FreezeRay - A Physically-Based Ray Tracer for Comparison of Rendering Equation Integrators

Daniel Elwell[1, 2] Kimberly Weston[2]
Sudarsun Kannan[1]

[1]Department of Computer Science, Rutgers University
[2]Department of Mathematics, Rutgers University

May 1, 2025

Abstract

In order to generate realistic images in a physically-based manner, most renderers attempt to solve what is known as the rendering equation, which is a complex intregral which cannot be easily solved analytically in the general case. Thus, renderers often approximate this integral using monte carlo integration techniques. In this project, we present FreezeRay, which is a physically-based renderer capable of rendering photorealistic images using ray tracing. FreezeRay implements support for multiple separate monte carlo integration techniques, and allows for the same scene to be rendered with each seamlessly. We will present an overview of the implementation of FreezeRay, including it's primary features and optimizations. We will also present a basic comparison of the integration techniques implemented within the program. The source code is relatively simple, and intended to be suitable for pedagogical purposes.

1 Background & Motivation

Computer graphics have a wide-range of applications across fields. For uses such as scientific visualizations or CGI for movies, it is often desireable to produce images in a completely realistic manner. For this, Physically-based rendering (PBR) techniques are often employed. PBR techniques render images according to physical principles rather than simplified approximations. At its core, PBR aims to solve the rendering equation, which serves a mathematical foundation for light transport simulation. The rendering equation,

first formalized by Kajiya [2], is often expressed as:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + L_r(x, \omega_o) \tag{1}$$

$$= L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i, \tag{2}$$

where $L_o(x, \omega_o)$ represents the outgoing radiance from a point x in a direction ω_o, $L_e(x, \omega_0)$ is the emitted radiance from x towards ω_o, f_r is the bidirectional reflectance distribution function (BRDF), $L_i(x, \omega_i)$ is the incoming radiance from a direction ω_i to x, and $(\omega_i \cdot n)$ is the cosine of the angle between the surface normal at x and the direction of incoming radiance. The integration is performed over the hemisphere of directions above the surface point x, Ω. $L_i(x, \omega_i) = L_o(y, \omega)$, where y is the point seen by x in the direction ω_i, thus the integral is recursive. The rendering equation presents a serious computational challenge for a number of reasons:

- It is a complex, recursive intregral.
- The integrand is often discontinuous and contains high-frequency detail.
- No analytic solution exists for a general scene.

In order to approximate solutions, Monte Carlo method predominate. These techniques estimate the integral by sampling the domain at random points, and averaging the results over many samples. Many different Monte Carlo integration methods have been developed for the rendering equation, such as path tracing, bidirectional path tracing [3], and Metropolis light transport [8]. These methods provide a framework to estimate the rendering equation in an unbiased manner, but present their own sets of challenges in implementation:

- They often require a deep understanding of probability theory and other mathematical concepts.
- Computational efficiency and convergence vary greatly across different scenes.
- Computational efficiency and convergence vary greatly across different lighting scenarios.

These challenges make implementation difficult, and often certain methods will perform much better than others for a given scene. For example, Metropolis light transport excels at rendering caustics, but for a scene dominated by simple diffuse materials, path tracing may be more computationally efficient.

Additionally, for each of these methods, there exist a variety of additional techniques, such as importance sampling [7] and next event estimation, which require further understanding and development time. For this reason, we believe that a single program implementing many of the most common Monte Carlo rendering techniques seamlessly represents a significant help to developers and artists alike. We hope that a simple and complete implementation will enable faster development and foster understanding through pedagogical source code.

2 Overview of FreezeRay

We present FreezeRay. FreezeRay is a physically-based renderer implementing many of the most common Monte Carlo integration techniqes as well as many variance reduction techniques. It is based on ray tracing. Some of its most notable features include:

- Many physically-based BRDF models.
- Many physically-based materials, composed of one or more BRDFs.
- Many physically-based light sources.
- Ability to render any scene composed of triangle meshes, loaded from 3D object files.
- A simple interface for rendering the same scene using multiple different integration techniques, and numerically comapring the results.

