Path Tracer - Exploring Physics Based Rendering

IGR Final Report

19 Feb 2026

Author : Baptiste GIRARDIN baptiste.girardin@telecom-paris.fr
School : **Télécom Paris**, France

Contents

Introduction

This project consists of the design and implementation of an interactive path tracing engine, executed entirely on the GPU via OpenGL 4.3 and GLSL shaders. It is based on Monte Carlo integral approximations coupled with temporal accumulation, enabling progressive convergence towards the solution of the rendering equation. Each pixel generates rays from a perspective camera which iteratively simulate light interactions (diffusion, specular reflection, refraction, emission) until they die, reach a light source or the environmental sky.

1.1 Objective

The main goal of this application is to explore as many physical models of light-matter interaction as possible, while maintaining a somewhat interactive architecture allowing dynamic adjustment of rendering parameters. This project was also made to handle the rendering of 3D models loaded from CPU to GPU, optimized by an acceleration structure. Lastly, a basic version of key-frames based 3D animations has been developed.

Technical Details

2.1 Rendering

This first part will focus on the techniques implemented in the fragment shader.

2.1.a A Useful Equation

Before explaining anything, it may be important to remind ourselves what we are working with. As briefly mentioned in the introduction, a path tracer shoots rays from the camera to estimate the inverse path of the photons using statistical approaches. As the ray is shot and intersects with the shapes in the scene (the equations of intersection will not be explained), we want to know how light reacts to those contacts. The light rendering equation is used for that:

$$\bullet\ L_o(x, \omega_o, \lambda) = L_e(x, \omega_o, \lambda) + L_r(x, \omega_o, \lambda)$$

$$\bullet\ L_r(x, \omega_o, \lambda) = \int_\Omega f_r(x, \omega_i, \omega_o, \lambda) L_i(x, \omega_i, \lambda)(\omega_i \cdot n) d\omega_i$$

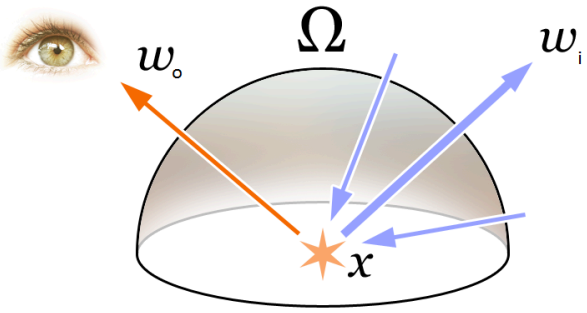

Figure 1: https://en.wikipedia.org/wiki/Rendering_equation#/media/File:Rendering_eq.png

- L_o : amount of light bouncing in the direction ω_o

- L_i : amount of light coming from the direction ω_i
- L_e : amount of light emitted at x in the direction ω_o
- λ : albedo of the object the ray just intersected with
- n : normal of the surface at x

2.1.b Multi-Importance Sampling (MIS)

