Analytic and Neural Approaches for Complex Light Transport

A thesis submitted in partial fulfillment
of the requirements for the degree of

Masters of Science
in
Computer Science and Engineering by Research

by

Ishaan Shah
2019111028
ishaan.shah@research.iiit.ac.in

Advisor: Prof. P. J. Narayanan
Co-Advisor: Dr. Adrien Gruson
Co-Advisor: Dr. Luis E. Gamboa

International Institute of Information Technology Hyderabad
500 032, INDIA

April 2024

International Institute of Information Technology Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that work presented in this thesis titled ***Analytic and Neural Approaches for Complex Light Transport*** by *Ishaan Shah* has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Advisor: Prof. P. J. Narayanan
Date	Co-Advisor: Dr. Adrien Gruson
Date	Co-Advisor: Dr. Luis E. Gamboa

To, my parents...

Acknowledgements

The work in this thesis wouldn't have been possible without immense support from my family, friends and my advisors.

Over the course of my Master's thesis I have been in a fortunate position of having multiple advisors who have helped me in my research journey. I would like to thank first and foremost my advisor Prof. P. J. Narayanan, who saw my interest in rendering and graphics and accepted me as his student. His experience in graphics and high performance GPU computing has been immensely helpful in my research. I also appreciate the amount of time he spends on research given his other equally demanding responsibilities as the director. I would also like to thank my co-advisors Prof. Adrien Gruson and Prof. Luis Gamboa. I have had a incredibly fruitful collaboration with both of them over the past year. Their expertise in rendering has been very helpful in improving my understanding of the field. Finally, I would like to thank Prof. Avinash Sharma who was my advisor before he left to join IIT Jodhpur. He was very supportive of my wildly varying research interests in the beginning of my research journey and has been instrumental in shaping my research path.

I would like to thank both my parents for creating an environment where I could learn and explore my interests. They have been incredibly supportive about my love of computers from a young age and have always encouraged and enabled me to pursue this interest. My mom has made sure that I never wavered in my academics and has helped me build strong theoretical foundations without which I would not be able to be where I am today. My dad who is an architect, has imparted invaluable practical knowledge which has helped me deal with the real world. My initial fascination in rendering began as I learnt about CAD and 3D modelling from seeing him work. He has taught me the importance of attention to detail which has greatly helped me in my research journey.

I've been fortunate to have supportive seniors and friends who've provided technical guidance and given valuable life advice. I would like to thank Aakash K. T. who introduced me to the field of rendering and helped me through the steep initial learning curve. He has consistently played a crucial role in guiding me through significant life choices and serves as my primary source of advice on research-related matters. I would also like to thank Chandradeep Pokhariya, who was my mentor in the initial days of my research journey. It was great working with him on what I consider my first research project. I'm grateful to my two closest friends, Rahul Goel and Sriram Devata, for making my four years of Bachelor's a truly enjoyable and unforgettable experience. Finally, I would also like to thank all of my friends and labmates at CVIT: Amogh, Astitva, B.V.K, Dhawal, Naren, Pranav and Shanthika for their support and help and making the lab a fun place to be in.

Abstract

The goal of rendering is to produce a photorealistic image of the given 3D scene description. Physically based rendering simulates the physics of the light as it travels and interacts with objects in the scene before finally reaching the camera sensor. Monte Carlo methods have been the go-to approach for physically based rendering. They are general and robust but introduce noise and are computationally expensive. Recent advancements in hardware, algorithms, and denoising techniques have enabled real-time applications of Monte Carlo methods. However, complex scenes still demand high sample counts. In this thesis, we explore and present the utilization of analytic and neural approaches for physically based rendering. Analytic methods offer noise-free renderings but are less general and may introduce bias. In recent years, neural-based approaches have gained traction, offering a balance between generality and computational efficiency. We compare and contrast the traditional Monte Carlo-based methods and emerging analytic and neural network-based methods. We then propose analytic and neural solutions to two challenging cases: direct lighting with many area lights and efficient rendering of glinty appearances on specular normal-mapped surfaces.

Direct lighting from many area light sources is challenging due to variance from both choosing an important light and then a point on it. Existing methods weigh the contribution of all lights by estimating their effect on the shading point. We propose to extend one such method by using analytic methods to improve the estimation of the light's contribution. This enhancement accelerates the convergence of the algorithm, making it more efficient for scenes with many dynamic lights.

The second case deals with the challenge of rendering glinty appearances on normal mapped specular surfaces efficiently. Traditional Monte Carlo methods struggle with this task due to the rapidly changing spatial characteristics of microstructures. Our solution introduces a novel method supporting spatially varying roughness based on a neural histogram, offering both memory and compute efficiency. Additionally, full direct illumination integration is computed analytically for all light directions with minimal computational effort, resulting in improved quality compared to previous approaches.

Through comprehensive analysis and experimentation, this thesis contributes to the advancement of rendering techniques, shedding light on the trade-offs between different methods and providing insights into their practical applications for achieving photorealistic rendering.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rendering is the process of generating an image from a detailed scene description. Accurately representing and rendering scenes in a photorealistic manner has been a long-standing goal in computer graphics. To achieve photorealistic renderings, one needs to simulate the interaction of light as it bounces throughout the scene before reaching the camera. This process of tracking the trajectory of light as it travels and interacts with the scene to render an image is called Physically Based Rendering (PBR). Since we follow the laws of physics, the results obtained from PBR are photorealistic and consistent with the real world. To achieve PBR, one needs to solve the Light Transport Equation (LTE). The LTE describes how light interacts with objects and travels throughout the scene. It is an infinitely high dimensional integral with no closed-formed solution available.

Currently, Monte Carlo-based methods are the most general and robust methods for solving the LTE. They are agnostic to the complexity of the scene geometry, reflection models, and lighting models and are guaranteed to converge to the correct answer. Monte Carlo methods operate by randomly sampling multiple paths in the scene and averaging their results to solve the LTE. This process of random sampling manifests as noise in the image. Increasing the number of random samples reduces the noise but is computationally expensive.

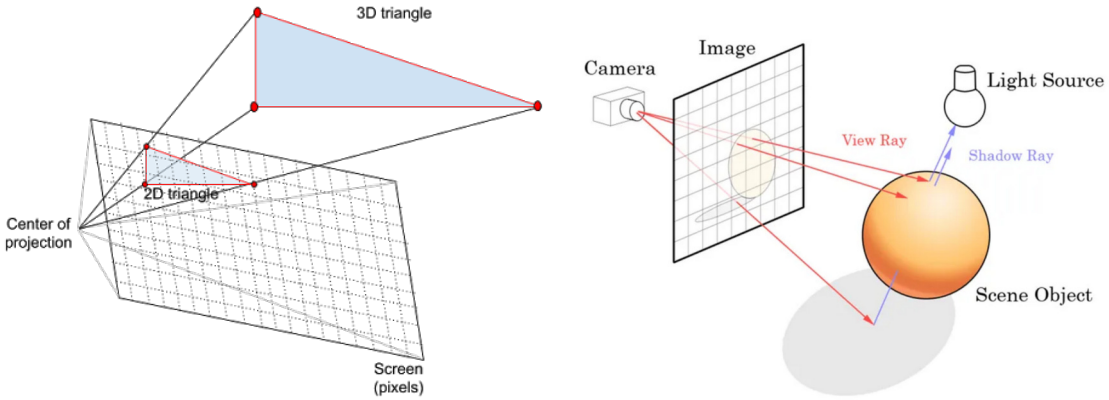
As discussed before, a closed-form solution to the LTE is not available in the general case. However, by making some assumptions, the LTE can be simplified, and a closed-form expression to solve the simplified LTE can be obtained. The exact assumptions made and the analytic method depend upon the problem being solved. Since no random sampling is involved, rendering produced by analytic methods is noise-free. Furthermore, analytic methods are generally cheaper as they only involve evaluating a mathematical expression. However, analytic methods are less general and often introduce some amount of known bias.

Neural networks have been increasing in popularity in recent years. Neural network-based approaches have been used in the rendering to varying extents, ranging from representing a complex BSDF to learning the radiance distribution of the scene. Neural approaches are general but must be carefully tuned to balance the runtime, quality, and bias induced based on their use case.

The goal of this thesis is to explore and present the usage of analytic and neural approaches and their application for solving light transport problems. We begin by providing a formal definition of rendering and presenting an overview of the two commonly employed algorithms for rendering. We then explain

the meaning of bias and variance in context of rendering algorithms and their implications. Finally, we provide a detailed discussion of two setups involving complex light transport, for which we propose solutions. The first one deals with direct lighting in the presence of many area lights, and the second deals with efficient rendering of glinty appearance of specular normal-mapped surfaces.

1.1 Rendering



(a) Rasterization.
Image Courtesy: Mariano Trebino.

(b) Ray Tracing.
Image Courtesy: NVIDIA.

Figure 1.1: Comparison between rasterization and ray tracing. In rasterization, we project the vertices to the image plane and check if the pixel lies inside the projection. In ray tracing, we cast a ray through each pixel and check if the ray intersects the object.

Rendering is the process of taking a 3D scene and producing a 2D image of the scene as seen from a camera. The scene can consist of multiple models with different surface properties and lighting. A rendering algorithm takes this 3D scene description and a camera as input and produces a 2D image of the world as seen from the camera. There are two broad types of rendering algorithms: rasterization and ray tracing. Each method has a set of strengths and weaknesses, which we will discuss in the next sections.

1.1.1 Rasterization

The rasterization algorithm can be broken into two stages. The first stage projects all of the scene geometry onto the image plane. In the second stage, there is an iteration over all pixels to check if the pixel lies inside any of the projected geometry and shade it based on the closest surface. Rasterization has been used by the graphics community ever since the inception of the field. Almost all Graphics Processing Units (GPUs) have special-purpose hardware built for rasterization. The simplicity of the algorithm and excellent hardware support have led to rasterization being used for real-time applications such as video games. However, achieving photorealistic rendering is hard with rasterization, and oftentimes, various tricks have to be used to obtain plausible results.

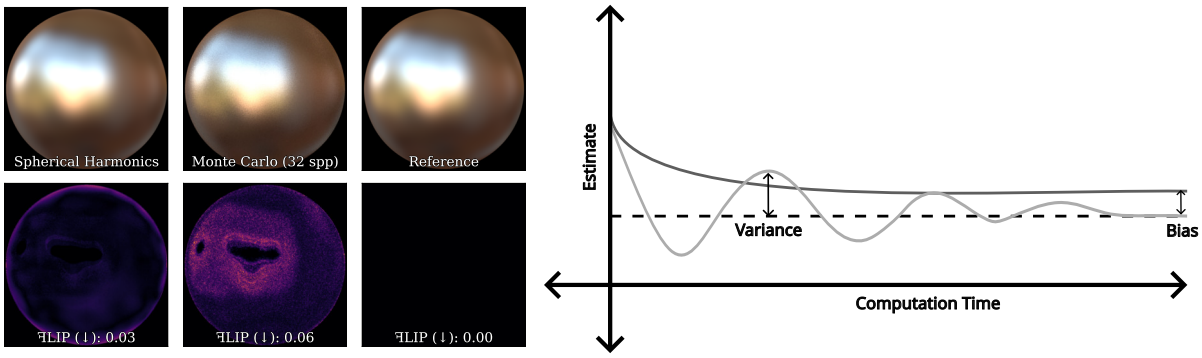


Figure 1.2: Illustration of bias and variance. Left: Render obtained with Spherical Harmonics (SH) is biased but has no variance. Meanwhile, the render obtained with Monte Carlo (MC) has variance, which manifests as noise. Increasing the number of samples removes the noise (Reference). Right: Convergence plot of the renderings. The dark grey line represents an analytic biased algorithm (SH), the light grey line represents a stochastic unbiased solution (MC), and the dotted line represents the reference.

1.1.2 Ray Tracing

The ray tracing algorithm can also be broken into two stages. In the first stage, rays passing through each pixel are generated. Then, for each pixel, the closest intersection point is found by iterating over all objects. The pixel is then shaded based on the interaction of light with the environment. Notice that this process is similar to rasterization, but the order of the iteration is flipped, i.e., the iteration is first made over pixels first and then over the objects in the scene. One can produce photorealistic images using ray tracing by tracking the trajectory of the light as it travels through the scene. However, ray tracing is much more computationally expensive and has been primarily used for offline rendering applications such as VFX or animation. Recent advances in hardware (NVIDIA RTX, AMD RDNA2), algorithms [7, 38, 47] and denoising [8, 52] methods has allowed ray tracing to be used in real-time applications such as video games. Several modern games use rasterization for the base rendering and ray tracing for certain effects such as reflections, refractions, soft shadows, and depth of field.

1.2 Bias and Variance

Bias and variance are two important properties associated with a rendering algorithm. Bias refers to a systematic error introduced by the algorithm by disregarding or approximating certain aspects of the light transport equation. For example, Linearly Transformed Cosines [24] (LTCs, see section 2.4.1) approximate the BSDF using a cosine and disregard visibility information. These approximations lead to a rendered image that is different from the ground truth.

Variance is generally associated with stochastic algorithms such as Monte Carlo Ray Tracing [60], Stochastic Photon Mapping [23] or Metropolis Light Transport [62]. These algorithms operate by randomly sampling paths and evaluating the LTE for these paths. This random sampling introduces variance

in the estimation of the LTE, which manifests as noise in the image. The variance reduces as the number of samples is increased and the method will converge to the exact solution in expectation.

Figure 1.2 illustrates bias and variance when estimating the LTE with two different algorithms. The rendering obtained with Spherical Harmonics is biased but has no variance. While the rendering obtained with Monte Carlo has variance which manifests as noise but is unbiased. The graph on the right shows the convergence properties of both of the methods.

Rasterization based techniques are generally biased and have zero variance, while ray tracing techniques are unbiased (i.e., they converge to the correct answer in expectation) but have finite or, in some cases infinite variance.

1.3 Our Contributions

Despite the recent advances in hardware support for ray tracing, complex scenes with intricate light transport still require high sample counts with Monte Carlo Ray Tracing to obtain noise-free renders. We identify two such cases and propose analytical and neural approaches to render them efficiently. This section offers a concise overview of our contributions. We begin by outlining the problem context for each contribution, providing an intuitive grasp of its complexity, and offering a brief overview of our proposed solutions.

