Monte Carlo Path Tracing with Photon Mapping

Avishek Biswas

CPSC 6050: Computer Graphics Project

Abstract

In this project, I develop a Monte Carlo Path Tracer in C++ to simulate light transport in Cornell Box Scenes. The program supports a variety of material types like diffuse, specular, glossy, perfect mirrors and refractive. I use Photon Mapping to render the caustics in the scene like mirrors and refractive objects. In the rendering phase, I have used the Next Event Estimation technique by computing both direct and indirect lighting at each ray-geometry hit-point. The indirect light bounces are computed using Cosine Importance Sampling to improve sample efficiency. Finally, as a denoising technique, a scene-aware filter is used. An adaptive sampling method is also adopted to render high discrepancy pixels with higher priority than low discrepancy ones. The implementation also uses parallel threading to speed up the rendering time.

1. Introduction

The motivation behind this project was to address the problem of Global Illumination by monte carlo sampling methods and explore the world of Physically Based Rendering. There has been a gradual but decisive progress in both the game industry and the movie industry towards Physical Based Rendering. This is partly due to the increased compute power and resources available to producers, but also due to the increased demands of real-world simulation in terms of light transport, models, animation, and rendering in general from the audience. Industry standard renderers like Arnold, RenderMan, MoonRay, Hyperion, have all shifted to a physical rendering system (some of them more recently than others). As such, PBR has been an active field of research among computer scientists.

2. The Rendering Equation

The rendering equation (Kajiya, 1986)[1] aims to represent the total irradiance at any given point in a scene. The irradiance is given by:

$$\mathbf{L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f(x, \omega_o, \omega_i)\, L_i\, (\omega_i.n)\, d\, \omega_i}$$

Where,

$L_o(x, \omega_o)$: Light exiting at point x towards direction ω_o

$L_e(x, \omega_o)$: Light emitted at point x towards direction ω_o

$\int_\Omega$: Integral over the unit hemisphere Ω surrounding the surface normal n

ω_i : Direction *towards* the incoming light from x

$f(x, \omega_o, \omega_i)$: Bidirectional Scattering Distribution Function (BSDF) at point x. This gives a ratio of the differentials between the outgoing light direction (ω_o) to the incident light direction (ω_i) at point x.

n: Surface normal at point x

$\omega_i.n$: The dot product of the incoming light direction and the normal. This provides a Lambertian effect of decaying the irradiance intensity as the angle of incidence moves away from the normal. This can also be written as $cos(\Theta_i)$ where Θ_i is the angle between the incoming light direction ω_i and the surface normal n.

The above equation tells us that the outgoing irradiance at each point in the scene can be calculated by taking an integral over all possible incident light rays in the

hemisphere of the intersecting object oriented towards the normal of the surface. The BSDF of the surface determines how much light will be reflected in the outgoing direction ω_o from the incoming light direction ω_i. This function determines the nature of the object, ie specular, diffuse, fresnel, etc.

The integral in the rendering equation does not have a closed form and formulates an equation over an infinite integral (there are infinite number of incoming light direction ω_i to compute). Solving the equation analytically is an impossible task even if we had the analytical form of the BSDF available. Instead, Kajiya suggested to use the Monte Carlo Sampling technique to solve the equation recursively. The process is discussed in the next section.

3. Monte Carlo Integration

Monte Carlo integration is a numerical method to solve integrals using statistical sampling. In Monte Carlo, we perform sampling over a random variable and observe the results of the simulation (a sequence of samples). Having collected the samples, we calculate the average of the samples with respect to the probability distribution of the samples to estimate the value of an integral in the specified domain.

Mathematically, if we want to evaluate the integral:

$I = \int_a^b f(x)\, dx$

We can calculate N independent samples $X_1, X_2 \dots X_N$ distributed by a probability density function $pdf(X_i)$, and then compute the estimate as follows:

$I_{estimate} = (1/N) * (b - a) * \Sigma_{i = 1:N} f(X_i) / pdf(X_i)$

In Monte Carlo Theory, it is said that at the limit of $N \rightarrow$ infinity, $I_{estimate} \rightarrow I$.