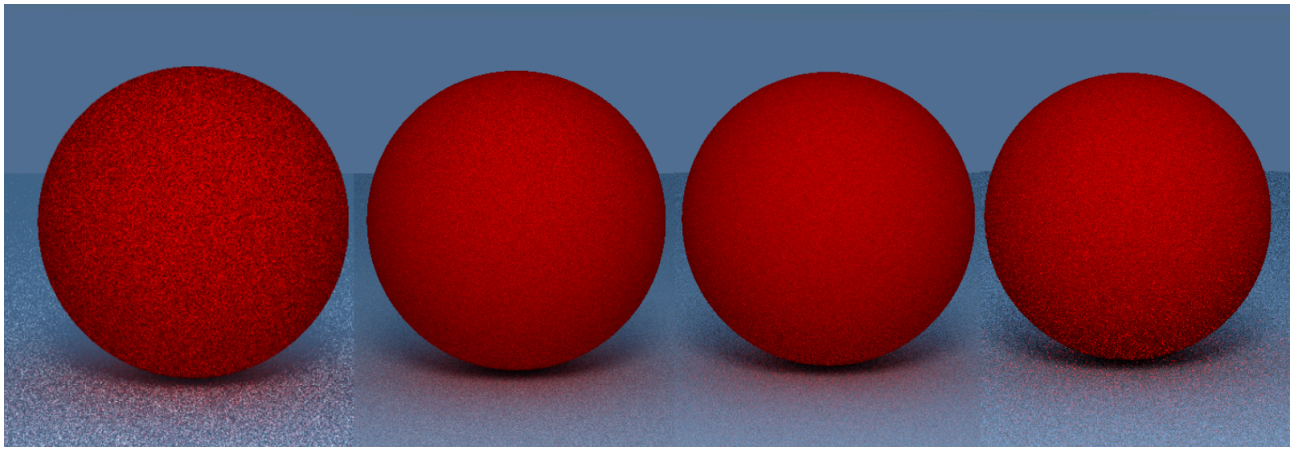

Figure 2: MIS between cosine hemisphere sampling and direct light sampling, accumulated on 75 frames. The coefficients are respectively $c_l = 0, c_l = 0.25, c_l = 0.5, c_l = 0.75$

The goal of the Monte-Carlo approach is to approximate the integral of L_r by shooting rays in random directions and weighting the throughput by the probability of shooting those rays.

I used a common way to enhance that method which works by shooting the rays towards privileged directions that emit the most amount of light. At each bounce the program decides wether it tries to directly sample towards the lights in the scene, or if it samples towards global illumination. This is what is MIS.
After sampling, our ray's throughput will be multiplied by that term:

$$\frac{f_r * (w_o \cdot n)}{c_g p_g + c_d p_d}$$

with c_i the probability of choosing the sampling technique associated with the probability density function (PDF) p_i, p_g being the PDF of the global illumination strategy and p_l the direct light sampling strategy.

2.1.c Diffuse Materials with Oren-Nayar

The first materials I implemented were rough diffuse materials. By default, a diffuse material with no roughness is a lambertian, whose reflection function is very well approximated by

$$f_r = \frac{\lambda}{\pi}$$

This is then extended by adding terms simulating roughness (such as for a rubber), using Oren-Nayar's Qualitative (QON) model [1] :

$$f_r = f_{r,\text{lambert}} * (A + B * g(w_i, w_o))$$

Multiple versions and approximations of this models have been published in the literature, but the ones that I focused on were the Fujii Oren-Nayar (FON) model and its improvement; the Energy-Preserving Oren-Nayar (EON) model [1].
As for the sampling technique I used the well known cosine hemisphere sampling, alongside MIS. By testing multiple values for the MIS factors, I noticed that it is around $c_g = 0.8, c_l = 0,2$ that the image converges the fastest.

