# Monte Carlo Ray Tracing

Abdelrahman Ameen
ID: 202201835
Zewail City of Science and Technology

***Abstract*—This project focuses on simulating the behavior of light in 3D environments using Monte Carlo methods to achieve photorealistic rendering [1]. By employing the rendering equation—which models the transfer of light within a scene—the project accounts for complex interactions such as global illumination, ambient occlusion, soft shadows, reflections, and refractions [2], [4]. Monte Carlo integration is utilized to approximate these interactions by randomly sampling possible light paths and averaging the results, providing an unbiased estimate of the scene's illumination [3], [4]. This approach enables the realistic depiction of indirect lighting effects—where light bounces multiple times between surfaces—and accurately portrays phenomena like the softening of shadows due to area light sources as well as the nuanced interplay of light as it reflects and refracts through different materials [5], [6].**

## I. Introduction

Realistic image synthesis in computer graphics necessitates accurate modeling of light transport within a scene [1]. The rendering equation, introduced by Kajiya, serves as a foundational model for this purpose, encapsulating both direct and indirect illumination [1]. However, solving this integral equation analytically is often infeasible due to the complexity of real-world scenes [2]. Monte Carlo integration offers a numerical approach to approximate the solution by statistically sampling light paths, making it a cornerstone in physically based rendering techniques [3], [4].

## II. Problem Definition

The core challenge lies in evaluating the rendering equation:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o)\, L_i(x,\omega_i)\, (\omega_i \cdot n)\, d\omega_i \tag{1}$$

where $L_o$ is outgoing radiance, $L_e$ emitted radiance, $f_r$ the BRDF, $L_i$ incoming radiance, $n$ the surface normal, and $\Omega$ the hemisphere of incoming directions [1], [4]. This integral accounts for all incoming light directions, making its direct computation computationally intensive, especially in scenes with multiple light interactions [2].

## III. Methodology

To approximate the rendering equation, we employ Monte Carlo integration:

$$L_o(x,\omega_o) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(x,\omega_i,\omega_o)\, L_i(x,\omega_i)\, (\omega_i \cdot n)}{p(\omega_i)} \tag{2}$$

where $N$ is the number of samples and $p(\omega_i)$ the sampling PDF [3], [4]. This stochastic approach allows simulation of complex lighting phenomena—including multiple light bounces, soft shadows, and caustics [5], [6].

For proof of concept, Python was used to validate our mathematical models and sampling strategies [7]. The final implementation utilizes C++ with Vulkan to leverage GPU acceleration on NVIDIA RTX cores, enabling real-time rendering of high-definition scenes [8].

## IV. Error Estimation

In Monte Carlo integration, the estimator's error is characterized by its variance. The standard error $\delta$ decreases as $1/\sqrt{N}$:

$$\delta \approx \frac{\sigma}{\sqrt{N}} \tag{3}$$

where $\sigma^2$ is the variance of the integrand [4], [9]. The sample variance is estimated by:

$$\sigma^2 \approx \frac{1}{N-1}\sum_{i=1}^{N}(f(x_i) - \bar{f})^2 \tag{4}$$

Variance reduction techniques—such as importance sampling and stratified sampling—lower $\sigma$ and speed convergence, directly impacting noise levels in the rendered images [3], [9].

## V. Results

Initial testing using Python demonstrated correct implementation but extremely high render times. Rendering a single 480p image with acceptable noise levels took nearly 24 hours due to the lack of parallelism. In contrast, the Vulkan implementation executed on an NVIDIA RTX GPU rendered similar images in under a second thanks to massive parallelization and hardware acceleration.

Vulkan output was smoother and featured less noise due to the ability to perform significantly more samples per pixel in a fraction of the time.

## VI. Comparison with Other Techniques

### A. Analytical Solutions

Analytical solutions to the rendering equation exist only for highly simplified scenarios—such as perfectly diffuse Lambertian surfaces in closed-form environments—which are rarely found in realistic rendering [2]. These methods offer high accuracy but are limited in scope.

### B. Finite Element Methods

Radiosity is a finite element method that works well for diffuse-only scenes. It solves for light transfer between surfaces but fails to capture specular reflections and caustics accurately [2].