1.3.1 Direct Lighting with Many Area Lights

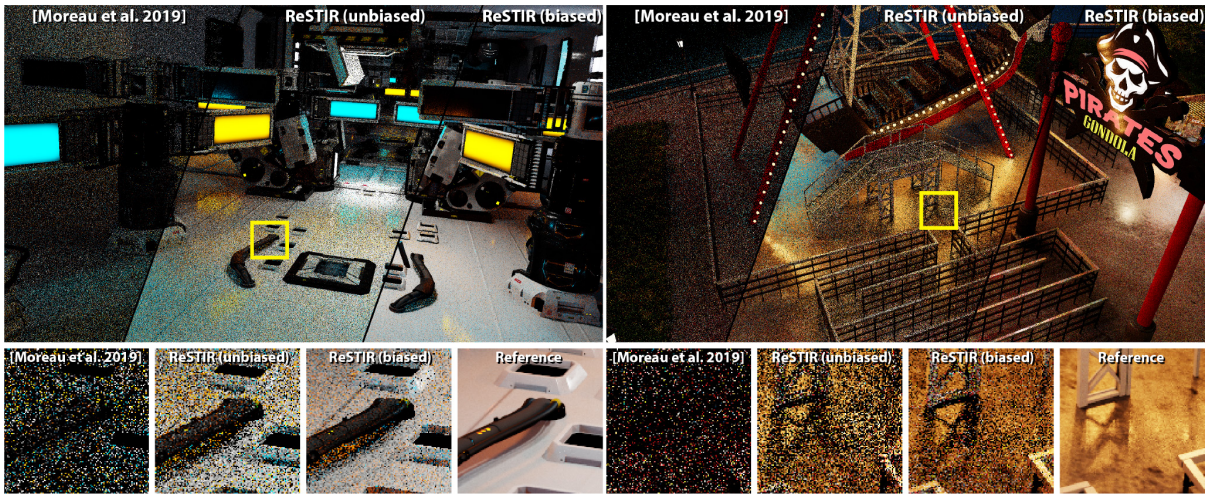


Figure 1.3: Two complex scenes ray traced with direct lighting from many dynamic lights. (Left) A still from the Zero Day video with 11,000 dynamic emissive triangles. (Right) A view of one ride in an Amusement Park scene containing 3.4 million dynamic emissive triangles. Both images show three methods [7, 41] running in equal time on a modern GPU. Image Courtesy: Bitterli et al. [7].

Direct lighting with many area lights is the accurate simulation of the illumination in a scene where numerous area lights contribute to the overall lighting conditions. The difficulty arises due to the intri-

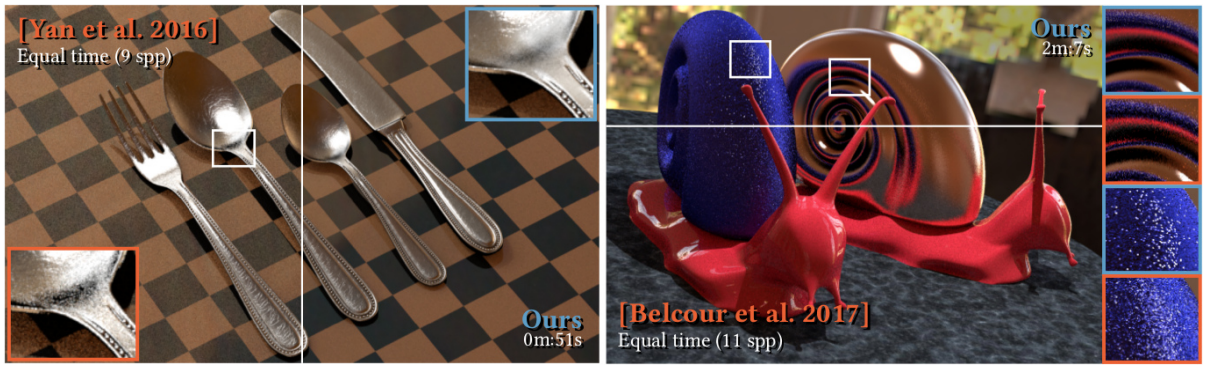


Figure 1.4: Two complex scenes with objects exhibiting glinty appearance. (Left) A render of the cutlery scene with direct illumination caused by a point light and an environment map. (Right) A still of the snail scene with global illumination effects. Image Courtesy: Gamboa et al. [20].

cate interactions and complex shadowing effects caused by the multiple light sources. A naive algorithm would be to iterate over all lights for each shading point and calculate its contribution. However, this quickly becomes expensive and doesn't scale well if the scene contains a large number of lights. For a given shading point, only a small set of lights close to it have a significant effect on it, and many lights can also be completely occluded and don't affect the shading point at all. Existing algorithms [7, 41] exploit this fact and propose algorithms that weigh each light's contributions based on the above factors. We extend upon [7] and use analytic methods to better estimate the effect of light on the shading point, leading to faster convergence of the algorithm.

1.3.2 Efficient Rendering of Glinty Appearance

Complex specular surfaces with rapidly changing microstructure exhibit glinty appearance. This appearance arises from the rapidly changing spatial characteristics of the underlying microstructure. Using Monte Carlo for rendering such surfaces is difficult because only a small portion of the surface is oriented to reflect the light in the correct direction. This integration is further compounded in the presence of large light sources as another integration over the light source needs to be performed. Existing algorithms use special data structures [4, 20, 72, 74] which help in efficiently querying the appropriately oriented microfacets. Additionally, they are also limited to a constant roughness for the entire surface. We present a new method that supports spatially varying roughness based on a neural histogram that computes per-pixel NDFs with arbitrary positions and sizes. Our representation is both memory and compute-efficient. Additionally, we compute full direct illumination integration analytically for all light directions with $\mathcal{O}(1)$ effort. Our method shows an improved quality compared to previous works by supporting smaller footprints while offering GPU-friendly computation and compact representation.

In the next chapter, we give a brief background of Physically Based Rendering and focus on some methods that are applicable to our contributions.

Chapter 2

Background

This chapter gives an overview of Physically Based Rendering (PBR). The goal of this chapter is to introduce some concepts that will be used in chapter 3 and chapter 4, establish the notation and terminology used, and provide an overview of various methods employed to achieve photorealistic renderings. We start by defining the Light Transport Equation (LTE) (section 2.1), which gives a mathematical formulation of how light travels and interacts in the world. Following this, we offer a brief survey of various methodologies employed to solve the LTE, with a specific emphasis on approaches relevant to our contributions. These methodologies are broadly categorized into three approaches, namely Monte Carlo-based methods (section 2.3), Analytic Methods (section 2.4), and Neural Approaches (section 2.5).

2.1 Physically Based Rendering

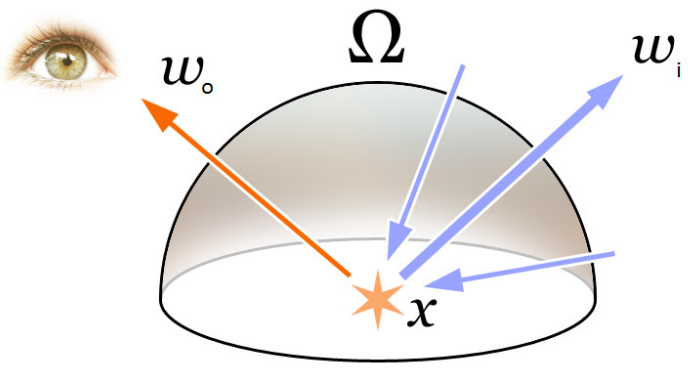


Figure 2.1: The LTE integrates the incoming light at a point modulated by the BSDF over the unit hemisphere around the shading point to obtain the radiance reflected in the direction ω_o. Image Courtesy: Wikipedia.

Physically Based Rendering (PBR) is a technique that models the physics of light interacting with the environment and reaching the camera sensor to create an image. At the heart of PBR lies the light transport equation (LTE) [31], which describes how the light travels and interacts in the scene. The LTE

is written as:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega_o}) = L_e(\boldsymbol{x}, \boldsymbol{\omega_o}) + \int_\Omega L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{\omega_i} \cdot \boldsymbol{n}) d\boldsymbol{\omega_i}, \tag{2.1}$$

where $L_o(\boldsymbol{x}, \boldsymbol{\omega_o})$ is the outgoing radiance from shading point $\boldsymbol{x}$ in direction $\boldsymbol{\omega_o}$, $L_e(\boldsymbol{x}, \boldsymbol{\omega_o})$ is the emitted radiance from the point, $L_i(\boldsymbol{x}, \boldsymbol{\omega_i})$ is the incident radiance on the surface from direction $\boldsymbol{\omega_i}$, $f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})$ is the Bi-directional Scattering Distribution Function (BSDF) which describes the surfaces appearance, and $\boldsymbol{n}$ is the surface normal. The LTE integrates the incoming light at a point modulated by the BSDF over the unit hemisphere around the shading point to obtain the radiance reflected in direction $\boldsymbol{\omega_o}$ (see fig. 2.1). This equation is a special case of the more general Radiative Transfer Equation (RTE) [9], which also considers the interaction of the light as it travels through the medium. In this thesis, we focus on the former and assume that the radiance doesn't change as the light travels through the medium.

2.2 Direct Lighting

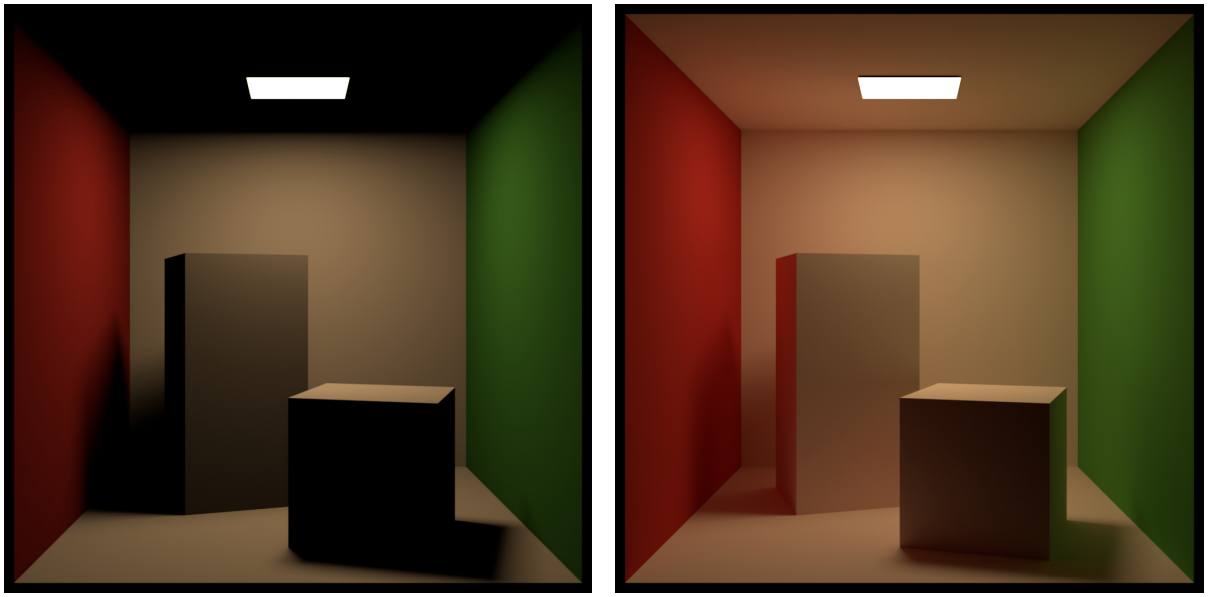


Figure 2.2: Comparison of the Cornell Box scene rendered with only direct illumination (left) and global illumination (right). Global illumination captures the effect of light bouncing on multiple surfaces before reaching the camera.

Direct lighting is a special case of the PBR, where we only consider paths with a single bounce. In direct lighting, we only consider radiance emitted by light sources and disregard the reflected (indirect) illumination from other surfaces of the scene. Figure 2.2 shows a comparison of the Cornell Box scene rendered with direct (left) and global illumination (right). Since we are interested in light directly coming from emitters in the scene, we can rewrite the LTE as a discrete sum over all light sources as

follows:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega_o}) = L_e(\boldsymbol{x}, \boldsymbol{\omega_o}) + \sum_{l \in \mathcal{L}} \int_{\Omega} L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) V(\boldsymbol{x}, \boldsymbol{\omega_i}, l)(\boldsymbol{\omega_i} \cdot \boldsymbol{n}) d\boldsymbol{\omega_i}, \tag{2.2}$$

where $\mathcal{L}$ is the list of all light sources in the scene, $V(\boldsymbol{x}, \boldsymbol{\omega_i}, l)$ is the binary visibility term, which indicates if the emitter is unoccluded in the given direction. In this thesis, we focus on two cases in which direct lighting is hard to solve and provide efficient algorithms for the same. In the rest of this chapter, we will briefly introduce some methods that can be used to solve eq. (2.2). Note that some of these methods are also applicable to solve the complete LTE (eq. (2.1)), but we focus on their application to solve the direct lighting equation.

2.3 Monte Carlo Integration

To render a scene with direct lighting, we must evaluate eq. (2.2) for each direction going into the camera. While the integral has a closed-form solution for simple light sources such as point or directional lights, a closed-form solution for most light types is unavailable. Hence, we use Monte Carlo integration to evaluate the direct lighting integral numerically. For a given function $f(x)$, the Monte Carlo estimator $\langle F \rangle$ of its integral $F = \int_{\mathcal{D}} f(x)dx$ over the domain $\mathcal{D}$ is formed by repeatedly evaluating the function at randomly sampled points in the domain and taking it's average. Mathematically, it is written as follows:

$$\langle F \rangle = \frac{|\mathcal{D}|}{N} \sum_{i=1}^{N} f(x_i), \quad \text{where } x_i \in \mathcal{D}. \tag{2.3}$$

Applying such an estimator for eq. (2.2) we get:

$$\langle L_o(\boldsymbol{x}, \boldsymbol{\omega_o}) \rangle = L_e(\boldsymbol{x}, \boldsymbol{\omega_o}) + \frac{2\pi}{N} \sum_{i=1}^{N} L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{\omega_i} \cdot \boldsymbol{n}). \tag{2.4}$$

Note that the domain of integration for direct lighting is the hemisphere around the shading point Ω and $|\Omega| = 2\pi$.

Monte Carlo integration computes an accurate (unbiased) solution to the LTE by randomly determining the direction for the next segment of a ray. However, this random sampling introduces variability, manifesting as noise in the rendered image. Increasing the number of samples (shooting more rays per pixel) helps in better estimating the integral and reducing the noise in the image. The variance of the Monte Carlo estimator is heavily influenced by the scene's complexity, with more complex light transport requiring a higher number of samples to achieve acceptable render quality, making rendering computationally expensive.

Various variance reduction techniques can be employed to mitigate the variance introduced by Monte Carlo. One such technique we delve into in the following sections is Importance Sampling and its varia-

tion: Multiple Importance Sampling [61]. For a comprehensive exploration of other variance reduction techniques and their analysis, readers are encouraged to refer to Chapter 2 of Veach's thesis [60].