Additionally, the code is fully open source and written simply with many comments, so it is quite suitable for pedagogical purposes. The Monte Carlo integrators it currently supports are:

- Path tracing.
- Bidirectional path tracing [3].
- Metropolis light transport [8].

Each of these techniques comes with many additional variance reductiom techniques as well. The source code was architected such that addition of new techniques is seamless and requires little development overhead. The project has also been well optimized at a low level, and is able to render complex scenes quite efficiently even on consumer hardware.

3 Methods

The development of FreezeRay was split into two main stages:

3.1 Literature Review

We began by conducting an extensive review of relevant literature surrounding monte carlo integration methods for the rendering equation. The purpose of this review was to identify commonly used and influential algorithms in light transport to consider for integration into FreezeRay. Specifically, we considered algorithms and variance-reduction techniques which were commonly used, computationally

efficient, suitable for different scene types, or just generally interesting. We primarly used Google Scholar to identify papers to review.

Additionally, we read and analyzed some foundational works in physically-based rendering to guide development of the core architecure and systems for FreezeRay. These included Kajiya's seminal paper on the rendering equation [2] and Nicodemus' on the BRDF [4].

3.2 Implementation

Drawing on our literature review of foundational works, we developed the core architecture of FreezeRay. The program is implemented in C++ using a modern C++ style object-oriented paradigm. We made this decision as C++ can be highly performant, but still remain flexible with an object-oriented design. This allowed us to add new BRDFs, materials, and integrators with very little development overhead. Much of the high-level architecture of the program was inspired by Pharr, Jakob, and Humphrey's *Physically Based Rendering: From Theory to Implementation* [6] and it's accompanying source code.

For our scene description, we opted to support only triangle meshes. We made this decision as triangle meshes are the most widely used form of 3D object description, with many full scenes readily available online. We did not add support for generalized geometries, such as implicit geometry, as we wanted to allow triangle mesh rendering to be highly optimized.

We included support for many physically-based components in light transport to allow for a wide range of lighting effects to be simulated. Most notably:

- **BSDF Models**: We implemented many physically-based bidirectional scattering distribution function (BSDF) models into FreezeRay. These included models for diffuse, perfect and rough specular, and refractive transmittance. We also included Fresnel models for both dielectrics and conductors.

- **Materials**: We implemented a system for creating a material composed of one or more BSDFs, allowing for more complex scattering properties. Our materials allow for texture-driven parameter mapping.

- **Light Sources**: We implemented many different light sources to enable a variety of lighting scenarios. These include physically-based light sources such as area lights and environment maps as well as analytical lights such as point lights or directional lights.

In order to optimize rendering performance, we performed a number of low-level optimization techniques. These allowed FreezeRay to generate high-quality images in just a few minutes on consumer-grade hardware. The most effective of these were:

- **Multithreading**: We implemented a task-based parallel processing system to render multiple regions of the image simultaneously, leveraging multi-core CPUs.

- **Hierarchical Data Structures**: We implemented a KD-tree to accelerate ray-scene intersection. The tree groups spatially close triangles together to allow for entire groups of triangles to be skipped during interection.
- **SIMD Vectorization**: We compute ray-triangle intersection with multiple triangles at once by leverating SIMD intrinsics, reducing ray-scene intersection cost.
- **Cache-Friendly Texture Access**: We store textures in a hierarchical grid to decrease the liklihood of cache misses when nearby texels are queried.

