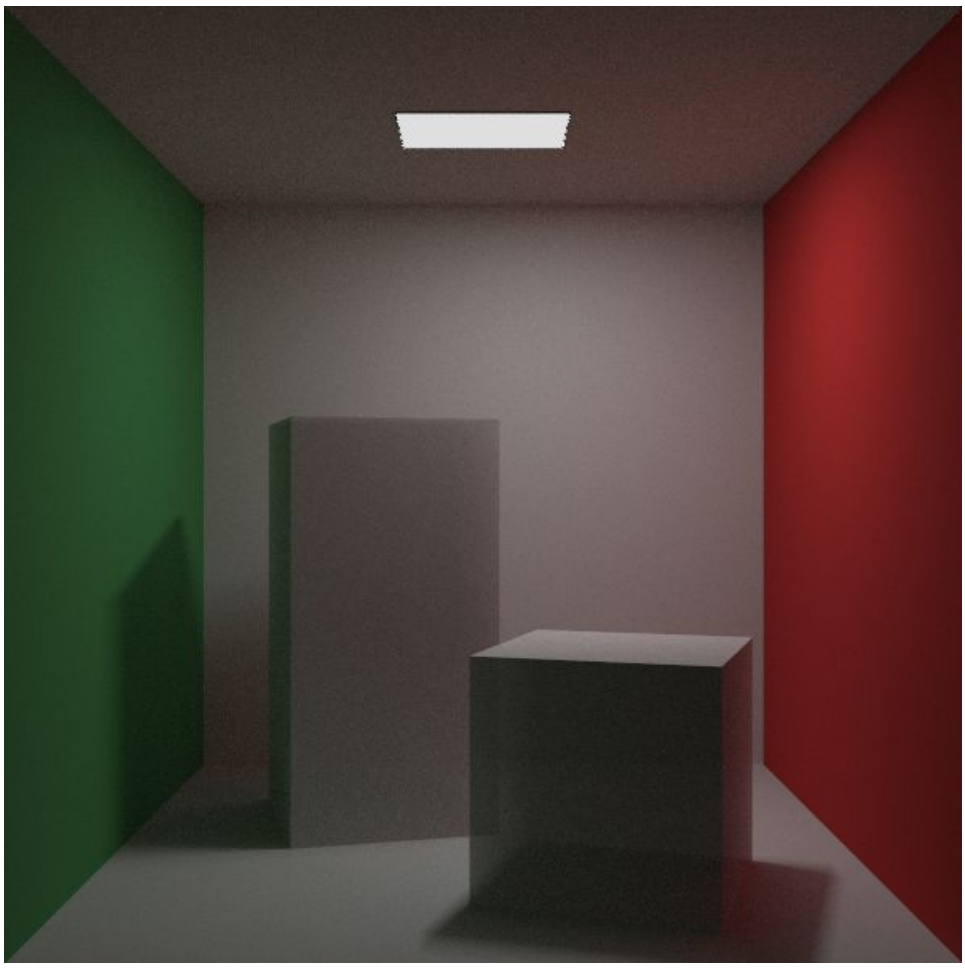
A Simple Approach To Translucent Materials

Kristoffer Lundgren

2020-03-27



Abstract

This paper introduces a simple way of implementing Subsurface scattering using already established ways of simulating lambertian materials and volume scattering. The purpose of the method is to simulate opaque materials as fast as possible while still producing credible results.

1. Introduction

Modeling the transport of light using ray tracing is a fundamental part of creating lifelike lighting generated by a computer. Ray tracing is based on the principle of rays being cast by either light sources(Forwards Ray Tracing) or from the camera(Backwards Ray Tracing). The problem with forward ray tracing is that the number of rays required to render a scene is so large that the performance is almost unacceptable. The advantages is that it is more accurate to how light behaves in real life which in turn will create better results with the disadvantage of long runtimes. Forward ray tracing is the most commonly used method when rendering scenes using ray tracing, and the method used in the implementation of opaque materials in this paper. This implementation aims to provide a simple way to simulate opaque materials with better performance than methods presented in other papers.