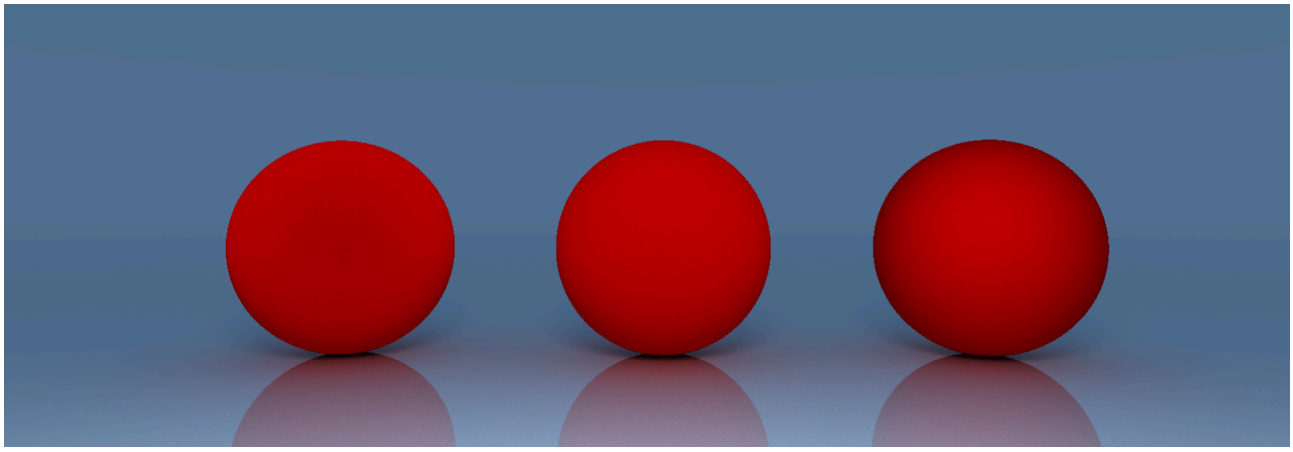

Figure 3: EON materials of roughness respectively $r = 1$, $r = 0.5$ and $r = 0$

2.1.d Metals with Cook-Torrance and GGX

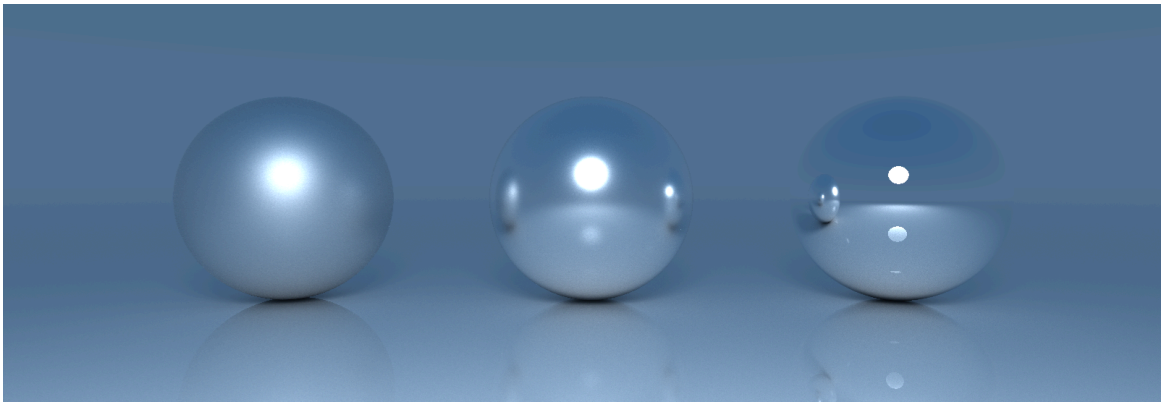

Figure 4: GGX materials of roughness respectively $r = 0.5, r = 0.25$ and $r = 0$

The next material I implemented was used to simulate rough metals. For this I used Cook-Torrance's equation [2] coupled with GGX's micro-facets model [3].

$$f_r = \frac{F * D * G}{4 * (n \cdot \omega_o) * (n \cdot \omega_i)}, \text{Cook-Torrance equation}$$

$$D_{\text{GGX}} = \frac{r^4}{\pi * \left[(n \cdot m)^2 (r^4 - 1) + 1\right]^2}, \text{GGX microfacets model}$$

$$G_{\text{GGX}}^{\text{Smith}} = G_1(n \cdot \omega_o)G_1(n \cdot \omega_i), \text{Geometry Term paired with GGX}$$

$$G_1(\theta) = \frac{2}{1 + \sqrt{1 + r^4 \frac{1-\theta^2}{\theta^2}}}$$

$$F_{\text{Schlick}} = F_0 + (1 - F_0)(1 - (\omega_o \cdot m))^5, \text{Schlick's approximation of Fresnel's function [4]}$$

As for the sampling technique, I used the one proposed by [3]. This technique samples the micro-facets' normal m, and not directly the ray. This is done according to this formula :

$$\theta_m = \arctan\left(\frac{r^2\sqrt{X_1}}{\sqrt{1 - X_1}}\right), \text{ Zenith angle relative to } n$$

$$\Phi_m = 2\pi X_2, \text{ Azimuth relative to } n$$

$X_1, X_2 \in \mathcal{U}([0; 1])$

For the MIS factors, $c_l = r^2 * 0.3$, $c_g = 1 - c_l$ work the best. Again, this was tested empirically on my side.