Choosing a $pdf(X_i) = 1$ denotes a strategy of choosing samples uniformly at random. Therefore, for a uniform sampling, the value of the estimator reduces to:

$I_{estimate} = (1/N) * (b - a) * \Sigma_{i = 1:N} f(X_i)$

One of the issues with the Monte Carlo Estimator (with arbitrary pdf) is that we often do not know what the probability density function of a random variable actually is. While a randomly uniform sampling strategy is guaranteed to converge at the limit; in practice it often requires a lot of samples. An optimization will be to use Importance Sampling instead. The idea is to draw samples from the functions where f(x) is going to be most "important", i.e. samples that have high probabilities to occur and/or have the maximum influence on the final evaluation.

If we were to choose a probability distribution g(x) whose shape is like the actual probability distribution pdf(x), we will be able to generate samples where the function f(x) is also important. We draw the samples $X_1, X_2, \dots X_n$ using a different (known) probability density function (say) g(x).

$I_{estimate} = (1/N) * (b - a) * \Sigma_{i = 1:N} [f(X_i) / g(X_i)]$

The closer the shape of $g(X_i)$ is to $pdf(X_i)$, the lesser number of samples we require for the Monte Carlo simulation to converge. It should be noted that a random variable can be sampled according to a given probability function g(x) by calculating the cumulative probability distribution function G(x) of g(x), and then simply sampling the Inverse CDF $G^{-1}(u)$ where u is a random float in the range [0, 1].

I will revisit Importance sampling later when I write about Cosine Importance Sampling.

4. Monte Carlo Path Tracing

In this section, the algorithm for a general Monte Carlo Path Tracer is discussed. Path tracing incrementally generates paths of scattering events starting at the camera and ending at light sources in the scene. A very naïve version of the Monte Carlo Path Tracer is presented next.

def path_trace(Ray ω_i , int depth) :

- If depth == 0 return BLACK
- Find nearest intersection
- If intersected with light source, return intensity of the source
- Generate a random ω_i by sampling (ϕ, Θ) depending on the point of intersection (x) and normal at the surface N_x
- L_i := path_trace(ω_i , depth - 1)
- $L_{out, x} := L_{emitted} + (L_i * BRDF * \cos(\phi))$
- Return $L_{out,x}$

At each hit point, we generate a random incident light direction by sampling the azimuthal angle Θ that goes from 0 to 2π and the zenith angle ϕ that goes from 0 to $\pi/2$.

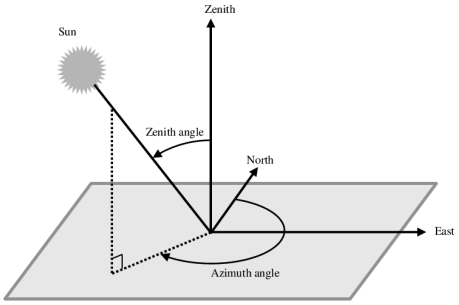

Figure 1: Azimuth and Zenith Angles

We send a ray from the hit-point towards this direction and compute the irradiance recursively. The obtained irradiance at each depth is aggregated to compute the illuminance. A ray can be terminated in three ways : a) if the maximum depth of recursion is reached, b) if the ray hits a light source, c) (not shown in the above algorithm) if the Russian roulette random variable determines the ray to be absorbed by the surface material. More about the Russian roulette technique is discussed later in 4.3.