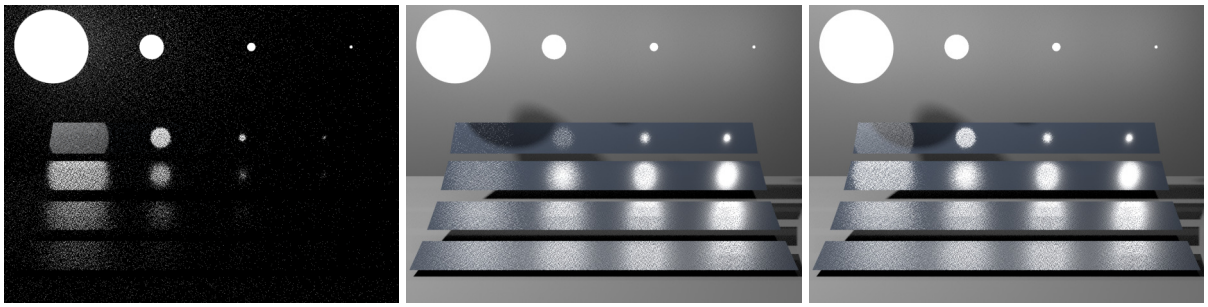
2.3.1 Importance Sampling

Importance sampling is a method to reduce the variance of a Monte Carlo estimator by sampling from a distribution that has a shape similar to that of the integrand. The closer $p(x)$ is to the integrand $f(x)$, the better the variance reduction achieved. An importance sampled Monte Carlo estimator for function $f(x)$ with a probability distribution function (PDF) $p(x)$ is given by:

$$\langle F \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}. \tag{2.5}$$

In the case of eq. (2.4), one can sample proportional to either the BSDF $f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})$ (BSDF sampling), the incoming radiance $L_i(\boldsymbol{x}, \boldsymbol{\omega_i})$ (emitter sampling) or the $(\boldsymbol{\omega_i} \cdot \boldsymbol{n})$ term (Cosine Weighted sampling). Usually based upon the properties of the shading point and the scene, one of the above strategies performs better than the others (see fig. 2.3).

2.3.2 Multiple Importance Sampling



(a) BSDF Sampling (b) Light Sampling (c) MIS

Figure 2.3: Comparison of various importance sampling strategies. BSDF sampling performs well if the light source is source is big or the surface is specular. Light sampling works well if the light is smaller or the surface is diffuse. MIS combines both of the sampling methods. Image Courtesy: PBRTv3 [50].

Combining the above importance sampling strategies is not straightforward. Simply averaging the results obtained from each sampling strategy with equal weights doesn't work well since Monte Carlo noise is additive. If a single strategy has a high variance, it will be added to the final estimator. Instead, we need to combine all the strategies based on which sampling strategy is better for the given shading point. This combination of multiple sampling strategies to get an optimal estimator is called Multiple Importance Sampling (MIS) [61]. MIS works by defining a set of weights $w_i(x)$ for each of the sampling

strategies to be combined and uses them to build an estimator as follows:

$$\langle F \rangle = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=1}^{M} w_j(x_{ij}) \frac{f(x_{ij})}{p_j(x_{ij})} \right), \tag{2.6}$$

where M is the number of sampling strategies used. For the estimator to be unbiased, the following two conditions have to be met by the MIS weights:

- The sum of weights should be 1 if the integrand is non-zero.
- The MIS weight for a strategy should be zero if the sampling probability of that strategy is zero.

There are many strategies to obtain the above weights. The Balance Heuristic [61] is a provably good way of calculating weights for MIS to achieve close to optimal variance reduction. When using the balance heuristic, the weight $w_i(\boldsymbol{x}, \boldsymbol{\omega_i})$ for a sampling strategy is given as:

$$w_i(x) = \frac{p_i(x)}{\sum_{j=1}^{M} p_j(x)} \tag{2.7}$$

It is trivial to see that the above MIS strategy satisfies both conditions required for an unbiased estimator. Kondapaneni et al. [34] propose an extension to the Balance Heuristic by also considering negative weights. We refer readers to Chapter 9 of Veach's thesis [60] for further in-depth analysis and proof of MIS.

2.4 Analytic Solutions

While the Monte Carlo methods are general and unbiased (if done correctly), it is computationally expensive to obtain noise-free renderings. Denoising algorithms help reduce this noise but can be expensive and also add uncontrolled bias. Analytical solutions, on the other hand, produce noise-free renderings and are much cheaper to evaluate. However, to come up with analytic solutions, various simplifying assumptions about the LTE are made (for example, disregarding visibility), which makes these methods less general. Moreover, these methods oftentimes introduce some amount of known bias. Nonetheless, the ability to produce noise-free renderings with considerably lesser computational cost has made analytic solutions popular in real-time applications. In this section, we describe two methods that will be used in the further chapters of the thesis. We also describe the source of bias for both cases and outline possible solutions to avoid them.

2.4.1 Linearly Transformed Cosines

Area lights are shapes in the scene that emit light from their surfaces with some directional distribution. Usually, area lights are composed of polygons or analytical shapes such as circles, spheres, cylinders, etc. In general, computing the LTE in the presence of area lights requires one to integrate

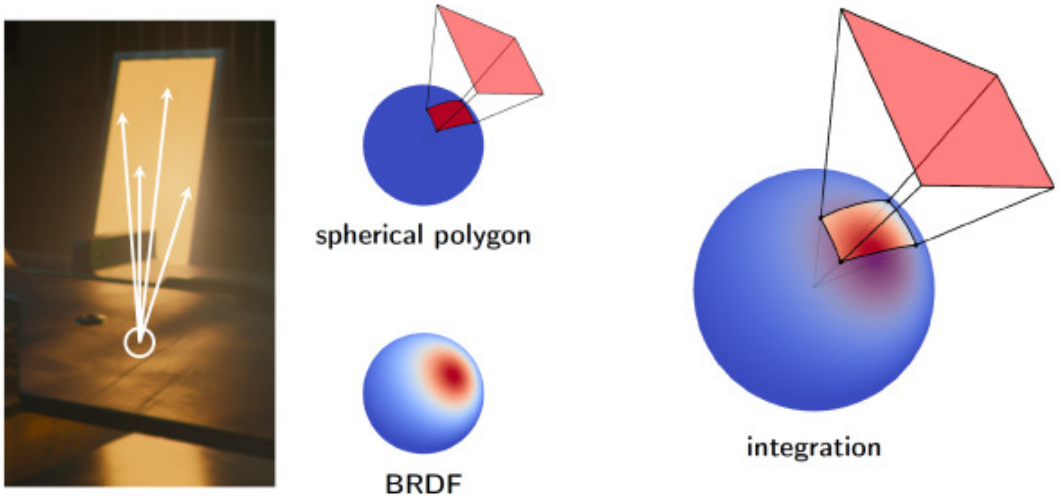


Figure 2.4: Shading in the presence of an area light requires one to integrate the Radiance-BSDF product over the surface of the light. A closed-form solution is often unavailable, and Monte Carlo integration is used. Image Courtesy: Heitz et al. [24].

over the surface of the area light (see fig. 2.4), and a closed-form analytical solution is not available. To calculate the shading of a point under the influence of an area light we need to solve the following integral:

$$L_o(A, \boldsymbol{x}, \boldsymbol{\omega_o}) = \int_A L_i(\boldsymbol{y} \rightarrow \boldsymbol{x}) f_r(\boldsymbol{x}, \boldsymbol{y} \rightarrow \boldsymbol{x}, \boldsymbol{\omega_o}) V(\boldsymbol{y} \rightarrow \boldsymbol{x}) G(\boldsymbol{x} \leftrightarrow \boldsymbol{y}) d\boldsymbol{y}, \tag{2.8}$$

where A is the surface of the area light, $\boldsymbol{x}$ is the shading point, $\boldsymbol{y}$ is a point on the light, $G(\boldsymbol{x} \leftrightarrow \boldsymbol{y})$ is the geometric term which includes the Jacobian for change of measure from differential solid angle $d\boldsymbol{\omega_i}$ to a differential area on the area light $d\boldsymbol{y}$ and the cosine foreshortening term.

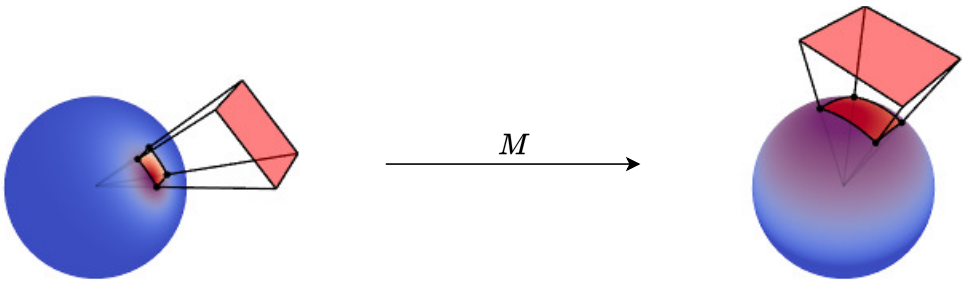


Figure 2.5: Linearly Transformed Cosines involve applying a transform M to the BSDF and light vertices to bring them into a frame where an analytical solution is available. Image Courtesy: Heitz et al. [24].

Linearly Transformed Cosines (LTC) [24] is a method to compute a plausible analytic approximation to the integral in eq. (2.8) in the presence of polygonal area lights. This is achieved with a matrix that transforms the light and BSDF from the true BSDF to a cosine distribution (see fig. 2.5) where analytic integration is possible. Concretely, given an area light A with vertices $(\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)$, we apply

a linear transform M, which transforms the BSDF into a cosine. These transformation matrices are precomputed for a set of view directions $\boldsymbol{\omega_i}$ and roughness values α and interpolated at runtime. Given the set of transformed polygonal vertices $\boldsymbol{p}' = M\boldsymbol{p}$, the approximate total unshadowed illumination $E(A)$ from the area light is given as:

$$E(A) = \frac{1}{2\pi} \sum_{i=1}^{n} \cos^{-1}\left(\langle \boldsymbol{p}'_i, \boldsymbol{p}'_j \rangle\right) \left\langle \frac{\boldsymbol{p}'_i \times \boldsymbol{p}'_j}{|\boldsymbol{p}'_i \times \boldsymbol{p}'_j|}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\rangle \tag{2.9}$$

Source of bias: The main source of bias in LTCs is due to the assumption that the emitter is completely unoccluded. This assumption removes all shadows cast by the emitter. Several solutions, both stochastic [25] and analytic [36] have been proposed to add back visibility information in LTCs.

Another source of bias comes from the fact that a linearly transformed cosine doesn't fit the BSDF exactly hence, the analytical integral would not exactly be equal to the actual one. In chapter 3, we propose to use LTCs for sampling instead of using them to evaluate the final shading. Doing so allows us to achieve a low variance Monte Carlo estimator, completely avoiding the issue of bias in LTCs.

2.4.2 Spherical Harmonics

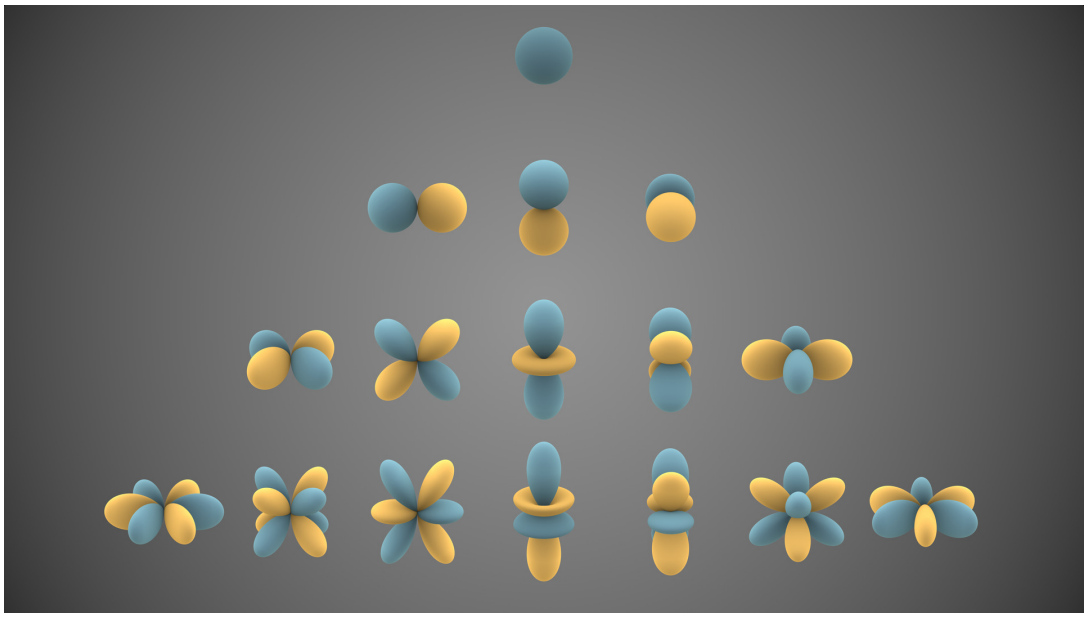


Figure 2.6: Visualization of the first four Spherical Harmonic bands. The radius of the sphere is given by the SH value in that direction. Blue denotes positive values, and yellow denotes negative values. Image Courtesy: Wikipedia.

Spherical Harmonics (SH) are orthonormal basis functions defined on the sphere that allow us to express directionally varying functions as a weighted sum. The real basis functions Y for band $l \in$

$\{0, \ldots, l_{\text{max}}\}$ and index $m \in \{-l_{\text{max}}, \ldots, l_{\text{max}}\}$ are defined as:

$$Y_m^l(\boldsymbol{\omega}) = \begin{cases} \sqrt{2}\, K_l^m\, P_l^{-m}(\cos\theta)\, \sin(-m\phi) & m < 0 \\ K_l^m\, P_l^m(\cos\theta) & m = 0 \\ \sqrt{2}\, K_l^m\, P_l^m(\cos\theta)\, \cos(m\phi) & m > 0 \end{cases} \tag{2.10}$$

where P are the associated Legendre Polynomials and K are the normalization factors.

Given a function f, computing its SH coefficients requires:

$$c_l^m = \int_\Omega f(\boldsymbol{\omega}) Y_l^m(\boldsymbol{\omega}) d\boldsymbol{\omega} \tag{2.11}$$

and with sufficient SH order (i.e. l_{max}), function f can be perfectly represented by a vector f_{SH} of all coefficients c_l^m. Spherical Harmonics have several useful properties that make them suitable for solving the LTE. One particularly useful property of SH, which we use in chapter 4 is the Double Product Integral (DPI). Given a function $f(\boldsymbol{\omega})$ and $g(\boldsymbol{\omega})$ that are represented using a set of SH coefficients f_{SH} and g_{SH} correspondingly, the integral of the product of both functions is given by the dot product of the SH coefficient vectors, i.e.:

$$\int_\Omega f(\boldsymbol{\omega}) g(\boldsymbol{\omega}) d\boldsymbol{\omega} = (f_{\text{SH}} \cdot g_{\text{SH}}). \tag{2.12}$$

If we represent the BSDF and the lighting in SH, then we can use the DPI property to calculate the unshadowed LTE (without the visibility term) at the shading point. Specifically we set $f = L_i(\boldsymbol{x}, \boldsymbol{\omega_i})$ and $g = f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})$. In chapter 4, we use this double product integral property to compute the shading due to the complex lighting and a complex BSDF efficiently.