2.1.e Dielectrics

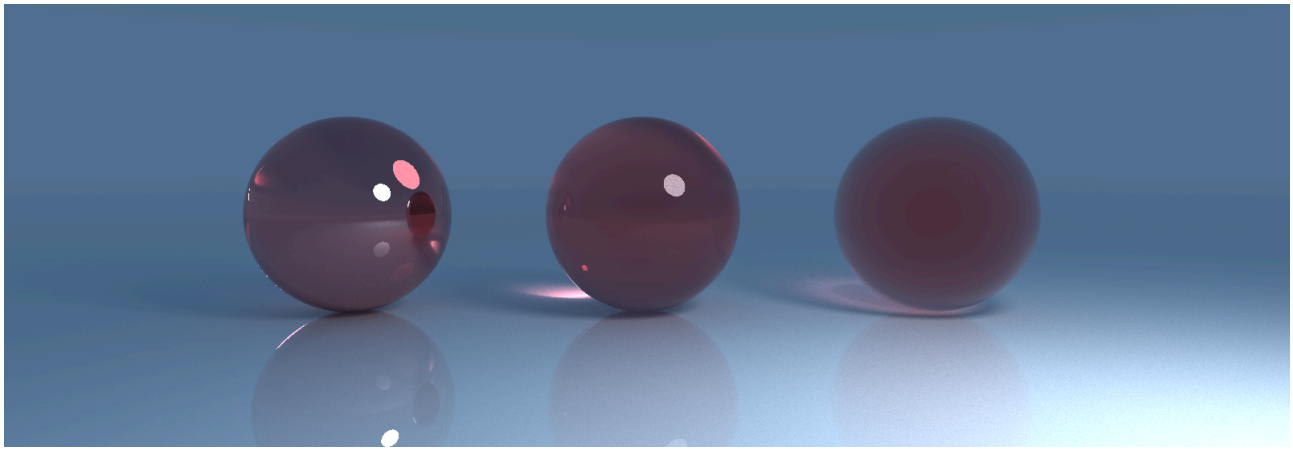

Figure 6: Dielectric materials of refraction index respectively $n = 4, n = 1.5$ and $n = 1.05$

One of the most beautiful feature about path tracers is being able to have caustics. To handle dielectrics and in particular how the ray's direction changes, the Fresnel-Descartes equation is a must. However, the ray splits up in a refracting and reflecting one with different energies, making the tracing of the ray recursive. This is not practical on a standart GPU, and so I used the Fresnel function (Figure 5) to decide the probability of choosing one or another, setting $F_0 = \left(\frac{1-n}{1+n}\right)^2$ with n being the refraction index of the dielectric.

I also wanted to handle dielectrics' color. This time I could not simply multiply the ray's throughput by the color after each intersection as dielectrics are a type of volume. For that I had to use Beer-Lamber's law :

$$\text{absorption} = e^{-(1-c)*d}, c \text{ the color and } d \text{ the distance traveled in the volume}$$

