illumination from figure 5*

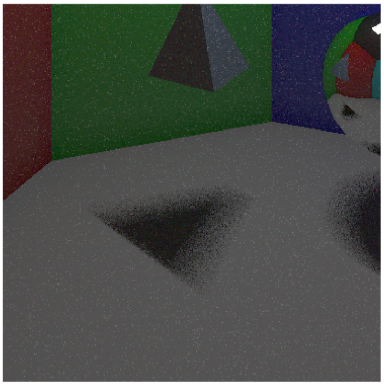

Figure 7: *5 samples per pixel and 5 shadow rays per pixel.*

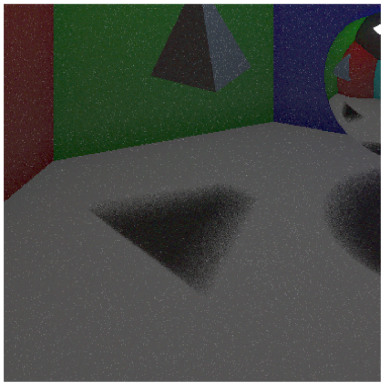

Figure 8: *5 samples per pixel and 20 shadow rays per pixel.*

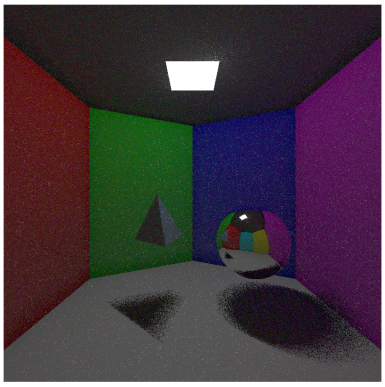

Figure 9: *Image rendered using 1 sample per pixel.*

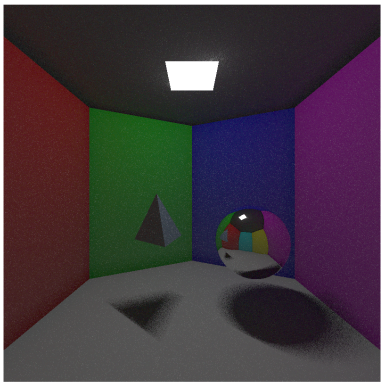

Figure 10: *Image rendered using 5 samples per pixel.*

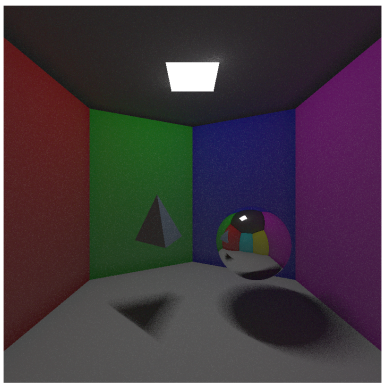

Figure 11: *Image rendered using 10 samples per pixel.*

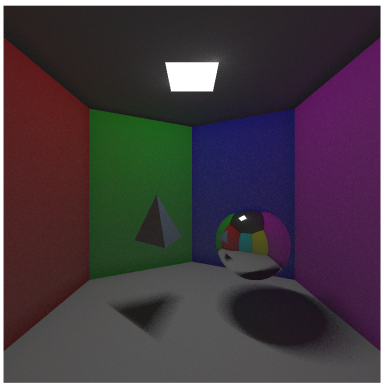

Figure 12: *Image rendered using 20 samples per pixel.*

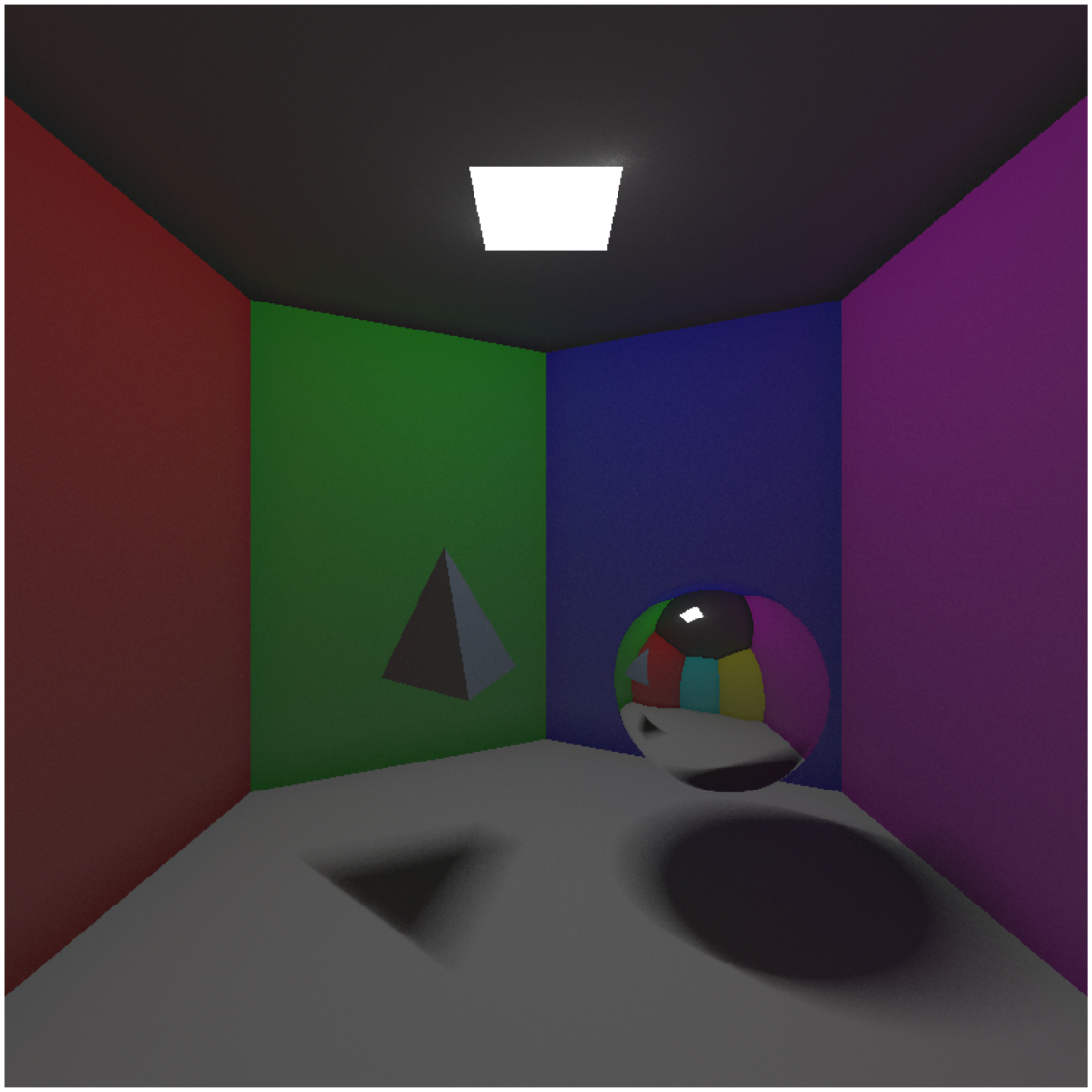

Figure 13: *Image rendered using 150 samples per pixel.*

4 Discussion

With the methods described in section 2, it was possible to implement a basic ray tracer. The ray tracer managed to render photo realistic light on the diffuse and specular surfaces in the scene. The scene has in figure 5 been rendered using 5 samples per pixel. The light from in the scene comes from a point light source. This can be seen by the hard shadows cast by the objects. Figure 6 shows the direct illumination of figure 6. Figure 7 and 8 are rendered with same amount of samples. However figure 8 used more shadow rays which resulted in smoother shadows. Figure 9 to 13 shows how the amount of samples used affect the amount of the noise in the image. Using 150 samples resulted in by far the least noisy result but also took a very long time rendering. Even though the implementation that was done in the project was not very well optimized, simulating photo realistic light is still generally a very expensive process. Optimizations could have been done both to the structure of the code as well as implementing other or additional methods. One of such methods is the Russian roulette for early termination during Monte Carlo integration. By giving the rays a chance to