Source of bias: Similar to LTCs, solving the LTE using DPI will discard visibility information. One can use the TPI property [45] to incorporate visibility information analytically; however, it is much more expensive to compute. Like LTCs, we can apply Ratio Estimators [25] to stochastically add visibility information.

In practice, we usually need to limit the SH order to some l_{max}. Doing so removes the high-frequency content of the signal being represented, also removing it from the final integral obtained from DPI. This bandlimiting of the signal causes a systematic bias to be introduced into the system. However, in most cases, choosing a sufficiently high cutoff frequency ensures this bias is negligible.

2.5 Neural Approaches

Neural networks are universal function approximators and have been increasingly popular for their high representation power in recent years. Recent works have started using Neural Networks in various parts of the rendering system to varying degrees. While neural methods are general, one has to carefully tune the training procedure, architecture, and inference method to balance computational cost, quality,

and error due to the network. In this section, we summarize some works which are relevant to our contributions.

2.5.1 Learning Radiance

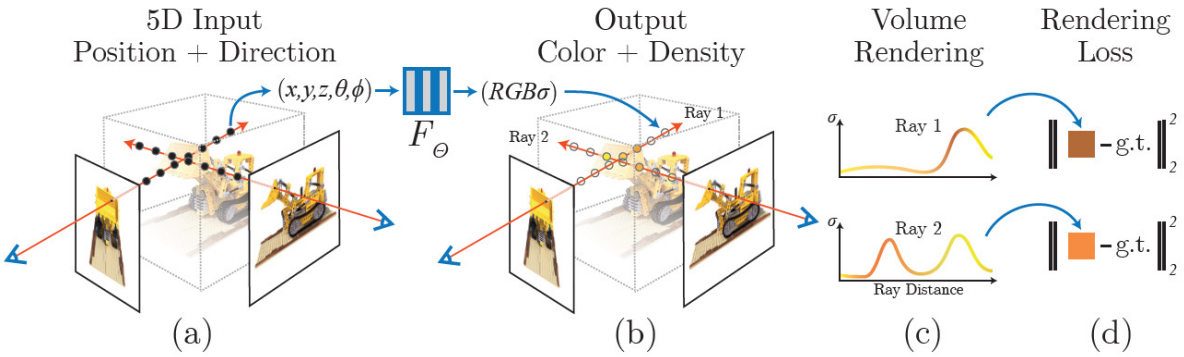


Figure 2.7: NeRF trains a neural network that learns the radiance distribution of the scene. The neural network takes the position $\boldsymbol{x}$ and view direction $\boldsymbol{\omega_o}$ of a point as input and outputs the radiance $L_o(\boldsymbol{x}, \boldsymbol{\omega_o})$, which can then be used to perform volumetric rendering for novel view synthesis. Image Courtesy: Mildenhall et al. [40].

Neural radiance fields (NeRFs) [69] have become popular neural rendering methods in recent years. NeRFs train a neural network that learns the radiance distribution of the scene. The neural network takes the position $\boldsymbol{x}$ and view direction $\boldsymbol{\omega_o}$ of a point as input and outputs the radiance $L_o(\boldsymbol{x}, \boldsymbol{\omega_o})$, which can then be used to perform volumetric rendering to produce novel views (see fig. 2.7). The weights of the network are optimized by backpropagating the gradients of the L_2 loss between the produced rendering and ground truth images. While NeRF is more applicable to the vision domain, the ideas it presents have also been applied to a traditional rendering setup.

Neural Radiance Caching (NRC) [43] applies the idea of using a neural network to predict the radiance to path tracing. It does so by training the neural network on the fly while rendering on a subset of pixels for which full path tracing is performed. The neural network then predicts the radiance for the other pixels. Further work along these lines has been made to limit the bias introduced by the neural network and efficiently render multiple scattering in hair [35]. Finally, work has also been done to improve convergence rates of Monte Carlo methods in the presence of complex lighting structures [77] by learning the emitted radiance distribution in the scene.

2.5.2 Learning BSDF

Another use of Neural Networks in the rendering loop has been to learn the BSDF distribution of the objects. Oftentimes, BSDFs used in production are quite complex and become the biggest bottleneck in the rendering, limiting their usage in real-time applications. Even in offline rendering, Manuka [19], the in-house renderer used by WetaFX, was architected while keeping this bottleneck in mind. Much

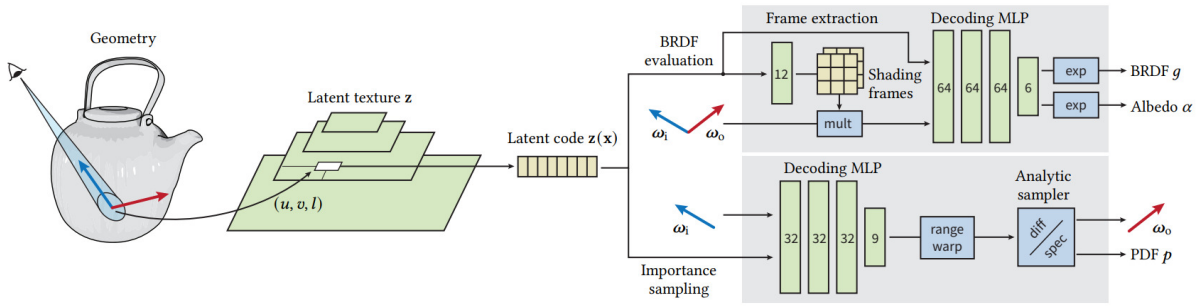


Figure 2.8: Neural networks being used for fitting to a BSDF. Often times, complex BSDFs are a bottleneck limiting their usage in real-time applications. Using small neural networks that are fast to infer allows such complex BSDFs to be used for real-time applications. Image Courtesy: Zeltner et al. [76].

research has gone into representing such BSDFs using neural networks. The idea is to take as input the view and outgoing direction pair $(\boldsymbol{\omega_o}, \boldsymbol{\omega_i})$ along with some optional auxiliary spatial information at point $\boldsymbol{x}$ to output the BSDF value $f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})$. Works in this domain include those by Kuznetsov et al. [37], Fan et al. [18], Zeltner et al. [76], and Xu et al. [70]. In chapter 4, we make similar use of neural networks to learn the multi-scale BSDF for a given position and scale. However, instead of taking $\boldsymbol{\omega_i}, \boldsymbol{\omega_o}$ as input, we predict the bin counts on the hemisphere, which represent a discrete binned BSDF.

This chapter provided an overview of Physically Based Rendering (PBR) and introduced key concepts such as the Light Transport Equation (LTE). We explored three main approaches to solving the LTE: Monte Carlo-based methods, analytical methods, and neural approaches. In the next two chapters, we will present our solutions to two cases with complex light transport.

Chapter 3

Direct Lighting with Many Area Lights

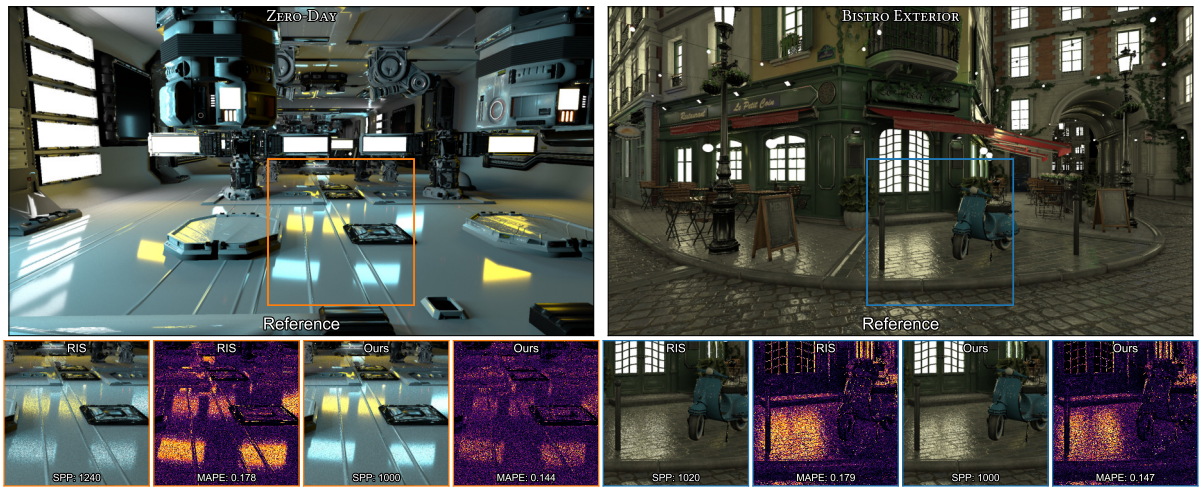


Figure 3.1: We propose to combine Resampled Importance Sampling (RIS) from ReSTIR [7] and Projected Solid Angle Sampling [49] for rendering scenes with a large number of area light sources. This figure shows an equal-time comparison of our method with RIS on the Zero-Day (left, 10K lights) and Bistro Exterior (right, 30K lights) scenes. Our method traces a similar number of samples as RIS and achieves a lower Mean Absolute Percentage Error (MAPE) w.r.t a 1M spp reference.

3.1 Introduction

In this chapter, we present an efficient method to render scenes with direct lighting in the presence of many area lights. Direct lighting from many area light sources is challenging due to variance from both choosing an important light and then a point on it. Resampled Importance Sampling (RIS) [57] achieves low variance in such situations, however, it is limited to simple sampling strategies for its candidates. Specifically for area lights, we can improve the convergence of RIS by incorporating a better sampling strategy: Projected Solid Angle Sampling (ProjLTC) [49]. Naively combining RIS and ProjLTC improves equal sample convergence, however achieves little to no gain in equal time. We identify the core issue for the high run-times and reformulate RIS for better integration with ProjLTC. Our method

achieves better convergence and results in both equal sample and equal time. We evaluate our method on challenging scenes with varying numbers of area light sources and compare it to uniform sampling, RIS, and ProjLTC. In all cases, our method seldom performs worse than RIS and often performs better.

Motivation Direct lighting from the environment and area light sources is at the core of Monte Carlo path tracing. Monte Carlo methods repeatedly sample light sources to evaluate the direct lighting integral. In the case of environment lights, sampling is in accordance with the environment map's PDF. However, in the case of area lights, sampling involves first choosing one from a list of all area lights and then choosing a point on it, introducing two sources of variance. Thus, direct lighting especially in the presence of many area light sources is an active research area.

Given a shading point $\boldsymbol{x}$ viewed from direction $\boldsymbol{\omega_o}$ and a set of area lights $A = \{A_1, A_2, ...A_j\}$, the radiance $L_o(\boldsymbol{x}, \boldsymbol{\omega_o})$ at $\boldsymbol{x}$ is given as:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega_o}) = \sum_{A_j \in A} L_o(A_j, \boldsymbol{x}, \boldsymbol{\omega_o}), \tag{3.1}$$

where $L_o(A, \boldsymbol{x}, \boldsymbol{\omega_o})$ is the total incoming radiance from the area light A reflected in direction $\boldsymbol{\omega_o}$ (see eq. (2.8)). Monte Carlo is used to estimate eq. (3.1) with a discrete probability $p(A_j|A)$ to sample a light and a continuous probability density $p(\boldsymbol{y}_i|A_j)$ to sample a point $\boldsymbol{y}_i$ on this light as:

$$\langle L_o(\boldsymbol{x}, \boldsymbol{\omega_o}) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L_i(\boldsymbol{y}_i \to \boldsymbol{x}) f_r(\boldsymbol{x}, \boldsymbol{y}_i \to \boldsymbol{x}, \boldsymbol{\omega_o}) V(\boldsymbol{y}_i \to \boldsymbol{x}) G(\boldsymbol{x} \leftrightarrow \boldsymbol{y}_i)}{p(A_j, \boldsymbol{y}_i)}, \tag{3.2}$$

where $p(A_j, y_i) = p(A_j|A)p(\boldsymbol{y}_i|A_j)$. The variance of this estimator is low if this joint PDF and, by extension, the individual conditional PDFs closely approximates the integrand.

In this work, we explore a clear gap: we combine RIS, the state-of-the-art direct lighting framework, with state-of-the-art point sampling of area lights, i.e., ProjLTC, for rendering scenes with a large number of area lights. A naive combination is to directly use ProjLTC for $p(\boldsymbol{y}_i|A_j)$ in RIS. We show that this results in poor run-time as RIS needs to sample this underlying conditional PDF multiple times, which is expensive to do with ProjLTC. We propose to improve the run-time by reformulating RIS to operate on $p(A_j|A)$ using LTCs and then applying ProjLTC once to sample from the selected light. We evaluate our and previous methods on three scenes with varying numbers and sizes of light sources. Compared to RIS-ProjLTC (the naive version), uniform sampling, RIS, and ProjLTC, our method achieves lower error in equal time.

Figure 3.1 shows equal time comparisons of our method with RIS on two scenes. In both scenes, our method achieves a lower quantitative and perceptual error.

3.2 Related Work

Sampling lights: There are various methods for efficiently sampling a single light from a list of lights on the GPU. The most straightforward approach is uniform sampling, where lights are selected with equal probability [50]. However, a more effective strategy involves sampling lights in proportion to their power, as suggested by Shirley et al. [54]. To speed up the sampling process, an alias table can be employed [63]. Although sampling proportional to power provides good results, it overlooks the distance of the light source from the shading point. Lights far away may have high power but might not significantly contribute to the illumination. LightBVH [17] builds a BVH on the lights in the scene and stochastically traverses it to sample proportional to the effective radiance received at the shading point. A further improvement is presented by Moreau et al. [41], who divide the BVH into multiple levels. This modification enhances the efficiency of rebuilding the LightBVH, making it suitable for real-time applications where rapid updates are required. Lightcuts proposed by Walter et al. [64] build a binary tree where the leaf nodes are the light sources and interior nodes are representative lights approximating the contribution of all their children nodes. The final illumination from the lights is determined by dynamically cutting the tree based on the shading point and then utilizing the leaf nodes of the pruned tree to calculate the overall illumination. Stochastic Lightcuts [75] extends this idea and uses the binary tree only for sampling, thereby removing the bias. An extension of the method [39] to make it suitable for GPUs has been proposed.