2.1.f Glossy Materials

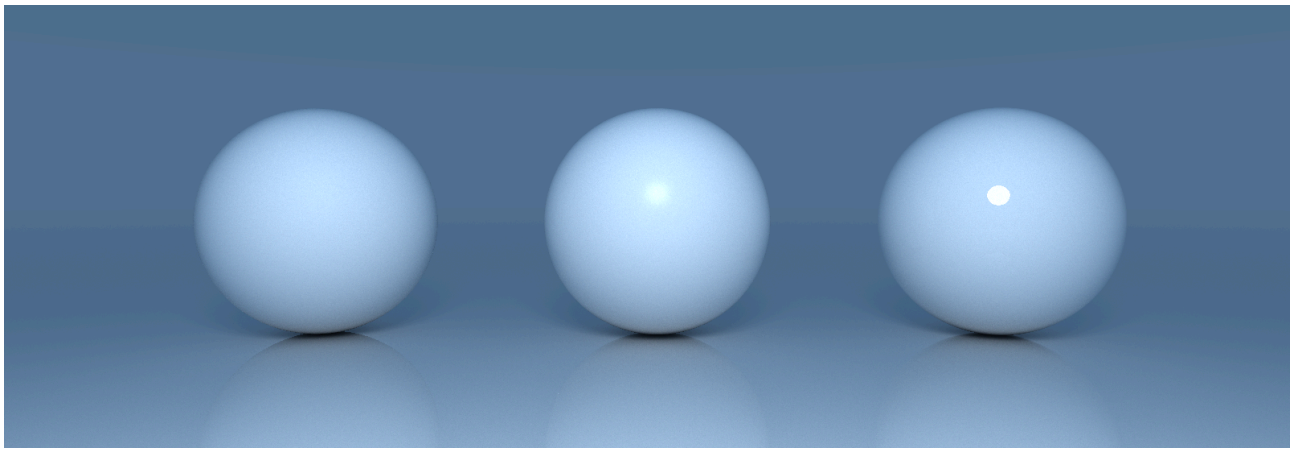

Figure 7: Glossy materials of roughness respectively $r = 0.5, r = 0.25$ and $r = 0$, and of metalness $m = 0.25$

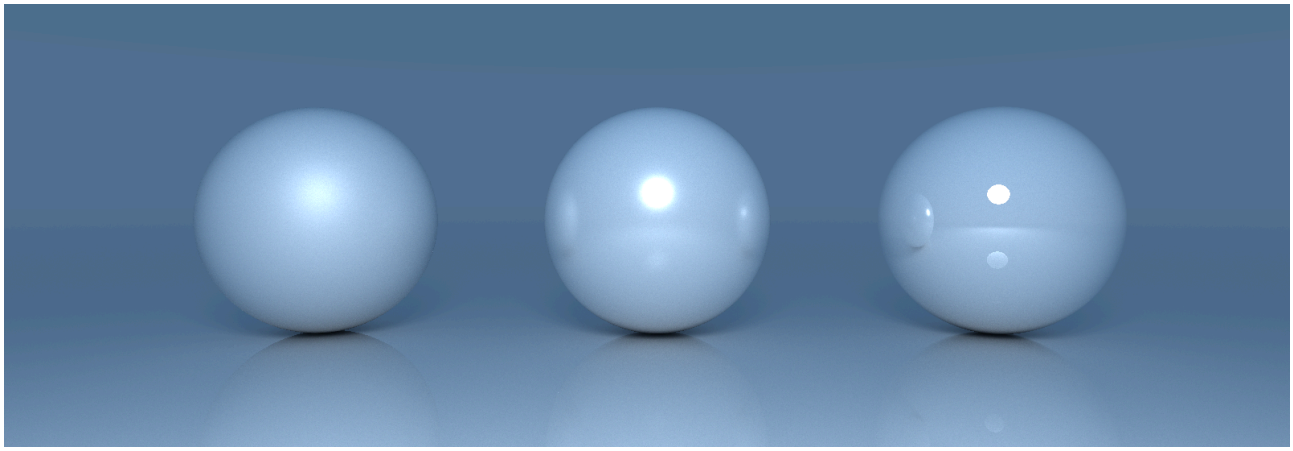

Figure 8: Glossy materials of roughness respectively $r = 0.5, r = 0.25$ and $r = 0$, and of metalness $m = 0.7$

Glossy materials are actually pretty simple. They are a mix of some of the materials presented above. As glossy materials are made of a tiny dielectric surface, and the interior is simply a diffuse material, I used again Fresnel's function (Figure 5) with $F_0 = \left(\frac{1- \text{ glossiness}}{1+ \text{ glossiness}}\right)^2$ to decide whether the ray bounces or is being diffused.