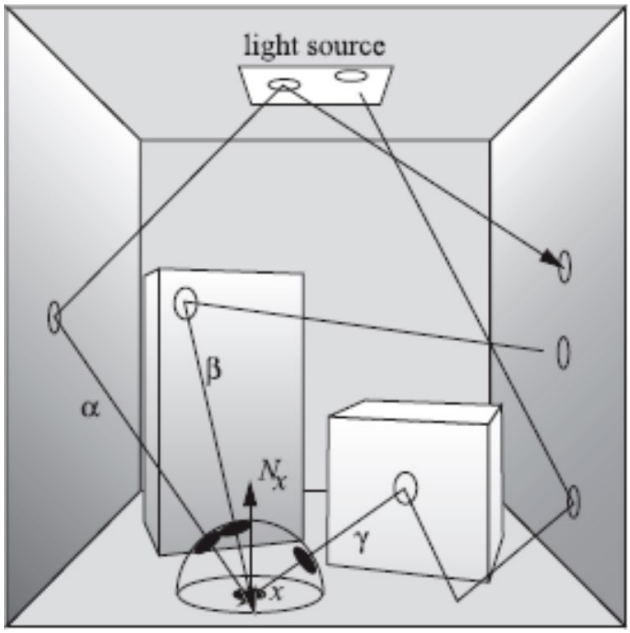

Figure 2: Tracing Paths in Path Tracing (Advanced Global Illumination, 2nd Edition, Kavita Bala)

4.1 Next Event Estimation

Next Event Estimation (NEE) is a technique commonly used in Monte Carlo Renderings to reduce the variance by directly sampling all the lights in the scene. In a sense, it is importance sampling the light sources in the scene, by maximizing the L_i term in the rendering equation. In my renderer, the direct illumination is calculated separately along with the indirect illumination at every depth-level of the path tracer and the net $Light_{out}$ is a direct sum of the two terms.

All light sources used in the renderer are area light sources. To calculate the direct lighting at any point in the scene, the surface of the area light is sampled stochastically (bi-linear interpolation of random variable for quadrilaterals and triangle, or point on unit sphere if object is a sphere), and it's visibility is computed just like in a regular Whitted Ray Tracer[9]. Since a new point is randomly sampled at every hit-point and at every iteration of the sampling process, integrating it into a path tracer is not complicated. At the same time, it produces significantly fast convergence during rendering.

A crucial precaution to take when employing Direct Illumination in a Path Tracer is to make sure that the light intensity is not accounted for twice as we are also calculating indirect illumination. In other words, we don't want the points of our world get illuminated twice by the same light source – once by direct lighting computation and again by indirect lighting computation. This has been handled by not returning any illuminance when an indirect ray intersects with a light source at any depth in the recursion. Thus, illuminance along the recursion tree is returned only if the camera ray hits the light source. The pseudocode provided at the end of this section illustrates this step.

4.2. Importance Sampling of BRDF

To sample successive rays after intersection, randomly generating next rays may not be a good idea. While random uniform sampling is a great choice for arbitrary BRDFs (a BRDF whose shape we know nothing about), it is generally better to do importance sampling when we have an idea about the shape of the BRDF. Importance sampling is sample efficient and tries to select such ω_i with high probability that will generate the maximum throughput and converge in lesser number of samples. For example, if we

are hitting a specular object, Importance Sampling would mean generating rays in the direction of the *specular lobe*. In general, to importance-sample any arbitrary BRDF, we need to draw samples of (ϕ, Θ) that intelligently favors the BRDF.

In this project, I used a Phong reflectance model given by

$$f(x, \omega_o, \omega_i) = f_{diffuse}(x, \omega_o, \omega_i) + f_{specular}(x, \omega_o, \omega_i)$$

$$= k_d * cos\ \phi * 1/\pi + k_s * n/2\pi * \cos^n a$$

k_d = diffuse coefficient,
k_s = Specular coefficient,
ϕ = Angle between incoming light and normal,
a = angle between reflective direction and outgoing ray direction,
n = shininess coefficient
The Phong BRDF needs to have $k_d + k_s <= 1$ in order to be energy conserving. [4][5]