Sampling points: Similarly, various methods exist for sampling a point $\boldsymbol{y}_i$ on the chosen area light. The simplest method samples $\boldsymbol{y}_i$ according to the area of the chosen light [50]. Methods which sample proportional to the solid angle [3, 21, 58] of the area light have also been proposed. These methods aim towards low sampling variance from $p(\boldsymbol{y}_i|A_j)$. A recent method [49] achieves a stable implementation with near zero variance using projected solid angle sampling and LTCs (section 2.4.1). This method achieves the state-of-the-art in sampling points on area lights, albeit at the cost of more computation. We refer to this method as *ProjLTC*. Instead of relying on data structures for light sampling and attempting to approximate the PDF proportional to LTE, Spatiotemporal Reservoir Sampling (ReSTIR) [7] proposes to use simple and efficient strategies for both $p(A_j|A)$ and $p(\boldsymbol{y}_i|A_j)$. The resulting PDF $p(A_j, \boldsymbol{y}_i)$ is then transformed to the target PDF which is proportional to the LTE using Resampled Importance Sampling (RIS) [57]. RIS with spatiotemporal reuse achieves state-of-the-art results on scenes with a large number of light sources.

3.3 Preliminaries

We start with a review of Resampled Importance Sampling (RIS), which is at the core of ReSTIR [7]. We then review the ProjLTC method [49] which uses LTCs (see section 2.4.1) with projected solid angle sampling to sample points on area lights.

3.3.1 Resampled Importance Sampling

Resampled Importance Sampling (RIS) [57] is an approach to obtain an importance sampled Monte Carlo estimator. Instead of combining multiple different sampling strategies like MIS (see section 2.3.2), RIS starts with a PDF $p(x)$ which is easy to sample from, and transforms it into a target PDF $\hat{p}(x)$ which is hard to sample from. It does so by generating an initial set of M candidate samples and choosing an index $z \in \{1, \dots M\}$ with probability:

$$p(z|x) = \frac{w(x_z)}{\sum_{i=1}^{M} w(x_i)} \quad \text{where} \quad w(x) = \frac{\hat{p}(x)}{p(x)}. \tag{3.3}$$

The chosen sample from RIS, a correction factor dependent on $w(x)$ and the target PDF $\hat{p}(x)$ can then be used for unbiased MC estimation as follows:

$$\langle F \rangle = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{f(x_z)}{\hat{p}(x_z)} \cdot \left(\frac{1}{M} \sum_{j=1}^{M} w(x_{ij}) \right) \right) \tag{3.4}$$

In the context of solving the Direct Lighting, the target distribution is set to be the unshadowed illumination i.e., $\hat{p}(\boldsymbol{x}, \boldsymbol{\omega_i}) = L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})$ at the shading point. For the candidate sampling PDF $p(\boldsymbol{x}, \boldsymbol{\omega_i})$, one can choose any function. However, better results are achieved if $p(\boldsymbol{x}, \boldsymbol{\omega_i})$ closely matches the target PDF. RIS has been used as a core component of recent algorithms which reuse samples spatially and temporally to achieve real-time direct [7] and global [38, 47] illumination. We extend upon [7] and reformulate RIS (see chapter 3) in the case of direct lighting with many area lights to achieve better results.

We refer to and use the version of RIS as described by ReSTIR. In the context of direct lighting with area lights, RIS sets $\hat{p}(\boldsymbol{x}, \boldsymbol{\omega_i}) = L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})$ and $p(x) = p(A_j|A)p(\boldsymbol{y}_i|A_j) = p(A_j|x) \cdot 1/|A_j|$, where $|A_j|$ denotes the area of the light source A_j. We set the number of candidates to $M = 32$. The choice of $p(A_j|A)$ is orthogonal to our contribution and method: we focus on using the best sampling strategy for $p(\boldsymbol{y}_i|A_j)$ in RIS.

3.3.2 Projected Solid Angle Sampling

Peters [49] described a method that can sample the projected solid angle of an area light with PDF:

$$p(\boldsymbol{y}_i|A_j) = \frac{G(\boldsymbol{y}_i \leftrightarrow \boldsymbol{x})}{S(A_j)}, \tag{3.5}$$

where $S(.)$ gives the projected solid angle. Given a light source, their method applies LTC, transforming the BSDF to the cosine space where projected solid angle sampling can be used. Their method achieves close to zero variance for diffuse BSDFs and uses an MIS strategy for low variance in the case of glossy BSDFs.

3.4 Method

We aim to improve the equal time and equal sample Monte Carlo convergence for direct lighting with a large number of area light sources. To that end, we start with a simple combination of RIS and ProjLTC (section 3.4.1). We observe that this leads to excessive run-time and propose to improve it by reformulating RIS for light sampling only and using ProjLTC for sampling a point on the chosen light and final shading (section 3.4.2).

3.4.1 Combining RIS and ProjLTC

In RIS for area light shading, the candidate PDF $p(\boldsymbol{y}_i|A_j)$ is uniform over the area of the chosen light. A naive way to incorporate projected solid angle sampling is to use the sampling and PDF of ProjLTC instead. The problem with this approach can be understood from section 3.3.1: sampling has to be done M times, and ultimately only one sample is chosen. The majority of the sampling effort is thus wasted and does not affect the final answer Furthermore, given that sampling using ProjLTC is more expensive, the efficiency decreases. This is precisely why the original RIS uses other simple and efficient strategies like uniform area sampling for $p(\boldsymbol{y}_i|A_j)$.

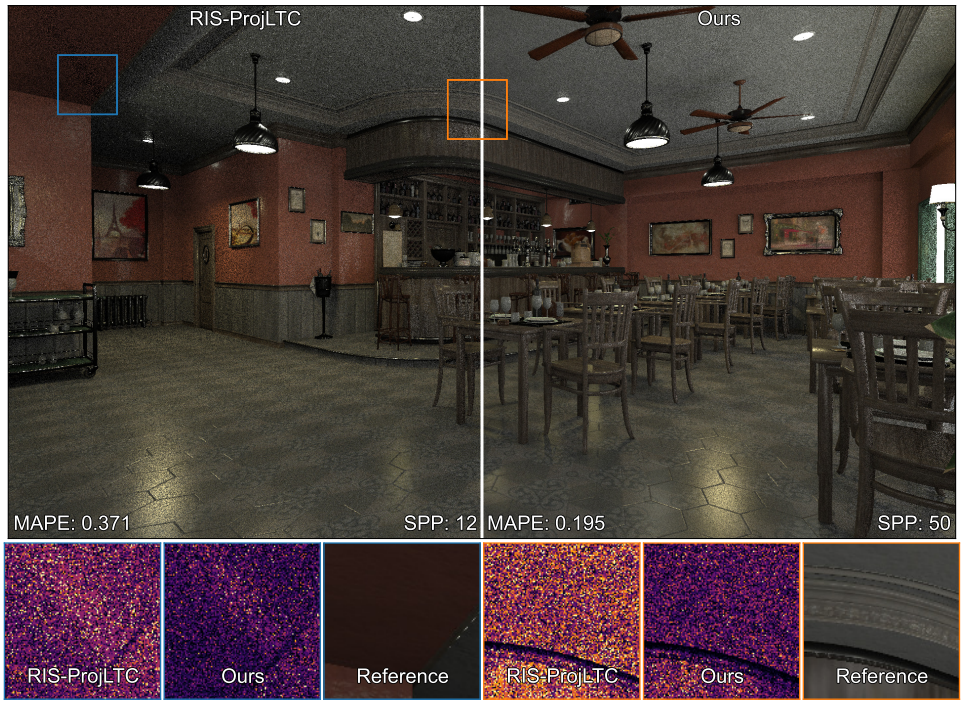


Figure 3.2: This figure shows an equal time (200 ms) comparison of our method (section 3.4.2) with RIS-ProjLTC (section 3.4.1) on the Bistro Interior scene with 2K light sources. The insets show difference images with a 1M spp ray-traced reference. Our method achieves lower error and traces more samples in the same time.

3.4.2 Reformulating RIS

The core issue is that RIS works on *point samples* on area lights. This implies that the candidate PDF $p(x)$ in eq. (3.3) should be over all points on all area lights. Therefore, to combine it with ProjLTC, we have no choice but to use ProjLTC for each candidate sample since ProjLTC itself is a point sampling strategy.

We can however reformulate RIS to sample a single light instead of a point on it by setting the target and candidate PDFs to:

$$\hat{p}(x) = E(A_j) \quad \text{and} \quad p(x) = p(A_j|x), \tag{3.6}$$

where $E(A_j)$ is the analytic solution of LTCs (eq. (2.9)). Our reformulated version of RIS thus chooses a point on an area light as follows: (1) sample lights from $p(x)$, (2) evaluate $\hat{p}(x)$ on the candidate lights with LTCs, (3) assign weights and sample a single light according to eq. (3.3) and (4) apply ProjLTC to this sampled light. Since we use LTCs strictly for sampling and not final shading, our method is unbiased.

Figure 3.2 shows an equal time (200 ms) comparison of the naive combination of RIS and ProjLTC (RIS-ProjLTC) with our method combining reformulated RIS with ProjLTC. Our method achieves lower error and traces more samples than RIS-ProjLTC. The difference image insets also shows a lower perceptual error for our method.

3.5 Implementation

We implement our reformulated RIS, projected solid angle sampling, and LTCs in an open-source Vulkan renderer [49]. We use the Frostbite BSDF as our material model and use LTCs that are optimized for it. Our RIS implementation uses Weighted Reservoir Sampling (WRS) [68] to stream samples, similar to ReSTIR.

Alg. 1 describes the pseudocode of our method to compute direct lighting given a shading point $\boldsymbol{x}$ (line 14). It starts with sampling of an area light with our reformulated RIS with $M = 32$ (line 15). The next step is to apply projected solid angle sampling (ProjLTC) to the chosen light (line 16) and then finally compute the radiance estimate (line 17). Note that L here is a multiplication of the BRDF, incoming light, visibility, and geometry terms (line 2).

The `ris_sample_light` function (line 5) describes our reformulated RIS that samples a light using LTCs. We use the definition of a `Reservoir` from ReSTIR [7] (lines 6, 7). Each of the M light candidates are sampled using $p(A_j|A)$ and the target PDF $\hat{p}$ is set to the respective candidate's LTC evaluation (eq. (3.6), lines 9, 10). In our implementation we choose the candidates uniformly at random, i.e. $p(A_j|A) = 1/|A|$. Better sampling strategies for choosing the candidate lights may further improve convergence rates. We use the LTC approximation of [26] for efficiency. Finally, line 11 computes the weights (eq. (3.3)) and updates the reservoir.

ALGORITHM 1: RIS with Projected Solid Angle Sampling

1 **Def** *proj_ltc(x, A_j)*:
  `/* Projected Solid Angle Sampling [49] */`
2 L = projected_solid_angle_sample(x, A_j) `//` $L = f_r \cdot V \cdot L_i \cdot G$
3 $\tilde{p}$ = projected_solid_angle_pdf(x, A_j)
4 **return** L, $\tilde{p}$
5 **Def** *ris_sample_light(x, M)*:
  `/* Refer to [7] for the definition of Reservoir */`
6 Reservoir r
7 $r.M = M$
8 **for** $i \leftarrow 1$ *to* M **do**
9 generate $A_j \sim p(A_j|x)$
10 $\hat{p} = E(A_j)$ `// Equation (3.6)`
11 r.update(A_j, $\frac{\hat{p}}{p(A_j|x)}$) `// Calculate weight Eq. Equation (3.3)`
12 r.W = $\frac{1}{E(r.y)} \left(\frac{1}{r.M} \cdot r.w_{sum}\right)$ `//` $E(.)$ `is the target PDF` $\hat{p}$
13 **return** r
14 **Def** *direct_lighting(x)*:
15 Reservoir r = ris_sample_light(x, 32) `//` $M = 32$
16 L, $\tilde{p}$ = proj_ltc(x, $r.y$) `//` $r.y$ `stores the chosen light` A_j
17 **return** $\frac{L}{\tilde{p}} \cdot r.W$

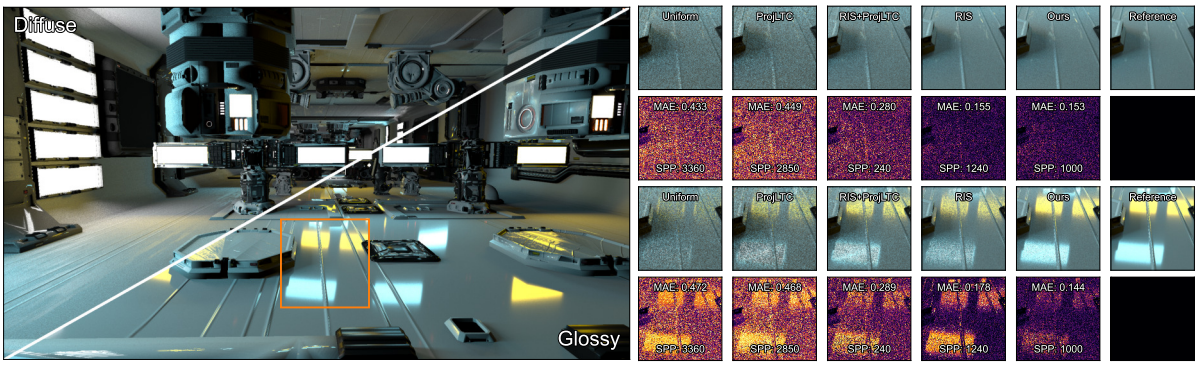
The `proj_ltc` method is straightforward: it takes shading point $\boldsymbol{x}$ and the chosen light A_j and samples it using projected solid angle sampling, potentially using MIS [49] (line 2). This function also returns the corresponding sampling PDF (line 3).

3.6 Results and Analysis

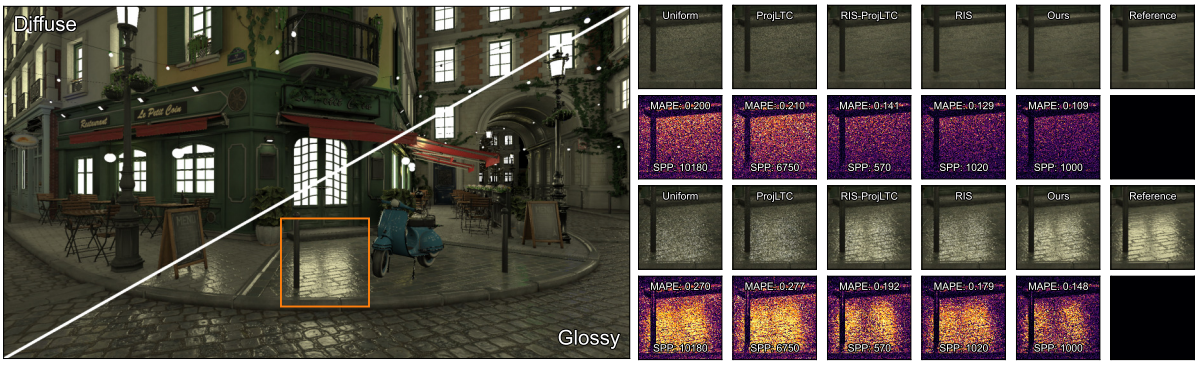
We qualitatively and quantitatively compare our method against the following: (1) **Uniform**: Uniform light and uniform area sampling, (2) **ProjLTC**: Uniform light sampling and ProjLTC, (3) **RIS**: RIS with uniform light and uniform area sampling and (4) **RIS-ProjLTC**: Naive combination of RIS with ProjLTC (section 3.4.1). Note that we do not compare against light sampling methods such as Moreau et al. [41] because Bitterli et al. [7] (RIS) already show superior results in the paper.