To take a step further, I made it so that the ray doesn't simply reflects when it does, but reacts like a GGX metal. Thus it is now possible to continuously transition from plastic materials to metal materials, and from smooth to rough.

2.1.g Some Handy Techniques

Once all those materials are implemented, I still had some small techniques to apply in order to make the renderer more efficient.

- First and foremost is the frame accumulation as mentioned in the introduction. I simply rendered each frame in a frame buffer, stored it in a texture, and reused it next frame to make a mean of all frame.
- Another technique I used is the Russian Roulette which decides after each bounce with a probability of the ray's throughput whether it should continue or not. This improves the computational time slightly while keeping the render almost identical.
- MIS is great for improving the rendering speed, but sometimes I observed the appearance of fireflies (Figure 9). To fix that, I looked at the code where the ray met a light and summed the radiance, and decided to clamp a part of the radiance added :

$$\text{added radiance} = \text{clamp}\ (\text{throughput} * w, 0, 1.3) * L_e$$

with w the weighting in the MIS. I chose 1.3 arbitrarily, but such that the result still has a good dynamic range while removing enough fireflies.

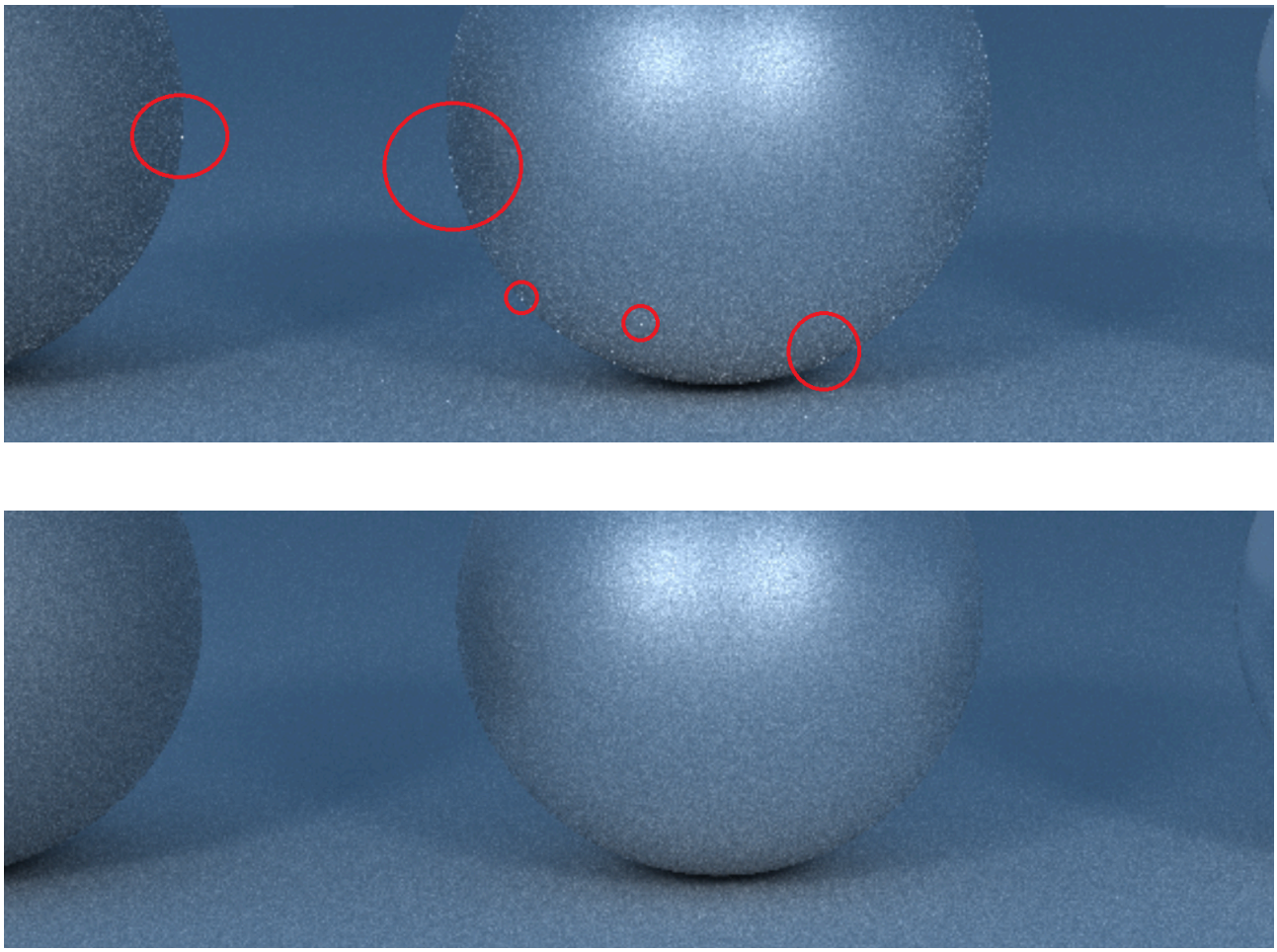

Figure 9: GGX metals rendered with 600 samples, without and with throughput clamping

2.2 Geometry