It can be proven[5] that drawing samples using the below formulation to generate the samples will favour ray direction that maximizes the cos ϕ term in the diffuse model, thus generating rays that will have the maximum influence for the light irradiating from the given point:

$$\Theta = \cos^{-1}(\sqrt{u_1})$$

$$\Phi = 2\pi u_2$$

For specular materials, we want to generate paths towards the "specular lobe" with high probability. To sample (a, Φ), we use the below values:

$a = \cos^{-1}(\sqrt[n+1]{u_1})$ where n = shininess coefficient

$$\Phi = 2\pi u_2$$

4.3. Russian Roulette

The Russian Roulette[6] method is a standard method to determine how and when to terminate a ray. Let's suppose we have a material where the diffuse coefficient is K_d, the specular coefficient is K_s, and the absorption coefficient is K_a. According to Russian Roulette scheme, we can draw a random variable, say r, and if $r < K_a/(K_a + K_d + K_s)$, we will assume that the ray was "absorbed" by the surface and will not take further part in the light computations. Russian Roulette is a staple technique used in many path tracing algorithms as it prunes paths of the rays that produces low throughput. In other words, paths that will not contribute a great deal to the final scene have higher probability of getting terminated.

The Russian Roulette scheme can also be extended to determine the nature of light interaction with materials with dual properties, i.e. diffuse reflection, specular reflection. For example, if $r < (K_a + K_d)/(K_a + K_d + K_s)$, we can reflect diffusely, and else sample for specular reflections.

The path tracer generates new light paths much more efficiently. It starts at the camera and finds the first intersection, determines the next event by Russian roulette, samples the BRDF at the point to determine the next ray direction, and advances the recursion to the next level.

The below illustration summarizes the skeleton of the path tracing algorithm. For clarity and directness, a few details such as refraction, pdf scaling, is omitted.

def path_trace(Ray ω_i , int depth) :

- If depth == 0 return BLACK
- Roulette = rand()
- If Roulette < (K_absorption) / (K_diffuse + K_reflect + K_absorb) return BLACK (*//Ray absorbed and terminated)*
- Find nearest intersection
- If intersected with light source, return intensity of the source if (ray is Camera Ray) else return BLACK (*//No light returned for Indirect Illumination in Next Event Estimation model*)
- Compute direct illumination L_{direct} by directly sampling light sources
- Generate random variables [u_1, u_2]
- If Roulette < (K_absorption + K_ diffuse)/(K_ diffuse + K_reflect + K_absorb):
  - $\Phi = 2*\pi*u_2$, $\Theta = \arccos(\text{sqrtf}(u_1))$ *(//Diffuse)*
    Else
  - $\Phi = 2*\pi*u_2$, $\Theta = \arccos(\sqrt[n+1]{u_1})$ *(//Specular)*
- $\omega_i =$ (ϕ, Θ) (*// Generate a random light ray using Cosine Importance Sampling*)
- $L_{indirect}$:= path_trace(ω_i , depth - 1)
- $L_{out,\ x} := (L_{indirect} * \cos(\phi)) + L_{direct}$
- Return $L_{out,x}$

5. Photon Mapping

Standard path tracing algorithms, and even advanced path tracing algorithms (like Bidirectional Path Tracing) are not

efficient in rendering caustics. Caustics is referred to lighting contributions due to light sources towards specular, reflective, or refractive surfaces on other surfaces on the scene. Below is an example of caustic:

Figure 3: Caustics. Image source : Advanced Global Illumination Siggraph 2008 Wojceich Jarosz, Henrik Wann Jensen

Photon Mapping[7] is a global illumination algorithm that efficiently simulates various effects like diffuse inter-reflections, color bleeding, caustics, and even more advanced effects related to participating media and subsurface scattering. In this project, the Photon Mapping technique is exclusively used to render caustics. The algorithm is a two-pass technique. In the first pass, photons are emitted from the light source(s) towards all the caustic objects in the scene, like mirrors and refractive objects, and they are collected in a Photon Map data structure when they hit a diffuse surface. Nothing is rendered in the first pass. The second pass is the rendering phase when we implement the path tracer described in Section 4. During this phase, the photons are queried for each hit point to determine whether a caustic formed on the point or not.