We use three scenes with varying numbers and sizes of area lights for the evaluation: (1) Zero-Day, 10K lights, (2) Bistro Interior, 2K lights and (3) Bistro Exterior, 30K lights. All renderings are done at a resolution of 1920×1080 on an NVIDIA RTX 3090. We use the Mean Absolute Percentage Error (MAPE) to quantify the error with respect to a ray-traced 1M spp reference.

Figure 3.1, fig. 3.2 and fig. 3.3 show rendered results with difference images for our and previous methods. Our method traces a nearly equivalent number of samples as RIS while achieving lower error. Figure 3.4 shows equal time convergence graphs for all methods. These graphs are plotted for diffuse



(a) Zero-Day



(b) Bistro Exterior

(c) Bistro Interior

Figure 3.3: We show the equal time comparison of our method (section 3.4.2), previous work, and the naive combination of RIS with ProjLTC (section 3.4.1). We show three scenes with diffuse (top) and glossy (bottom) variants and difference images. The reference is rendered with 1M samples. Our method achieves lower error when compared to all other methods.

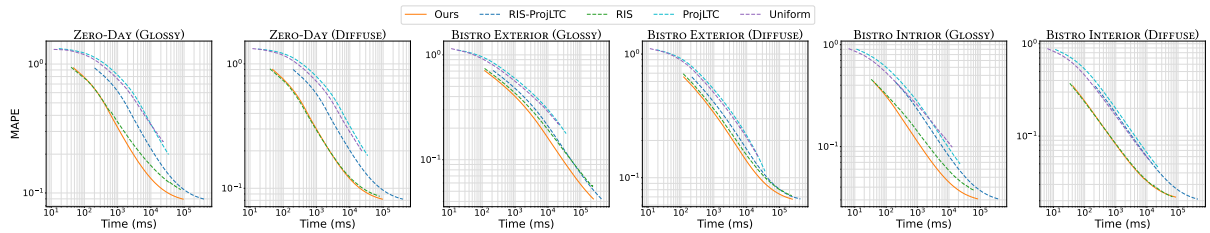


Figure 3.4: We plot equal time convergence graphs of our method (orange, section 3.4.2), RIS-ProjLTC (blue, section 3.4.1), RIS (green), ProjLTC (cyan) and uniform sampling (purple) for both diffuse and glossy versions of three test scenes. Our method achieves a lower Mean Absolute Percentage Error (MAPE) in the same amount of time.

Table 3.1: We show the time to render one frame in milliseconds (ms) and equal sample MAPE for RIS and our method on three scenes.

			MAPE			
	RIS	Ours	100 spp		2K spp	
Scene	(ms)	(ms)	RIS	Ours	RIS	Ours
Zero-Day (10K)	3.98	4.70	0.499	**0.458**	0.173	**0.131**
Bistro Int. (2K)	3.33	4.06	0.214	**0.168**	0.066	**0.047**
Bistro Ext. (30K)	11.85	12.03	0.411	**0.378**	0.135	**0.107**

and glossy versions of the three scenes mentioned above. These graphs show that our method converges to a lower error in equal time and that our method is most beneficial for glossy scenes. Note that it never performs worse than RIS in the case of diffuse scenes. Also note that our method's convergence graph is essentially a translation of RIS-ProjLTC, clearly demonstrating that our method improves its efficiency.

Finally, we show the time taken to render one frame in milliseconds (ms) and equal sample MAPE of our method in comparison to RIS in Table 3.1. On average, our method takes about 1ms more time to render but achieves a lower error.

3.7 Discussion

In this chapter, we presented a naive combination of Resampled Importance Sampling (RIS) and Projected Solid Angle Sampling (ProjLTC) for rendering scenes with many area lights. We identified issues that make this method inefficient. We then proposed a reformulation of RIS that improved the efficiency of this combination, achieving a similar run-time as RIS with a lower error. We analyzed our method's convergence and compared it with previous methods demonstrating superior quantitative and qualitative performance.

Our method makes a contribution to improving the convergence of RIS. ReSTIR uses RIS at its core and thus ReSTIR itself and any improvement on it will benefit from our contribution. For example, the spatio-temporal reuse of samples as already done by ReSTIR can use our method instead to further

improve the convergence. It should be noted that our reformulated RIS will always be faster than the naive combination of RIS and ProjLTC, independent of the hardware and the implementation. To see why, consider that ProjLTC has to evaluate LTCs and then do a few more operations on top for sampling. Since our method stops after evaluating LTCs, it will always be faster. Our method, like RIS (and ReSTIR) naturally supports textured area lights. Note however that in this case, the target PDF of RIS will be further away from the true target which may affect convergence. One of the limitations of our method is that it is limited to single-lobed BRDFs as LTCs cannot represent multi-lobed BRDFs.

For future work, we would like to investigate the performance of using LTCs instead of ProjLTC with our RIS reformulation. We would also like to investigate the performance in the presence of textured area lights. Finally, using LTCs for sampling instead of the final evaluation, to achieve unbiased rendering is a recent trend that we would like to further explore.

Chapter 4

Efficient Rendering of Glinty Appearance

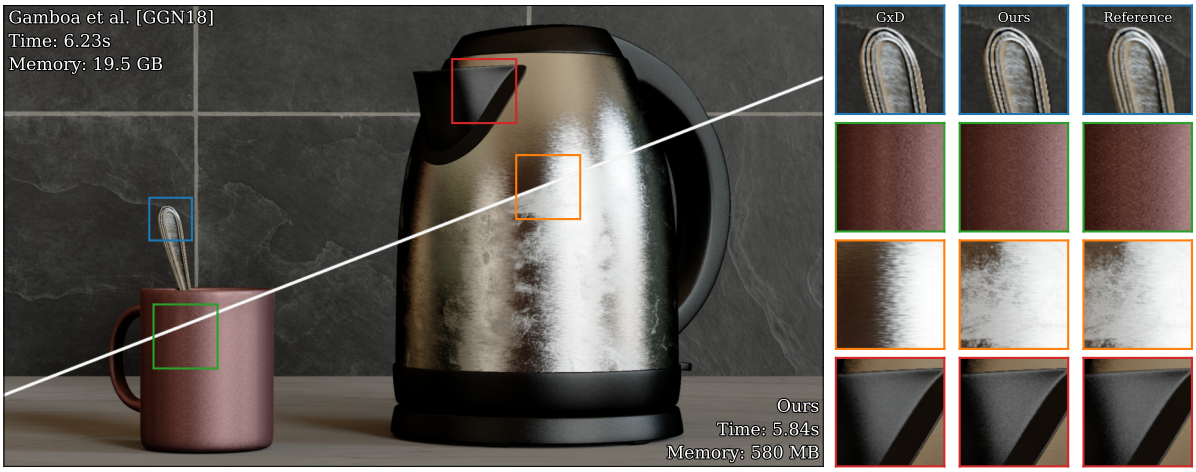


Figure 4.1: Our method improves realism by supporting high-frequency normal-mapped surfaces with spatially varying roughness (orange). Gamboa et al. [20] doesn't support spatially varying roughness and doesn't capture the smudge effect on the kettle. Our method has modest memory requirements, even with different normal maps (we show four). Furthermore, we also reduce errors (green) and improve performance (1spp).

4.1 Introduction

In this chapter we present an efficient method to render scenes containing objects with glinty appearance. Complex glinty appearance that arises from detailed normal mapped surfaces at different scales requires computing expensive per-pixel Normal Distribution Functions; this integration is further compounded by large light sources. Existing solutions focus on fixed material roughness. We present a new method that supports spatially-varying roughness based on a neural-histogram that computes per-pixel NDFs with arbitrary positions and sizes. Our representation is both memory and compute-efficient. Additionally, we compute full direct illumination integration for all light directions with $\mathcal{O}(1)$ effort. Our method shows an improved quality compared to previous works by supporting smaller footprints while offering GPU-friendly computation and compact representation.

Many real-world objects exhibit a rich, glinty appearance due to mesoscale features like scratches, bumps, or flakes. Efficiently modeling these appearances at different scales is crucial for achieving realism in the rendering process. This phenomenon can arise even under large light sources, requiring additional integration.

Motivation Material appearance is often modeled using high-resolution normal maps. Unfortunately, normal map filtering is not a linear process and requires special filtering techniques to keep the appearance of the surface across scales. Dedicated data structures [4,20] or hierarchical representations [14,74] are used to model the underlying normal distribution (NDF) inside the pixel footprint. These techniques can be expensive in memory or not support analytical integration over large light sources.

The NDF typically has a standard deviation parameter that models the roughness of the surface. The prior focus has been on efficiently computing the per-pixel NDF for constant roughness across the surface. Constant roughness, however, does not exist in practice. Material mixing or impurities cause changes in roughness that cannot be described fully with a normal map. Such details significantly improve realism (fig. 4.1) and cannot be reproduced with existing techniques.

We propose a new method to accurately and efficiently compute per-pixel NDF from high-resolution normal maps and varying roughness under complex lighting. We achieve this by combining a roughness-independent binned NDF [20], with a novel neural-histogram and an improved adaptive discretization. Our representation needs $60\times$ less memory for the underlying distribution compared to fixed roughness. Our approach introduces support for spatially varying roughness at negligible additional cost both in memory and performance. Our main contributions are:

- An adaptive binning strategy to reduce discretization error of the continuous $\mathcal{P}$-NDF.
- A neural network-based method for obtaining the spherical histogram for a patch, reducing the memory requirements and improving the performance of the algorithm.
- Support for double filtering both the material appearance and lighting using Spherical Harmonics (SH).
- A compact and tabulated Zonal Harmonics (ZH) representation and efficient on-the-fly rotation to render glinty surfaces with spatially-varying (SV) intrinsic roughness.

4.2 Related work

To the best of our knowledge, no existing technique supports rendering complex high-resolution normal-mapped surfaces with spatially varying roughness. For this reason, we relate to existing works grouped by the way they represent and query the NDF.

Microfacet theory: Similarly to prior works, our method is based on microfacet theory and targets to support the filtering operation of the NDF at different scales. However, unlike the analytical model [5, 65], our BSDF model parameterizes the NDF by using a normal map. Support for SV roughness in the analytical model is possible by relying on linear filtering of a roughness texture or top-level integration. However, this approach cannot be used for normal-mapped NDFs.

Implicit representations: Some prior works use an implicit representation for the NDF, meaning the normal distribution is generated on the fly. For instance, Jakob et al. [29] stochastically produce discrete oriented facets and determine how many are well aligned based on the spatial and directional support. Similar ideas have been adapted to real-time rendering based on bi-scale NDF [78], and physically-based multi-scale precomputed NDF [10]. Using neural networks, we explicitly express the NDF from the normal map and target offline rendering.

Explicit NDF Filtering the underlying normal map accurately is a challenging task. LEAN [46] and LEADR [16] express multiscale auxiliary statistics such as mean and variance that are compatible with MIP-mapping [67]. These methods are cheap and effective but cannot model complex materials' glint distributions across different scales accurately. Yan et al. [72] developed a hierarchical technique to prune normals in the NDF that do not contribute to the final shading based on a fixed intrinsic roughness parameter. This technique is highly accurate, but its cost increases proportionally with footprint size. Later approaches have improved performance [74] or introduced wave-effects [73], however, there's no support for SV roughness or analytic integration of large light sources. Wang et al. [66] accelerate a 4D Gaussian query by blending and synthesizing different distributions. Additionally, their technique supports environment maps using a prefiltering approach, however, they cannot synthesize macro-scale glints. Deng et al. [14] developed a prefiltering method that provides constant-cost for rendering glints. They achieve this by expressing the non-overlapping NDF images for a 32×32 footprint and then compressing this representation using tensor decomposition [33]. Their approach uses MIP-mapping for larger footprints, however, we found that trilinearly interpolated NDFs result in a blurry distribution. In contrast, we use a Binned NDF representation with a neural network to properly interpolate NDF at different scales without introducing blurriness.

Binned NDF: Gamboa et al. [20] combine normal map filtering and environment map lighting. Their technique uses SH expressed in half-direction domain to compute the double product integral between a histogram-based NDF and environment lighting. Although this approach is computationally efficient, it requires a significant amount of memory, i.e., 4.8 GB for a normal map with a resolution of $2K \times 2K$. We borrow from their ideas, improving data representation to increase accuracy, achieve smaller memory footprint and higher performance while adding support for SV roughness. The approach by Atanosov et al. [4] uses a forest of kd-trees to accelerate histogram lookup. While this data representation is not memory intensive ($\sim$10-40 MB depending on histogram resolution), the lookup cost still depends on footprint size. It also loses the ability to filter both surface appearance and environmental lighting.

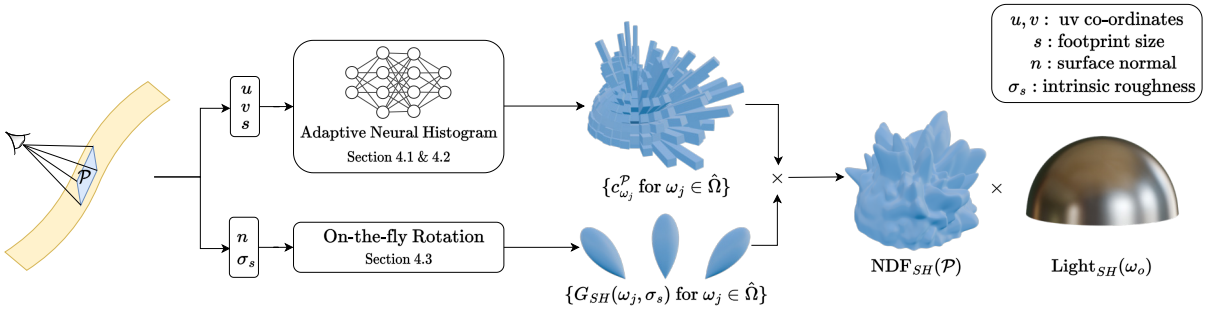


Figure 4.2: For a pixel footprint with location (u, v) and size s, we query our Adaptive Neural Histogram (sections 4.4.1 and 4.4.2) to get bin counts. SH coefficients for each rotated Gaussian lobe with roughness σ_s are obtained through on-the-fly Zonal Harmonics rotation (section 4.4.3). Multiplying these coefficients with the corresponding bin count yields the SH vector of the $\mathcal{P}$-NDF. Finally, a dot product of the $\mathcal{P}$-NDF and Light SH vectors (section 4.3.2) results in the final shading.