2. Background

2.1 Monte-Carlo Ray Tracing

This paper implements a monte carlo ray tracer which is the basis from which different types of materials and surfaces can be added. The ray tracer works by shooting rays from a camera into an imaginary plane populated by pixels. A predetermined number of samples are taken in random locations in each pixel, the color values from all the samples are then averaged and the pixel is colored with that value. This process is then repeated for each pixel on the screen and the result is written to a text file which can then be opened as a PPM image.

2.2 Importance Sampling

When a ray is fired from the camera it bounces according to what materials it impacts which means the ray will bounce until it reaches the maximum number of bounces. In order to reduce the number of bounces and improve performance the rays have a probability to be directed at a light source at every bounce. This has the effect of creating harder shadows and decreasing the number of bounces required to generate a scene. It also eliminates the need to implement shadow rays.

2.3 BRDF

Bidirectional reflectance distribution function is a function which defines how the light rays bounce on an opaque surface. The function uses four variables: incoming direction, outgoing direction, both of which contains two variables each results in four total variables. The method in this paper is a simplification of full BRDF. It uses an orthonormal coordinate system and a probability density function(PDF) to distribute the rays evenly across a hemisphere.

2.4 Lambertian Materials

A lambertian material or a matte material is a material with no specular reflections. The lambertial materials in this paper uses the previously mentioned PDF and a generated

orthonormal coordinate system to reflect the rays in a evenly distributed way. The only modification from a true lambertian material is direct light sampling to decrease the number of bounces and improve performance.

2.5 Volume Scattering

The method of simulating subsurface light transport in the paper "A Practical Model for Subsurface Light Transport" Uses two parts for the simulation. The single scattering term and the diffusion approximation. This method produces extremely lifelike results but the downside is the difficulty of implementing the diffusion term. This paper uses only a single scattering term to simulate the subsurface scattering. When a ray hits the material it has a probability of penetrating the material. And when the ray enters the material the probability of scattering depends on the density of the material and the distance the ray travels inside the medium.

3. Theory

3.1 Ray Tracer

Since the ray tracer is such an important part of any light simulation this paper will summarize how the ray tracer this paper uses works.
A ray is fired through a plane which is divided into an arbitrary amount of pixels which is decided before runtime. The number of rays per pixel is also decided before runtime.

$$(\sum_{0}^{p} pixelColor) \div nrPixels \quad (1)$$

All the colors from all the rays in one pixel are then averaged to one single color, described by equation (1).

When a ray hits an object a plane to ray intersection is run to check for collisions. In order to handle when objects are moved the inverse of the move vector on the object is applied to the ray to transform them into the same space. Rotations of objects are a little more complicated. For rotations arounds the y axis the following applies.

$$newX = cos(\theta) * x - sin(\theta) * y \quad (2)$$
$$newY = sin(\theta) * x + cos(\theta) * y \quad (3)$$

These new coordinates can then be used to calculate a new bounding box. To check for a hit on a rotated object, the rays direction and origin needs to be modified by taking the rotation into consideration.

$$originX = cos(\theta) * rayOriginX - sin(\theta) rayOriginZ \quad (4)$$
$$originZ = sin(\theta) * rayOriginX + cos(\theta) rayOriginZ \quad (5)$$

$$dirX = cos(\theta) * rayDirX - sin(\theta) * rayOriginZ \quad (6)$$
$$dirZ = sin(\theta) * rayDirX + cos(\theta) * rayOriginZ \quad (7)$$

The ray is now in the same coordinate space as the rotated object and a normal ray plane intersection can be run to check for collisions. Equations 4, 5, 6 and 7, are only valid for rotations around the Y axis but can easily be modified to work for other rotations.