terminate early would enable the ray tracer to use more bounces without much increase of performance. Another way of speeding up the rendering would be to implement support for multi-core rendering which would have improved the rendering time significantly. There are also several other methods and lighting models that could've been used to improve the ray tracer and make the end result more photo realistic. However for a simple scene such as the one used in the project it was sufficient to use a simple ray tracer to get a photo realistic image.

References

[1] Dutré P, Bekaert P, Bala K. Advanced Global Illumination. 1 st. Natick,Massachusetts: A K Peters, Ltd; 2003.

[2] Scratchapixel An Overview of the Ray-Tracing Rendering Technique [Internet]. Scratchapixel 2.0; 2014 [2020-1-27]. Available from:https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection

[3] Scratchapixel An Overview of the Ray-Tracing Rendering Technique [Internet]. Scratchapixel 2.0; 2014 [2020-1-20]. Available from: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/light-transport-ray-tracing-whitted

[4] Scratchapixel A Minimal Ray-Tracer: Rendering Simple Shapes (Sphere, Cube, Disk, Plane, etc.) [Internet]. Scratchapixel 2.0; 2014 [2020-01-20]. Available from: https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection?fbclid=IwAR3JihjcSt0XxcU33yBIQWbRU4Th3RHQnQYdS-0LKE6gfuCLeuNwoNk2Qow

[5] Bidirectional reflectance distribution function [wiki on the Internet]. Wikipedia, the free encyclopedia; 2019 [2020-01-17]. Available from: https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function

[6] Wann Jensen, Henrik. Global Illumination using Photon Maps [Internet]. Utgivningsort: Department of Graphical Communication, The Technical University of Denmark; pages 21-30, 1996. [2020-01-14]. Available from: http://www.gk.dtu.dk/ hwj

[7] Turner Whitted, An Improved Illumination Model for Shaded Display, 1979