Neural-network in material models: Neural models [44] or differentiable indirection [12] have proven to be an alternative versatile compression scheme. In particular, neural networks have been used extensively for BSDF appearance compression [18, 27, 51] or improving filtering for SVBRDF map [22]. A recent approach by Zeltner et al. [76] demonstrates how careful network design and hierarchical MIP-mapped latent code [37] can capture detailed surface appearance at different scales. We also use NeuMIP [37] in our technique to compress and pre-filter our representation.

Note that most deep appearance models require MC integrations of the incoming lighting. Specialized approaches have been developed to generate samples proportional to the neural material. Xu et al.'s [70] histogram approach can be seen as similar to ours, however, our histogram captures the NDF profile and supports pre-filtering operations.

4.3 Preliminaries

Our method is memory-efficient, GPU-friendly, and capable of analytical integration. We employ explicit NDF and Spherical Harmonics (see section 2.4.2) for surface appearance modeling and light integration. We also support spatially varying roughness and don't need additional precomputations when the roughness map changes. Yan et al. [74] and Atansov et al. [4] solve for a single roughness value, resulting in fixed representations. We now explain the relevant concepts briefly.

4.3.1 Explicit NDF

Microfacet models [11] have been traditionally used for describing the BRDF at a point $\boldsymbol{x}$

$$f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = \frac{F(\boldsymbol{\omega_i} \cdot \boldsymbol{\omega_o}) G(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) D(\boldsymbol{x}, \boldsymbol{\omega_h})}{4(\boldsymbol{n} \cdot \boldsymbol{\omega_o})(\boldsymbol{n} \cdot \boldsymbol{\omega_i})}, \tag{4.1}$$

where $\boldsymbol{\omega_i}$, $\boldsymbol{\omega_o}$ are the incident and outgoing directions respectively, $\boldsymbol{n}$ is the geometric normal at $\boldsymbol{x}$ and $\boldsymbol{\omega_h} = \frac{\boldsymbol{\omega_i}+\boldsymbol{\omega_o}}{\|\boldsymbol{\omega_i}+\boldsymbol{\omega_o}\|}$ is the half-vector. F and G are the Fresnel and shadowing-masking terms, respectively, and the *normal distribution function* (NDF), $D(\boldsymbol{x}, \boldsymbol{\omega_h})$, describes the distribution of microfacets on the surface. In this work, we focus on explicit NDFs defined by an underlying high-resolution normal map instead of infinitesimally small microfacets [72].

The explicit NDF ($\mathcal{P}$-NDF) is defined by a set of normals present in a patch $\mathcal{P}$ on the normal map. The patch is given by projecting the pixel footprint on the normal map. [72] and subsequent work perform 2D gaussian-weighting across the footprint. Contrary to this, we use equal-weighting similar to prior work on discrete $\mathcal{P}$-NDF [4, 20, 29]:

$$D(\mathcal{P}, \boldsymbol{\omega_h}) = \frac{1}{N_{\mathcal{P}}} \sum_{\boldsymbol{x} \in \mathcal{P}} G_r(\boldsymbol{\omega_h}; \boldsymbol{\omega_x}, \sigma_s), \tag{4.2}$$

where G_r is an isotropic Gaussian that enables intrinsic roughness σ_s around a mean direction obtained from the normal map, i.e., $\boldsymbol{\omega_x} = \mathrm{normal}(\boldsymbol{x})$, and $N_{\mathcal{P}}$ is the number of normals in the patch.

We can further discretize normals in the normal map into a set $\widehat{\Omega}$ of directions $\boldsymbol{\omega_j}$, resulting in the binned $\mathcal{P}$-NDF:

$$\widehat{D}(\mathcal{P}, \boldsymbol{\omega_h}) = \frac{1}{N_{\mathcal{P}}} \sum_{\boldsymbol{\omega_j} \in \widehat{\Omega}} c^{\mathcal{P}}_{\boldsymbol{\omega_j}} G_r(\boldsymbol{\omega_h}; \boldsymbol{\omega_j}, \sigma_s)\,, \tag{4.3}$$

where $c^{\mathcal{P}}_{\boldsymbol{\omega_j}}$ indicates the count for normal $\boldsymbol{\omega_j}$ inside footprint $\mathcal{P}$. We discuss the set $\widehat{\Omega}$ and present our solution in section 4.4.1. Gamboa et al. [20] store this binned NDF as a spherical histogram, which is memory inefficient. To this end, in section 4.4.2 we propose a neural-based histogram for efficiently querying arbitrary footprints and achieving a low-memory profile.

4.3.2 Filtered Appearance Rendering

The light transport equation described in eq. (2.1) calculates the radiance at a single point $\boldsymbol{x}$. However, a camera pixel covers multiple such points, forming a patch $\mathcal{P}$. The contributions from all points in $\mathcal{P}$ must be considered to obtain a correct answer. To consider this, we use a modified version of the LTE that averages the contributions of all points in the patch $\mathcal{P}$ as follows:

$$L_o(\mathcal{P}, \boldsymbol{\omega_o}) = \int_{\mathcal{P}} \int_{\Omega} L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) f_r(\boldsymbol{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\boldsymbol{n} \cdot \boldsymbol{\omega_i}) d\boldsymbol{\omega_i} d\boldsymbol{x}\,. \tag{4.4}$$

Monte Carlo integrators perform this filtering task implicitly at no extra cost per sample, however, using high-resolution normal maps requires extremely high sample counts to resolve all normal variation and lighting effects correctly [72]. Under a far-field assumption and a microfacet model, we can consider all normal filtering and variation inside the $\mathcal{P}$-NDF [29, 72]:

$$L_o(\mathcal{P}, \boldsymbol{\omega_o}) \approx \int_{\Omega} \frac{L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) F(\boldsymbol{\omega_i} \cdot \boldsymbol{\omega_o}) G(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) \widehat{D}(\mathcal{P}, \boldsymbol{\omega_h})}{4(\boldsymbol{n} \cdot \boldsymbol{\omega_o})} d\boldsymbol{\omega_i}\,, \tag{4.5}$$

with $\mathcal{P}$ computed around the observed point $\boldsymbol{x}$ using e.g. ray differentials [28].

We can approximate eq. (4.5) by decoupling F and G terms and then re-parameterize to half-direction space to form a double product Light-NDF integral. The effective problem to solve becomes:

$$\begin{aligned} L_o(\mathcal{P}, \boldsymbol{\omega_o}) &\approx \widehat{FG}(\boldsymbol{\omega_o}) \int_\Omega L_i(\boldsymbol{x}, \boldsymbol{\omega_i}) \widehat{D}(\mathcal{P}, \boldsymbol{\omega_h}) d\boldsymbol{\omega_i} \\ &= \widehat{FG}(\boldsymbol{\omega_o}) \int_{\Omega_h} \underbrace{\widehat{L_i}(\boldsymbol{x}, \boldsymbol{\omega_o}, \boldsymbol{\omega_h}) 4(\boldsymbol{\omega_h} \cdot \boldsymbol{\omega_o})}_{\text{Light}} \underbrace{\widehat{D}(\mathcal{P}, \boldsymbol{\omega_h})}_{\text{NDF}} d\boldsymbol{\omega_h} \,. \end{aligned} \tag{4.6}$$

Going between $\widehat{L_i}(x, \boldsymbol{\omega_o}, \boldsymbol{\omega_h})$ and $L_i(\boldsymbol{x}, \boldsymbol{\omega_i})$ is trivial as $\boldsymbol{\omega_i} = \text{reflect}(\boldsymbol{\omega_o}, \boldsymbol{\omega_h})$. Projecting each function to Spherical Harmonics (SH) allows solving eq. (4.6) efficiently with:

$$L_o(\mathcal{P}, \boldsymbol{\omega_o}) \approx \widehat{FG}(\boldsymbol{\omega_o}) (\text{Light}_{\text{SH}}(\boldsymbol{\omega_o}) \cdot \text{NDF}_{\text{SH}}(\mathcal{P})) \,. \tag{4.7}$$

Successfully computing these coefficients is expensive, as projecting a function is an integration process, and under eq. (4.7), each SH vector is dependent on either view direction or footprint. Light_{SH} can be precomputed densely for $\boldsymbol{\omega_o}$ for environment emitters.

Additionally, Light_{SH} and NDF_{SH} are in the global and local frame, respectively, which mandates a change of frame, i.e., an expensive SH rotation, before the dot product. Gamboa et al. [20] precomputed rotated SH lobes tied to a fixed roughness to accelerate this process. Instead, in section 4.4.3, we show how to obtain an arbitrary roughness NDF_{SH} relying on on-the-fly ZH rotation while lowering memory requirements and improving performance.

4.4 Method

Our method pipeline is shown in Figure 4.2. We introduce an adaptive binning strategy (Sec 4.4.1) that reduces discretization errors and significantly improves rendering quality for equal histogram resolution. Our neural histogram (Sec 4.4.2) compresses and predicts this adaptive representation, allowing efficient appearance filtering. Finally, we propose to change how SH coefficients of $\mathcal{P}$-NDF are calculated, reducing the memory requirement and allowing spatial varying or live editing of intrinsic roughness (Sec 4.4.3).

4.4.1 Adaptive Binning

The binned $\mathcal{P}$-NDF introduces discretization errors, especially at lower intrinsic roughness values or under sharp lighting. This error translates as structural artifacts inside the rendered images (Figure 4.3, left). A naïve way of reducing this error is to increase the histogram resolution at the cost of runtime and storage requirements. We mitigate this by distributing bins based on the distribution of normals in the normal map (Figure 4.3, center). Specifically, we want to identify the set $\widehat{\Omega}$ of $N_\theta \times N_\phi$ bin centers that best represent the underlying distribution. We have analyzed multiple normal maps and observed

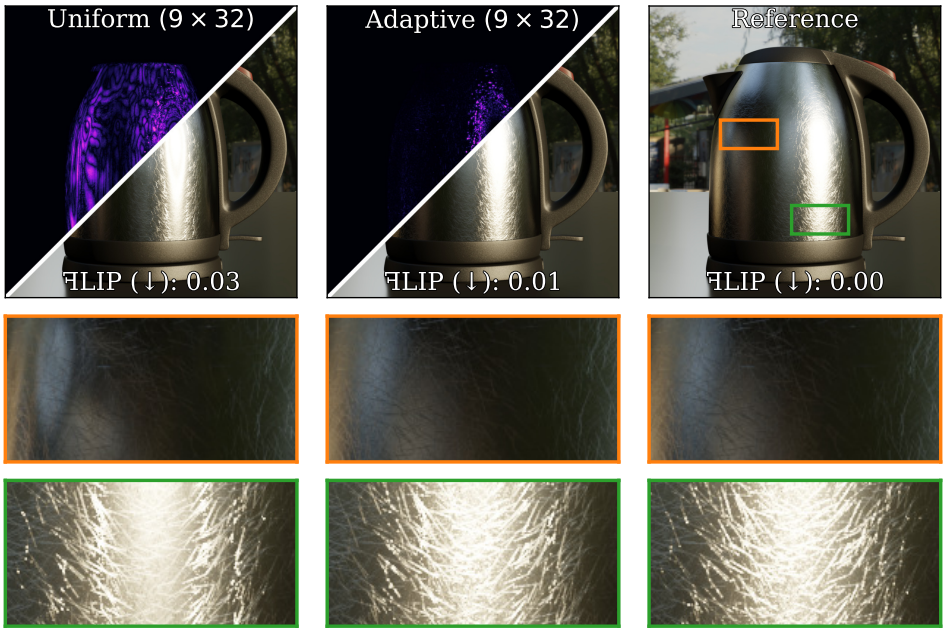


Figure 4.3: Comparing uniform and adaptive binning techniques under low-frequency (orange) and high-frequency (green) lighting. We isolate the impact by using binned NDF and rendering all images with Monte Carlo. Adaptive binning clearly improves quality (ꟻLIP difference scaled by 2×).

that the distribution of ϕ is mostly uniform. Based on this observation, we seek to find only the optimal θ bins while retaining the uniform discretization along ϕ.

Adaptive θ discretization Our approach uses k-means++ [2], which improves the seeding procedure for the k-means algorithm that minimizes an L_2 norm. More specifically, we use the greedy initialization variant of this technique. Compared to simpler solutions, k-means assigns bins to clusters, emphasizing regions with a high concentration of normals and providing a balanced representation important to conserve the material's appearance (Figure 4.3).

4.4.2 Neural Histogram

We present a compact, flexible representation that supports filtering for different pixel footprints and can be easily parallelized on the GPU. This replaces the memory-intensive summed area table in prior work [20]. Our main idea is to compress the underlying distribution by storing a sparse histogram for a given footprint and perform interpolation at run-time to reconstruct the histogram for a given footprint. In practice, we set the minimum footprint size $s = 8 \times 8$ to balance the memory budget and the accuracy of our baseline model. We can explicitly iterate over all the corresponding normals for smaller footprints.

Our approach is similar to Deng et al. [14], which converts a high-frequency normal map into a set of sparsely distributed NDF images organized as a grid and compresses each image NDF using tensor

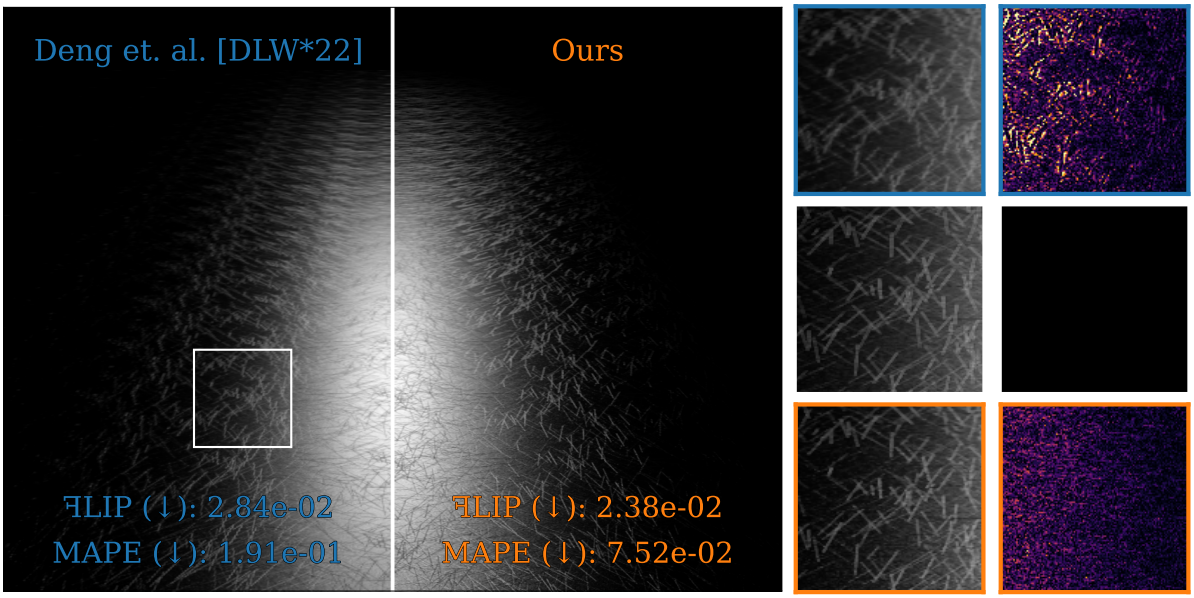


Figure 4.4: Comparison between [14] and our technique. Our use of neural networks results in fewer interpolation artifacts being observed. We include a reference with a square footprint to precisely showcase the difference between the methods.

decomposition [33]. MIP-mapping interpolation is then used to reconstruct the NDF image associated with a given query, decompressing their decomposition for a single direction. In contrast, we obtain directly the requested histogram from our network. Additionally, their representation is tied to the input NDF image set computed from fixed intrinsic roughness. This process fundamentally differs from ours since a histogram is naturally roughness-independent.