I then wanted to be able to render complex models. The issue is that currently the path tracer checks for every object in the scene at each intersection for each rays. And sometimes

models can be composed of thousands if not millions of triangles. In order to make the such a thing possible I implemented a simple version of the tree-based Bounding Volume Hierarchy (BVH) [5] acceleration structure, passing the tree linearized on the GPU with SSBOs.

Figure 10: On the left, we can see the bounding boxes of the acceleration structure of the 3D model on the right

2.3 Animation

For the final feature of this application, I decided to implement a very simple animation system based on key frames. On the CPU, I kept track of some the scene's informations in a `KeyFrame`, alongside its temporal position on the timeline. All the key frames are being stored in a `vector<KeyFrame>`, and when the *"Render Animation"* button is pressed, each frame of the scene is rendered, and the values are linearly interpolated from one key frame to another.

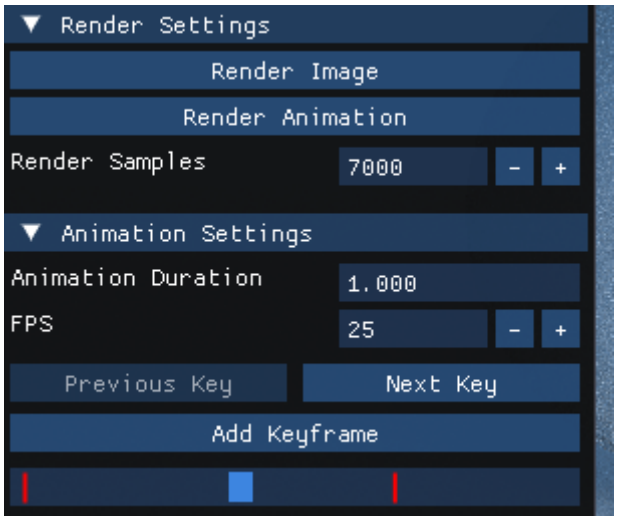

Figure 11: Screenshot of the animation and render settings of the application

Modifications & Improvements

3.1 Rendering

3.1.a EON's importance sampling

A first thing that could be improved in my diffuse materials is the way I sample the rays. [1] gives a good way to importance sample EON, alongside its associated PDF.

3.1.b Dielectrics

Right now the caustics take a lot of frames to denoise. With a variant of photon mapping, some form of importance sampling could be done.