3.2 Volume scattering

There are different approaches to volume scattering depending on how complex the simulation is aimed to be. This paper uses single scattering which means that the ray will bounce inside the material, stop or pass straight through.

$$hitDist = -(1/density) * log(rand) \quad (8)$$

If the value of Equation 8 is calculated to be less than the total distance that the ray will travel through the box the ray will scatter else the ray will continue through the medium. The probability that the ray will scatter inside the medium is dependent on the density of the material and random number generated between zero and one.

3.3 Implementation

The proposed method uses properties of standard lambertian materials and the previously mentioned implementation of volume scattering to simulate translucent materials.

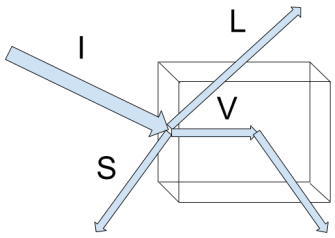


Figure 1: The different behaviours of a ray when it hits a box

The incoming ray I hits the surface of the box and 80% of the time it will scatter on the surface of the box. The rays that scatter on the surface have a probability of either being directed straight towards the light according to L or being scattered according to a PDF which is shown by S. Rays that enter the box as shown by V behave as described in section *3.2.* The ratios that decide where a ray will go next can be adjusted and they impact the look of the translucent material and the lighting of the scene.

4. Result

Figure 2: Image was generated in 50 seconds on a quad-core cpu running at 4.2GHz. Resolution is 300x300 and 300 samples per pixel.

Figure 2 is an example of low density materials simulated by the program. Even at relatively low resolutions and samples per pixels the results look relatively good.

FIgure 3: Image was generated in 50 seconds on a quad-core cpu running at 4.2GHz. Resolution is 300x300 and 300 samples per pixel

Figure 3 is the same resolution and the same number of samples per pixel as Figure 2 but the density of the boxes is higher, making the

subsurface scattering limited to the edges of the boxes.
This is what would be used to simulate materials like milk and marble.

5. Discussion and Future Improvements

The method described in this paper is a simplification of already proposed methods and aims to produce similar results with less computing. The performance numbers provided are not the best case since they are from the program run in WSL(Windows Subsystem for Linux) which negatively impacts performance. Ray tracing is an operation which heavily benefits from parralization which is why they should be run on the GPU. This is evident when looking at Nvidia's efforts to bring real time ray tracing to games. By running ray tracing with compute shaders performance can see staggering improvements. The only problem is resource sharing when working with so many independent threads. This paper uses multithreading to improve performance which helps but to bring this anywhere close to real-time a GPU compatible program is needed. The only upside to doing something like this on the cpu is simplicity, working with something that runs on the CPU is easier then something running on the GPU.

A great next step for this solution would be to adapt it for GPU acceleration. Another problem is the somewhat inconsistent "dead" pixels that appear on the renderings. As of now a specific reason has not been found but it could be a result of the multithreaded operation or an integer overflow. The most time consuming part of working with ray tracing is the process of tracing down bugs, since at first look there doesn't seem to be anything wrong. Miscalculations can still produce images that look right but the issue may not be apparent until much later.

6. References

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, Pat Hanrahan. 2001. *A Practical Model for Subsurface Light Transport*

Per H. Christensen, Brent Burley. 2015. *Approximate Reflectance Profiles for Efficient Subsurface Scattering*

Matt Pharr, Wenzel Jakob, Greg Humphreys. *Physically Based Rendering: From theory to implementation. ISBN: 978-0128006450*

Peter Shirley. 2020. *Ray Tracing in One Weekend* (*https://raytracing.github.io/books/RayTracingInOneWeekend.html*)

Peter Shirley. 2018. *Ray Tracing: The Next Week* (*https://www.realtimerendering.com/raytracing/Ray%20Tracing_%20The%20Next%20Week.pdf*)

Peter Shirley. 2018. *Ray Tracing: The Rest of Your Life* (*https://www.realtimerendering.com/raytracing/Ray%20Tracing_%20the%20Rest%20of%20Your%20Life.pdf*)