A KD-Tree data structure is a spatial partitioning structure that returns a range search in O(k + log n) time, where, n is the number of photons stored, and k is the number of points found. As such, the algorithm is much faster with a smaller range query, and provides little overhead (log n) for areas of the scene that aren't connected to a caustic since k = 0.

5.1. First Pass – Caustics Map

All the caustic objects in the scene are first identified. Then photons are emitted from a light source towards each of these objects. The total number of photons are fixed for every light source, and the intensity of every photon is determined by dividing the light's intensity with the number of photons.

$$P_{photon} = P_{light} / n_{photons}$$

The photons are represented in a simple structure containing the below information:

- Photon's position
- Direction
- Power

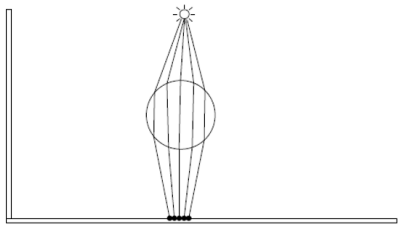

Figure 4: Building a Caustics Map. Image source : Advanced Global Illumination Siggraph 2008 Wojceich Jarosz, Henrik Wann Jensen

The number of photons sent out towards each caustic can also be determined by a Russian Roulette type scheme, where the probabilities are scaled by the surface area of each caustic, and the relative $K_{refract}$ and/or $K_{reflect}$. In my implementation, I have opted for a more naïve approach where I am sending an equal amount of photons for every caustic in my scene. This worked out well for the images I tried to render since the relative importance of all the objects in my scene were similar to each other. However, I think this is still an area with room for improvement.

The photons are made to interact with the objects it hit, and decides whether to reflect, refract, or absorb depending on the Russian Roulette technique mentioned in Section 4.2. At each interaction, the power of the photon is attenuated depending on the cosine of the angle of incidence, surface coefficients, and distance travelled. Once the photons hit a diffuse surface, the photon is inserted in the Photon Map.

Def trace_caustic (Ray ω_i , int depth, Color power):

- if depth == 0, return
- Compute nearest object intersection
- If object is reflective or refractive
  - Generate russian_roulette random variable r
  - If r → absorb: return
- If r → reflect: trace_caustic (reflected_ray, depth – 1, power * K_d * surface_color * cos Θ)
- If k → refract : trace_caustic (refracted_ray, depth – 1, power * K_d * surface_color * cos Θ)
- If object is diffuse
  - Insert photon into Photon Map Data Structure

5.2. Second Pass – Rendering

In the rendering phase, the traditional path tracing algorithm is followed. Only for the eye-ray however, it is checked if the intersection point is in the vicinity of a photon. This is done by a range-query on the photon map. The collection of photons returned by the query is then aggregated using a cone-filter to compute the total power contributing to that point in the scene.

6. Adaptive Super-sampling

The renderer for this project adopts a super-sampling approach on pixel like the strategy first introduced in the 1984 Siggraph paper on Distributed Ray Tracing by Cook[8] et. al. where they suggested to sample the pixel space with jittered sampling to avoid aliasing artefacts. Jittering on the pixel space refers to stochastically generating any point on the pixel and sending the eye ray through that. Jittered sampling is inexpensive in a Path Tracer as multiple samples are needed anyway per pixel. For every new sample per pixel, a random point is selected inside the pixel and a ray is sent through the same.