Furthermore, the light reflection on Figure 6 are also quite noisy. Here a simpler form of MIS could also be done using direct light sampling in a clever way, as the surface of the dielectrics are smooth.

Finally, I think adding micro facets to dielectrics would be great

3.1.c Depth of Field

A good way to grasp the perspective and give cinematic results is to add some form of depth of fields. I think I will add this pretty soon in the future.

3.1.d Variance Sampling

Sometimes some pixels don't really need to be sampled anymore because the image has pretty much converged at that point. To make the renderer more efficient it would be very effective to compute a variance texture of the scene by looking at the spacial and temporal variations, and then to shoot more rays towards high variating regions and less rays towards low variating ones.

3.2 Geometry

3.2.a BVH Heuristic

Currently my BVH structure is a bit rough. The bounding boxes are computed according to the indexes of the triangles and their order along axes. A way to make it more efficient would have been to implement a sort of heuristic such as the Surface Area Heuristic (SAH).

3.3 Animation

3.3.a Physics Implementation

Linear interpolation is great, but physics are better. I think a simple way to enhance the animator would be to add some type of rigid body simulation, or even cloth or fluid simulation.

3.3.b Better Scene Management

Finally, apart from the 3D model loaded from CPU to GPU, everything in the scene is hard coded in the GPU. This is highly inefficient in terms of workload when trying to render things. Some type of scene loading from CPU would have been great.

Conclusion

This project allowed the design and implementation of a fully GPU-based interactive path tracer using OpenGL and GLSL. Starting from the rendering equation and its Monte Carlo approximation, the engine progressively evolved into a physically grounded renderer capable of simulating a wide range of light–matter interactions, including diffuse reflection (Lambert and Oren–Nayar), microfacet-based metallic materials (Cook–Torrance with GGX), dielectrics with Fresnel-based probabilistic branching, volumetric absorption using Beer–Lambert's law, and glossy layered materials.

A strong emphasis was placed on physically-based models and importance sampling strategies. The integration of Multi-Importance Sampling significantly improved convergence speed, while techniques such as Russian Roulette termination, frame accumulation, and throughput clamping helped balance visual quality and computational efficiency. The implementation of a BVH acceleration structure enabled the rendering of complex triangle meshes, making the engine scalable beyond simple procedural scenes.

Beyond pure rendering, the addition of a basic keyframe animation system demonstrated the flexibility of the architecture and opened the door to more advanced scene management and dynamic simulations.

Although the current version already produces visually convincing and physically consistent results, several improvements remain possible. More advanced sampling strategies, better BVH construction heuristics such as SAH, improved caustics handling, adaptive variance-based sampling, and extended physical simulations would further enhance both realism and performance.

Overall, this project required a deep practical understanding of physically based rendering, stochastic integration, GPU programming, and real-time graphics architecture. It highlights how modern rendering techniques combine mathematical rigor, physical modeling, and efficient parallel implementation to approximate the solution of the rendering equation with increasing realism.

I extremely enjoyed making this project and looking into the research papers, seeing how things turn out and how much room there always is for improvement. I will definitely continue to work on this project on a regular basis.

Bibliography

[1] J. Portsmouth, P. Kutz, and S. Hill, "EON: A practical energy-preserving rough diffuse BRDF." [Online]. Available: https://arxiv.org/abs/2410.18026

[2] D. Sawicki, "Modeling the metallic reflection using appropriate BRDF model," *Przeglad Elektrotechniczny*, vol. 83, pp. 24–27, 2007.

[3] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, in EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 195–206.

[4] Atrix256, "Raytracing Reflection, Refraction, Fresnel, Total Internal Reflection, and Beer's Law." [Online]. Available: https://blog.demofox.org/2017/01/09/raytracing-reflection-refraction-fresnel-total-internal-reflection-and-beers-law/

[5] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Trans. Graph.*, vol. 26, no. 1, p. 6–es, Jan. 2007, doi: 10.1145/1189762.1206075.