Again drawing on our literature review, we implemented several Monte Carlo integrators to approximate the rendering equation:

- **Path Tracing (PT)**: A relatively simple, unidirectional integration method. PT traces paths of light beginning at the camera, bouncing them randomly at each surface hit, until they either hit a light source or escape the scene bounds. At each step in the path, a point on one of the scene's light sources is sampled, and a "shadow ray" is traced towards it to sample direct lighting.
- **Bidirectional Path Tracing (BDPT)**: A generalization of path tracing. BDPT traces subpaths of light beginning both at the camera and at points on scene light sources. Attemps are then made to connect these subpaths at each possible pair of vertices, creating full paths from camera to light source [3].
- **Metropolis Light Transport (MLT)**: An integrator based on the Metropolis-Hastings sampling algorithm. MLT generates paths of light using BDPT, then slightly mutates those paths transporting high relative radiance to generate more high-radiance paths with higher probability [8].

Additionally, we implemented several variance-reduction techniques on top of the integrators:

- **Importance Sampling**: Instead of choosing a uniformly-random direction in which to bounce a light path after it hits a surface, importance sampling says to sample this direction from a distribution relative to the surface's BRDF. This means that directions reflecting more light will be sampled more frequently [7].
- **Multiple Importance Sampling (MIS)**: When sampling a direction towards a light source for a "shadow ray," MIS says to sample two directions: one from relative to the BRDF of the current surface point, and another relative to the distribution of points on all light sources. Then, these samples are weighted based on the density of their probability distributions. This allows multiple distributions to be combined and sampled from simultaneously [7].

To evaluate the relative performance of the integrators and variance reduction techniques, we implemented support in FreezeRay for rendering identical scenes with different techniques. We also added basic functionality for computing quantitative differences between the rendered images, such as the mean-squared error (MSE) against a reference image.

4 Results

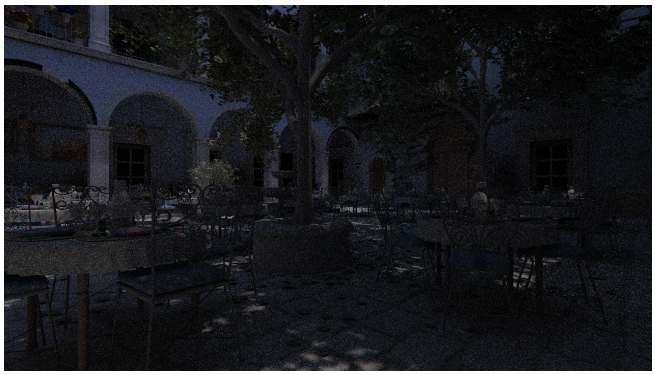
To demonstrate the capabilities of FreezeRay, we present renders of various well-known scenes in the field of computer graphics, using different integrators and variance reduction techniques. Each render was performed on a laptop equipped with an 8-core Ryzen 9 5900HX processor.

4.1 Comparison of Integrators

4.1.1 San Miguel

The San Miguel scene is an outdoor scene with very complex gemoetry. For our render, we lit it with a single environment map. A download can be found here.

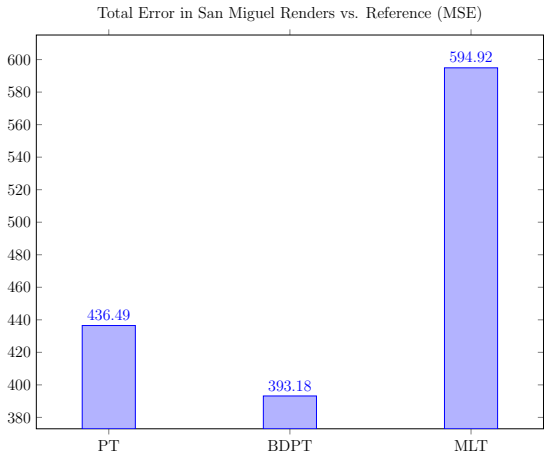
(a) PT, 16 samples/pixel (∼8min render time)



(b) BDPT, 12 samples/pixel (∼8min render time)