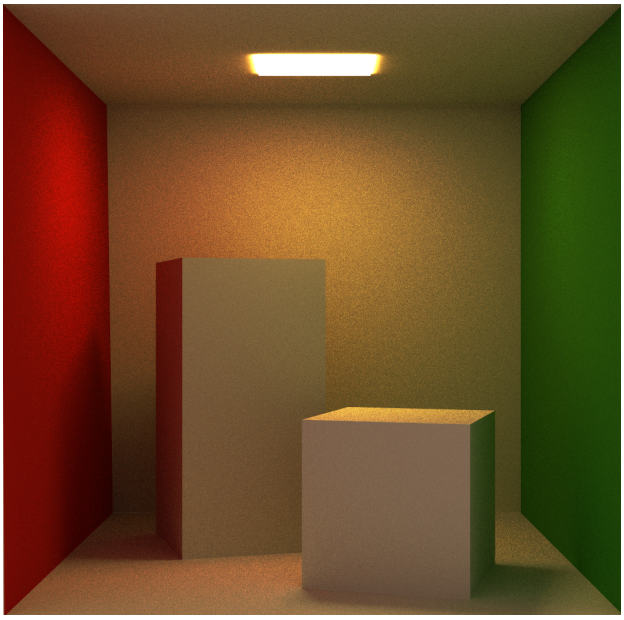

Fig. 1. Vulkan (GPU-accelerated) output

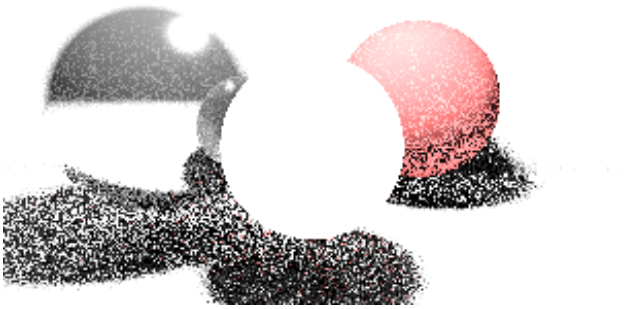

Fig. 2. Python (CPU-based) output after 24h

### C. Deterministic Ray Tracing

Conventional ray tracing techniques trace primary rays and reflections deterministically but lack stochastic sampling. These methods are faster but prone to aliasing and cannot handle global illumination without extensions like photon mapping or path tracing [9].

### D. Matlab Routines

Tools like MATLAB are often used for educational ray tracing experiments, but are not viable for high-performance or photorealistic rendering due to lack of GPU-level access and extensive libraries.

### E. Advantages of Monte Carlo

Monte Carlo methods offer unmatched generality and can handle all types of lighting interactions with unbiased results. However, they are noise-prone and computationally expensive without hardware acceleration [3], [4].

## VII. Conclusion

Monte Carlo ray tracing provides a powerful and physically accurate framework for simulating light transport. Despite its computational expense, the ability to incorporate complex lighting effects like global illumination and soft shadows makes it the technique of choice in modern rendering engines. While Python implementations offer accessibility for experimentation, high-performance applications necessitate GPU acceleration, as demonstrated with Vulkan and RTX hardware. Compared to analytical and deterministic methods, Monte Carlo techniques yield far superior visual fidelity at the cost of render time—an issue mitigated by parallel computing.

## References

## References

[1] J. T. Kajiya, "The rendering equation," in *Proc. 13th Annu. Conf. Comput. Graph. Interact. Tech.*, 1986, pp. 143–150.
[2] A. S. Glassner, *An Introduction to Ray Tracing*. San Diego, CA, USA: Academic Press, 1989.
[3] E. Veach, "Robust Monte Carlo methods for light transport simulation," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 1997.
[4] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2016.
[5] Graphics Compendium, "Chapter 38: Monte Carlo Ray Tracing," Graphics Compendium. [Online]. Available: https://graphicscompendium.com/raytracing/19-monte-carlo
[6] Scratchapixel, "Monte Carlo Methods in Practice," Scratchapixel. [Online]. Available: https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration.html
[7] Stanford Univ., "Monte Carlo Path Tracing The Rendering Equation," Stanford University, 2010. [Online]. Available: https://graphics.stanford.edu/courses/cs348b-10/lectures/path/path.pdf
[8] NVIDIA, "RTX Technology," NVIDIA Developer. [Online]. Available: https://developer.nvidia.com/rtx
[9] P. Shirley and R. Morley, *Realistic Ray Tracing*. A. K. Peters, Ltd., 2003.