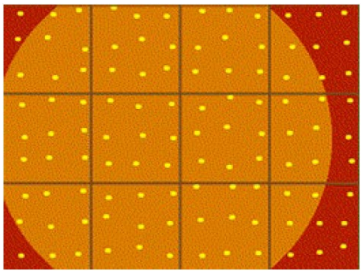

Figure 5: Stochastic or Jittered Supersampling (Image Credit: Dr. Victor Zordan Slides)

The renderer also uses an adaptive sampling technique meant to sample pixels that have high variance compared to its adjacent pixels. At every step before the recursive path tracing function is called, the discrepancy in the pixel is estimated by browsing through its adjacent pixels. In case the pixel is found to be different than a majority of it's neighbors, the path tracing function is called with the usual maximum depth; otherwise, the path tracing function is either called with a reduced depth limit, or not called at all. In order to reduce the bias of this technique, the adaptive pruning is done only after a minimum number of samples have been collected for all pixels (usually between 30 and 50 depending on the complexity of the scene).

7. Experiments and Code Structure

The entire program was written in C++ programming language without using any external library. An OpenGL window was used to run a progressive renderer on a separate thread to draw the color values produced by the path tracer into a pixel buffer array. Multiple threads were used on specific parts of the image to parallelize the program. Static Cornell box scenes were mainly rendered along with primitive objects like triangles, quadrilaterals, planes, and spheres. An edge-aware cone filter was employed to reduce noise in the image, but I found the adaptive sampling technique discussed in Section 5 produced better results (at the cost of more time) than the filtering technique. Filtering can be set to ON or OFF as a hyperparameter before running the program. In fact, many components of the program can be tuned as hyperparameters, for example, filter kernel width, filter intervals, maximum depth of recursive samples, switching on Next Event Estimation, number of photons to emit, Samples per Pixel(SPP) etc. Material presets were created using properties like diffuse albedo, K_specular, K_diffuse, K_refract, IOR, etc. An object-oriented approach was adopted for modelling the geometric objects like the Triangle, Sphere, Plane, and Quadrilaterals which derive from a virtual Object class. A KD-Tree data structure was written to be used for storing the Photon Map efficiently. The range query was modified from traditional implementation to also set the maximum number of photons to fetch. This allowed for faster rendering times at the cost of very little difference in the final image. Finally, an option for ONLINE and OFFLINE photon map rendering was added as a hyperparameter. In the ONLINE option, the photon map is calculated during the path-tracing step, and the illumination due to the photons are added to the direct illumination and indirect illumination. In practice this can take a lot of time, but produces more accurate results than the OFFLINE option. In there, the photon density across the scene is calculated in a separate pre-pass and stored in a separate buffer of the same resolution as the final image. During rendering, the value for the photon map is simply looked-up in the buffer and

superimposed on the final image, thus avoiding expensive range queries at every iteration.

8. Results

Following are some of the images rendered by the program. No kind of filtering was used for any of the images unless otherwise specified.