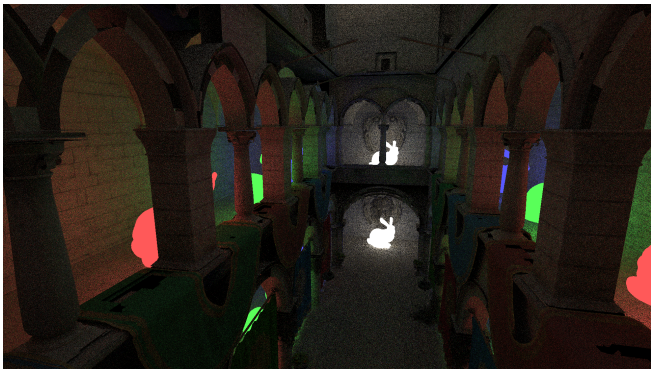
(c) MLT, 30 mutations/pixel (∼8min render time)

(d) Reference

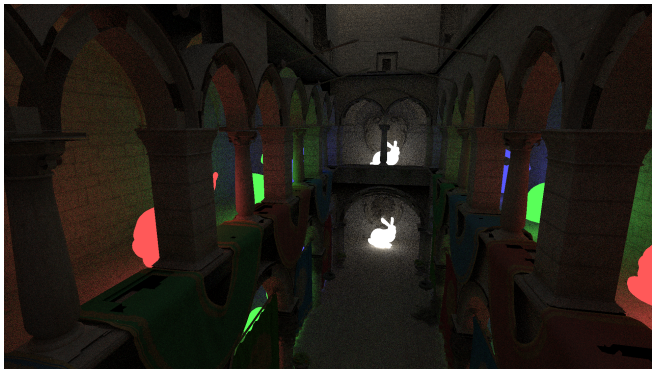


4.1.2 Sponza

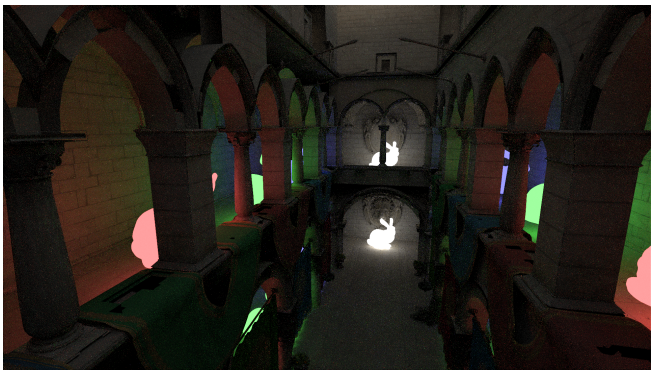
The Sponza scene is a partially outdoor and indoor scene with moderately complex gemoetry. For our render, we lit it with many area lights as well as a nightime environment map. Downloads for the scene as well as the mesh for the area lights can be found here.



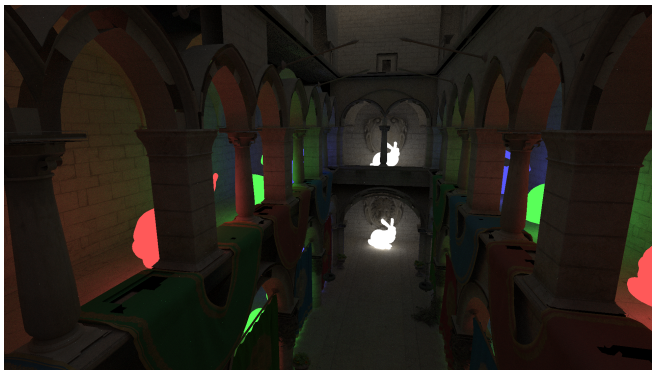
(a) PT, 64 samples/pixel (∼7min render time)



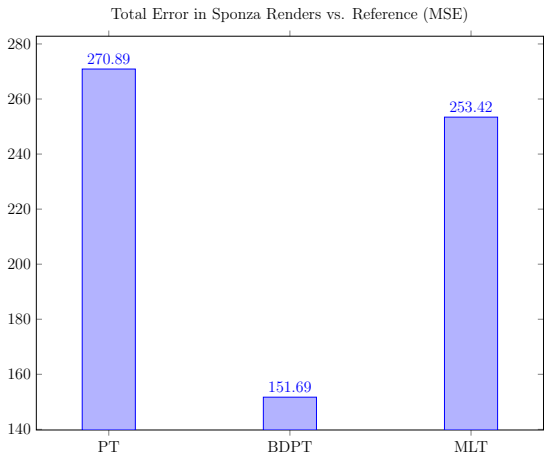
(b) BDPT, 20 samples/pixel (∼7min render time)



(c) MLT, 175 mutations/pixel (∼7min render time)

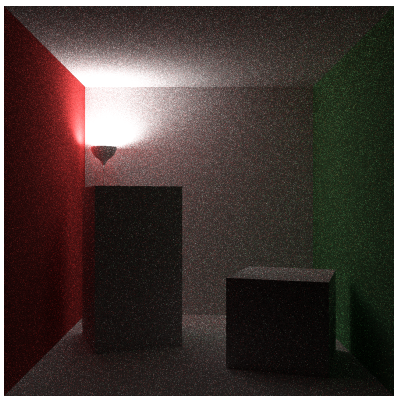


(d) Reference

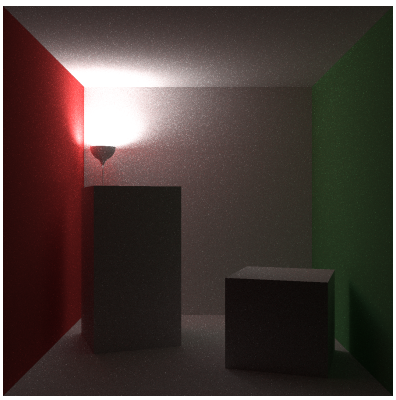


4.1.3 Cornell Box

The Cornell Box scene is an indoor scene with simple geometry. For our render, we lit it with a small lamp in the corner. This is different from the traditional Cornell Box, where the light comes from the ceiling, but we hoped to create a scene where more complex light paths are required to transport radiance.

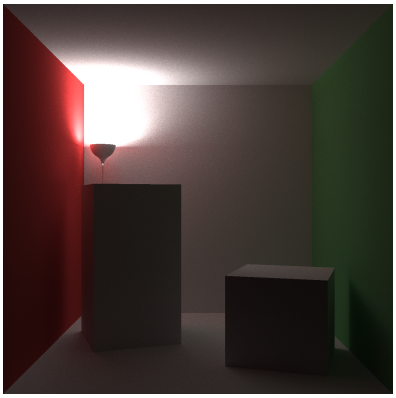


(a) PT, 256 samples/pixel (∼7min render time)

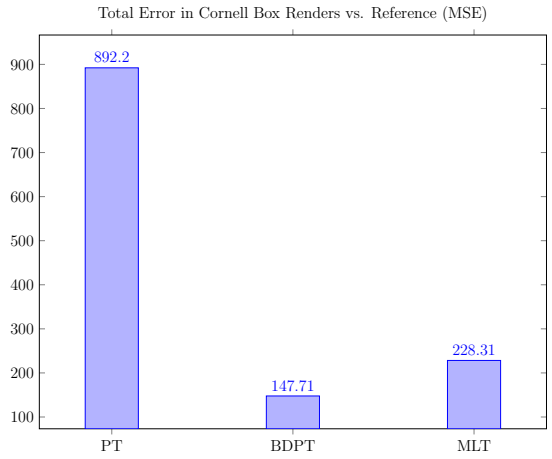


(b) BDPT, 32 samples/pixel (∼7min render time)

(c) MLT, 275 mutations/pixel (∼7min render time)