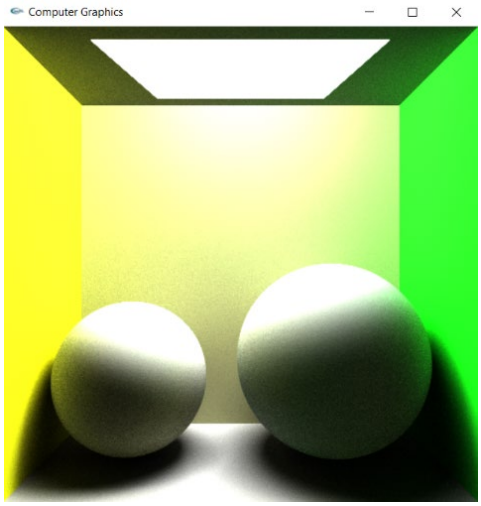

Figure 6: Diffuse Interreflections

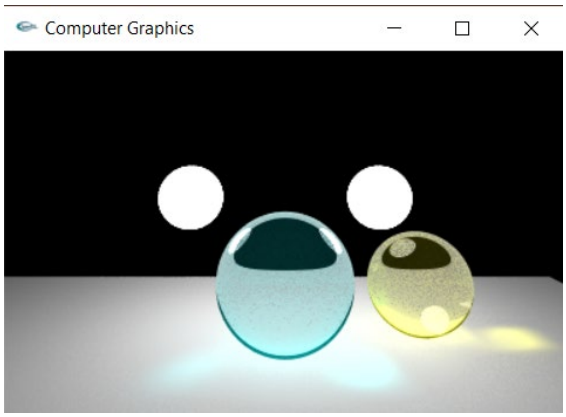

Figure 7: Caustics from different light sources

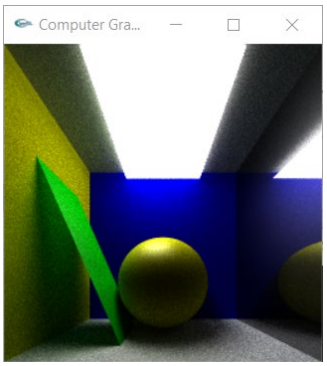

Figure 8: Specular Highlight with low-intensity light source

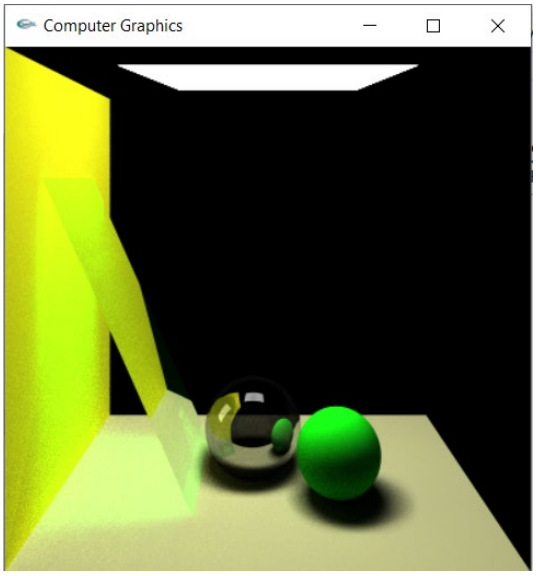

Figure 9: Mirror Ball, Diffuse Green Ball, and a Green Glass. The Photon Mapping technique is responsible to produce the greenish caustic on the wall and the ceiling

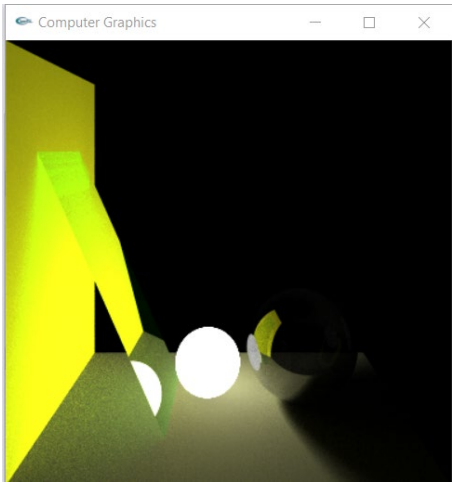

Figure 10: Spherical Light Source and Dual-natured Glass object supporting both Reflections and Refraction possible due to the Russian Roulette technique described in Section 4

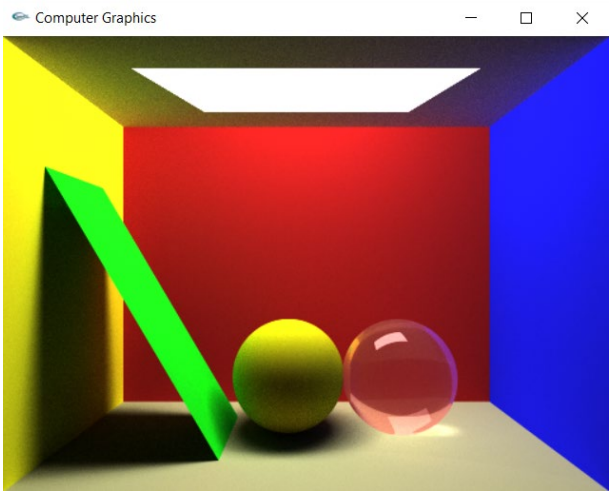

Figure 11: A classic Cornell Box Scene. The white caustic below the glass ball is possible due to the Photon Mapping