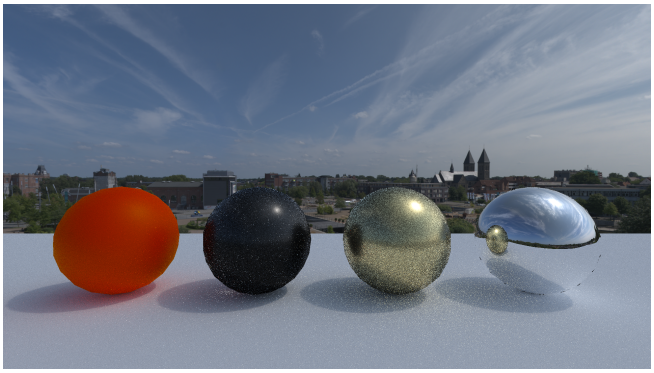
(d) Reference



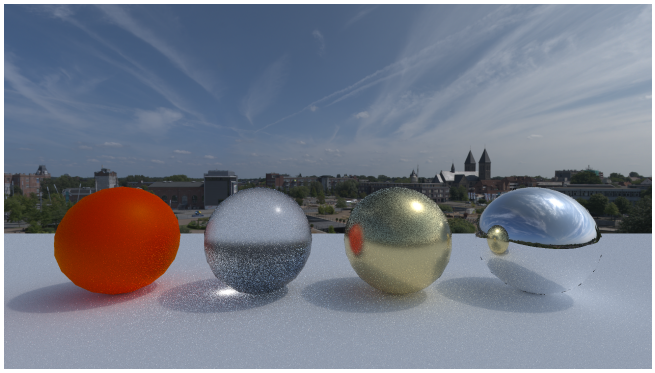
4.2 Variance-Reduction Techniques

To demonstate the effectiveness of the variance-reduction techniques implemented in FreezeRay, we present a benchmark image consisting of multiple spheres of different materials. They are, from left to right, plastic, rough glass, gold, and mirror.

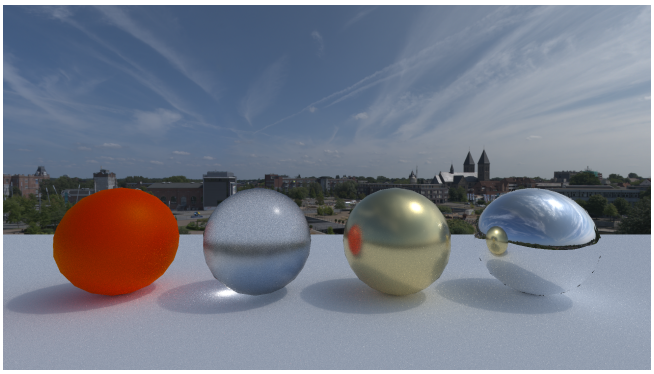
Note how the each variance reduction technique dramatically improves the image quality in specific regions.



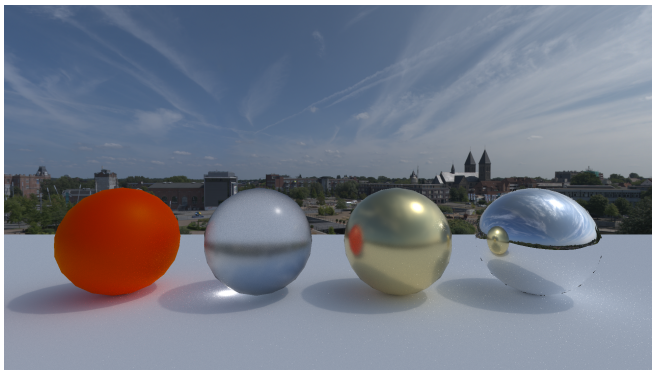
(a) Naïve PT, 512 samples/pixel (∼2min render)