9. Extensions

Along with all the features proposed in the original proposal, the below extensions were also fulfilled:

1. **Photon Mapping:** A Photon Map Data structure (KD-Tree) was written from scratch and a caustic photon map was implemented.
2. **Adaptive Sampling:** To achieve faster rendering times, and intelligent pixel sampling, the depth of recursion for the path tracer was decided by first checking the variance of every pixel with the nearby pixels.

10.Challenges and Future Work

After two months of hard-work, reading research papers, viewing tutorials, speaking with Professors, studying books, and experimenting with my own code, I believe I have just scratched the surface with Physically Based Rendering. It was a nice experience to deal with such a project on my own and I have myself to congratulate and cheer when I am reaching the end of it. Having said that, I believe there are a lot more to achieve and a lot of ways to extend the current implementation. Here are a few thoughts for possible extensions:

- More advanced Light Transport methods like Bidirectional Path Tracing, Metropolis Light Transport, or even VCM (Vertex Connection and Merging) which combines Bidirectional Path Tracing and Photon Mapping to give excellent results.[10]
- A better analytical BRDF model like Cook-Torrance, Oren Nayar, or even a physically based BRDF database like MERL. [11]
- A better filtering technique like bilateral filtering or Non-Local Means Filtering will be a major step forward. [12]
- OBJ parsers are easy to write and would make a great addition to the project. Supporting a bounding volume hierarchy for storing the scene geometry will be a necessary pre-step before trying to ray-trace millions of triangles though.

11.Acknowledgements

I would like to thank Professor **Victor Zordan**, my teacher and mentor who incentivized me to take up this project and regularly gave feedback on my progress. I would also like to thank Professor **Daljit Singh Dhillon** for being kind enough to allow me to sit in his Rendering and Shading class, answering my doubts, and suggesting me books to read. I also have some anonymous colleagues to thank who provided me feedback on the quality of my renderings. I am most thankful for the fact that now I feel confident that I can talk with anyone about rendering for over an hour and be able to engage them in conversation.

12. References

[1] James Kajiya, The Rendering Equation, SIGGRAPH 1986

[2] Arnold: A Brute-Force Production Path Tracer

[3] Daniel Heckenberg, Christopher Kulla, Marc Droske, Jorge Schwarzhaupt Path tracing in Production, SIGGRAPH 2019 Course Notes

[4] Jason Lawrence: Importance Sampling of the Phong Reflectance Model

[5] E. Lafortune and Y. Willems. Using the modified Phong reflectance model for physically based rendering. Technical Report CW197, Dept. Comp. Sci., K.U. Leuven, 1994

[6] Eric Veach, Robust Monte Carlo Methods for Light Transport Simulation, 1997
[7] Wojciech Jarosz, Henrik Wann Jensen, Craig Donner: Advanced Global Illumination Using Photon Mapping, Siggraph 2008 Courses
[8] Robert Cook, Thomas Porter, Loren Carpenter, Distributed Ray Tracing
[9] Turner Whitted, "An Improved Illumination Model for Shaded Display," Communications of the ACM, vol. 23, pp. 343-349, 1980.
[10] Iliyan Georgiev, Jaroslav Kˇrivanek, Tomas Davidovic, Philipp Slusallek, Light Transport Simulation with Vertex Connection and Merging, Siggraph Asia 2012
[11] MERL Database : https://www.merl.com/brdf/
[12] Antoni Buades, Jean-Michel Morel, A non-local algorithm for image denoising
[13] https://www.scratchapixel.com/
[14] Greg Humphreys, Wenzel Jakob, Matt Pharr - Physically Based Rendering, 3rd Edition
[15] Kavita Bala, Philippe Bekaert, Philip Dutre - Advanced Global Illumination, 2nd Edition