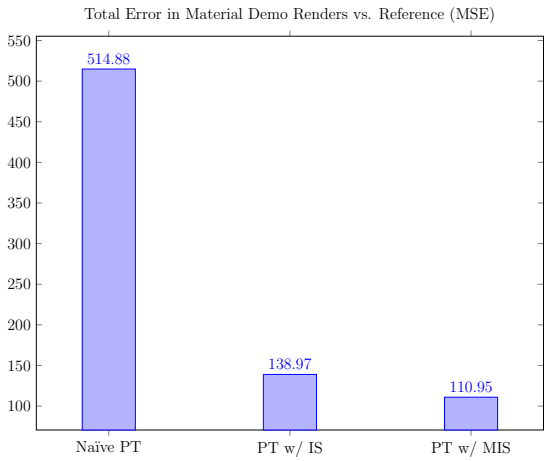
(b) PT w/ IS, 512 samples/pixel (∼2min render)



(c) PT w/ MIS, 256 samples/pixel (∼2min render)



(d) Reference



5 Future Work

In order to build on the core program architecture and research done during this project, we have identified several additions and improvements we would like to make to FreezeRay in the future.

We would like to implement several more Monte Carlo integrators for evaluation and for greater flexibility in rendering, specifically:

- **Photon Mapping**: Photon mapping is a biased, two-pass technique that first traces light rays from sources and deposits then onto the scene through a "photon map." The "photon map" is then used to approximate the radiance at each point seen by the camera [1].
- **ReSTIR GI**: Reservoir-based Spatiotemporal Importance Resampling for Global Illumination (ReSTIR GI) is a recent advancement designed for real-time rendering that uses temporal and spatial resampling to greatly reduce variance. It has both biased and unbiased variants [5].

Additionally, we intend to expand the functionality of the core FreezeRay program to allow for a greater variety of lighting scenarios and scenes to be rendered. We plan to incorporate the features:

- **Participating Media**: Participating media refers to volumetric objects such as fog or smoke. This will allow much more realistic scenes to be rendered.
- **Lens Simulation**: We plan to extend our current camera model to be physically-based, allowing for simulation of effects such as depth of field and bokeh.
- **Further Optimization**: We plan to push the performance of FreezeRay even further by optimizing more aggresively. Most notably, we hope to accelerate rendering using the GPU, allowing for a potentially exponential speedup.

Perhaps most importantly, we hope to develop an even more comprehensive framework for evaluating monte carlo integrators. We plan to add more quantitative metrics to compare renders, as well as automatic estimates of visual quality. This, we hope, will help to bridge the gap between the deep understanding of theory required to implement these integrators with the actual perceived results on a variety of scenes.

References

[1] Henrik Wann Jensen. "Global illumination using photon maps". In: *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Porto, Portugal: Springer-Verlag, 1996, pp. 21–30. ISBN: 3211828834.

[2] James T. Kajiya. "The rendering equation". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: https://doi.org/10.1145/15886.15902.

[3] Eric Lafortune and Yves Willems. "Bi-Directional Path Tracing". In: *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics' 93* (Jan. 1998).

[4] Fred E. Nicodemus. "Directional Reflectance and Emissivity of an Opaque Surface". In: *Appl. Opt.* 4.7 (July 1965), pp. 767–775. DOI: 10.1364/AO.4.000767. URL: https://opg.optica.org/ao/abstract.cfm?URI=ao-4-7-767.

[5] Y. Ouyang et al. "ReSTIR GI: Path Resampling for Real-Time Path Tracing". In: *Computer Graphics Forum* 40.8 (2021), pp. 17–29. DOI: https://doi.org/10.1111/cgf.14378.

[6] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd ed. Morgan Kaufmann, 2016. URL: https://www.pbr-book.org/3ed-2018/contents.

[7] Surya T. Tokdar and Robert E. Kass. "Importance sampling: a review". In: *WIREs Computational Statistics* 2.1 (2010), pp. 54–60. DOI: https://doi.org/10.1002/wics.56.

[8] Eric Veach and Leonidas J. Guibas. "Metropolis light transport". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76. ISBN: 0897918967. DOI: 10.1145/258734.258775. URL: https://doi.org/10.1145/258734.258775.