In Figure 4.4, we show that trilinear interpolation-based methods suffer from additional blurriness due to linearly blending different NDFs. In contrast, our method uses a deep learning-based approach to produce faithful surface appearances. For this comparison, we use a scratches normal map with a fixed roughness of 0.05 and a point light. Similar results were observed for different normal maps. The reference is computed with MC using square footprints to isolate the differences between the NDF representation and interpolation approaches. In the next subsection, we will empirically describe and justify our network design.

4.4.2.1 Baseline neural model

We adapt NeuMIP design [37], which uses a feature pyramid as a natural solution to our problem. The feature pyramid is arranged as textures with decreasing spatial resolution, similar to MIP-mapping. We obtain the feature vector by trilinearly interpolating inside the feature pyramid for a given footprint size and location. Then, a Multi-Layer Perceptron (MLP) network decodes them to obtain the final histogram for the given footprint.

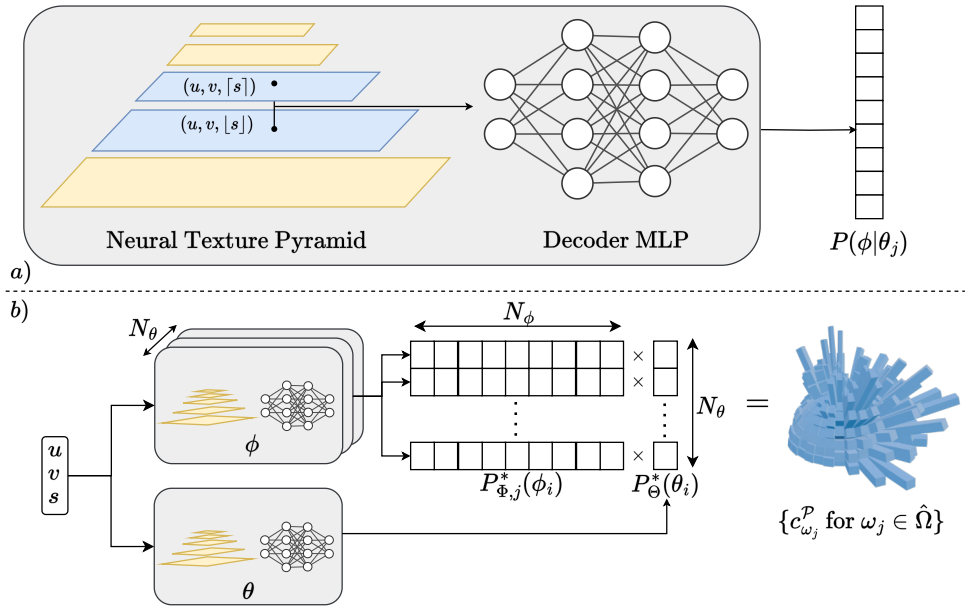


Figure 4.5: Our proposed neural architecture. a) Our neural texture pyramid is inspired by [37] and a decoder MLP, which predicts a partial histogram. b) We decompose the histogram into multiple smaller partial histograms and compress them using $N_\theta + 1$ smaller MLPs.

We experimented by replacing trilinear interpolation to choose one level stochastically [76], but found the additional noise added to final images detrimental as our integration is noise-free. Usually, our targeted histogram resolution is 9×32 bins. For this resolution, the baseline model uses MLP with 4 layers of width 256 each. Each layer is followed by a **ReLU** non-linearity. We apply the **TriangleWave** non-linearity with amplitude of 1 and period of 2 on the final layer since the normalized range of histogram count will always lie in the range $0 - 1$. We normalize this output by the total sum of predicted logits to ensure they sum to 1.

4.4.2.2 Improved network design

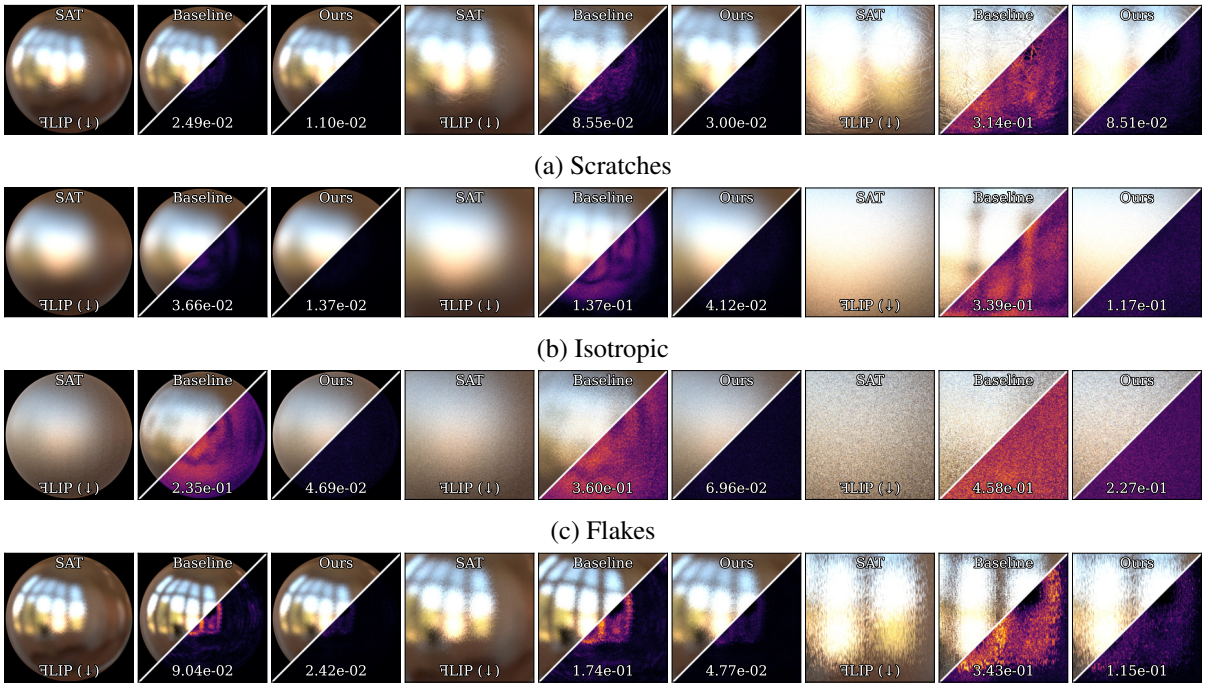
While the baseline method works well, it has two issues. First, the size of the MLP required for high-quality results is too expensive, as it directly affects the performance of histogram queries. Second, scaling to higher histogram resolutions requires an even bigger MLP and potentially larger feature vectors. Keeping multiple MLPs rather than a large one is beneficial as it allows important practical optimization, e.g., operation fusing or exploiting GPU caching architecture.

Based on these issues, we propose partitioning the histogram into partial histograms and training multiple MLPs to predict them (fig. 4.5). The normalized bin count can be expressed as a probability mass function $P(\phi_i, \theta_j)$ for a bin oriented in (ϕ_i, θ_j). We can then decompose this density as

$$P(\phi_i, \theta_j) = P(\phi_i|\theta_j)P(\theta_j) \approx P^*_{\Phi,j}(\phi_i)P^*_{\Theta}(\theta_j)\,, \tag{4.8}$$

where $P^*_{\Phi,j}$ is the learned conditional partial-histogram with θ_j oriented bins, and P^*_{Θ} is the learned densities of ϕs. Figure 4.5 show our proposed architecture. We need $N_\theta + 1$ neural network inference to estimate the full histogram.

4.4.2.3 Baseline vs improved architecture



(a) Scratches

(b) Isotropic

(c) Flakes

(d) Brushed Metal

Figure 4.6: Comparison between the baseline neural model and our multiple small network design for different normal maps and $\mathcal{P}$-NDF at three different scales. The summed area table (SAT) acts as a reference pixel response, but uses $87\times$ more memory than our neural approach. Our network design outperforms the baseline model while maintaining quality across different scales.

Our improved architecture (section 4.4.2.2) relies on partitioning the histogram to make learning the signal easier. Instead of using the baseline model (section 4.4.2.1), which consists of a 256×4 MLP, we can use a much smaller MLP of size 64×2 for better results. Furthermore, since individual evaluations of the smaller MLP are much faster than the larger MLP, evaluating multiple such MLP is still faster. Our improved version is $1.2\times$-$1.3\times$ faster than the baseline model. This difference can be explained as the improved architecture also respects hardware constraints on GPU, allowing faster computation and fused networks.

Figure 4.6 shows that both methods resolve different normal distributions at different scales, however, our improved architecture always produces a more accurate NDF distribution. This results in fewer errors in the generated renders. For reference, we use the NDF SAT representation used in Gamboa et al. [20], which accurately reproduces the NDF distribution at the cost of using $87\times$ more memory than

our neural representation. More comparisons between our model and the technique of Gamboa et al. are available in the results section.

4.4.3 On-the-fly Rotation

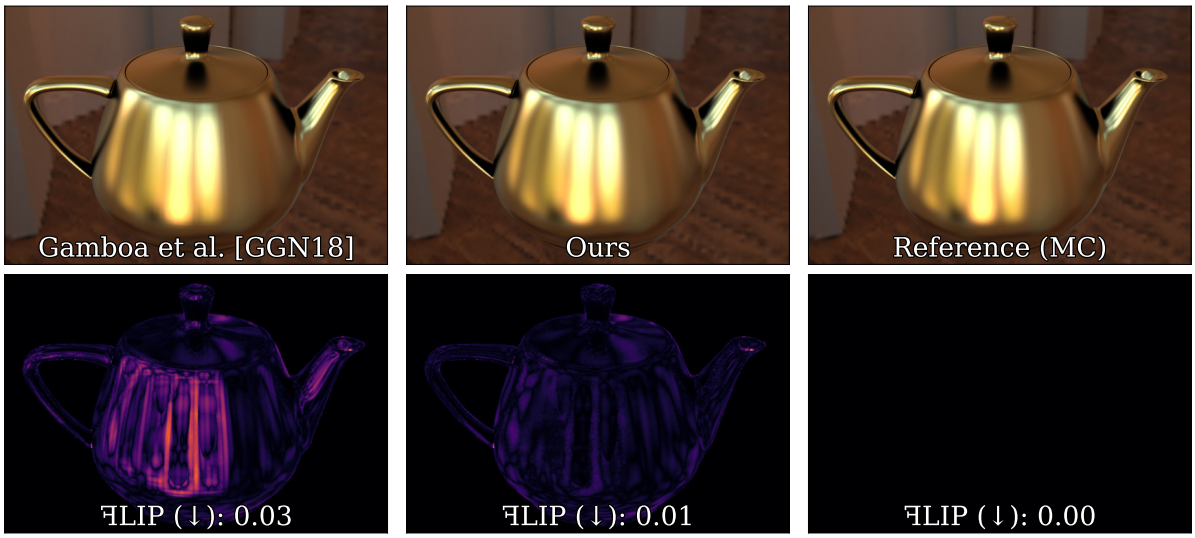


Figure 4.7: Comparison between precomputed [20] and our on-the-fly computation of NDF_{SH} coefficients. The $(2\times)$ ꟻLIP difference (bottom row) shows that our method eliminates block-like artifacts stemming from the discretization of stored SH coefficients.

Given the spherical histogram of a footprint, the SH coefficients of $\mathcal{P}$-NDF, i.e., substitute eq. (4.3) in eq. (2.11), are computed as follows:

$$\text{NDF}_{\text{SH}}(\mathcal{P}) = \frac{1}{N_{\mathcal{P}}} \sum_{\boldsymbol{\omega_j} \in \widehat{\Omega}} c^{\mathcal{P}}_{\boldsymbol{\omega_j}} G_{\text{SH}}(\overline{\boldsymbol{\omega_j}}, \sigma_s) \ , \tag{4.9}$$

where $c^{\mathcal{P}}_{\boldsymbol{\omega_j}}$ is the frequency of $\boldsymbol{\omega_j}$ in $\mathcal{P}$ and $G_{\text{SH}}(\overline{\boldsymbol{\omega_j}}, \sigma_s)$ are the SH coefficients of a Gaussian lobe oriented in direction $\overline{\boldsymbol{\omega_j}}$ and roughness σ_s. Notice that if $\boldsymbol{\omega_j} = \overline{\boldsymbol{\omega_j}}$, then the resulting SH coefficients are in the local frame, however, when solving eq. (4.9) we can go straight to the global frame by making $\overline{\boldsymbol{\omega_j}} = \text{toGlobal}(\boldsymbol{\omega_j})$. Gamboa et al. [20] accelerate this computation by storing G_{SH} coefficients with fixed directions at $4\times$ higher resolution than the histogram resolution (i.e. 65×128). Then, at runtime, they bi-linearly interpolate to compute the SH coefficients for an arbitrary direction. This approach has two issues:

- Interpolation and discretization can generate visual artifacts with low roughness surfaces (fig. 4.7, left).
- The rotation array has fixed σ_s, and storage cost significantly increases if there are objects with spatially varying roughness or if there are objects with different roughnesses.

4.4.3.1 Fast Rotation of Zonal Harmonics

Zonal Harmonics (ZH) are a special case of SH that can represent circularly symmetric functions aligned about z using only the $m = 0$ coefficients. Of interest is the fast rotation property of ZH [55] that allows computing the resulting SH coefficients g_l^m after rotating ZH coefficients f_l^0 to an arbitrary direction $\overline{\boldsymbol{\omega_j}}$ by evaluating and scaling the basis functions Y_l^m in that direction:

$$g_l^m = f_l^0 Y_l^m(\overline{\boldsymbol{\omega_j}}) \sqrt{\frac{4\pi}{(2l+1)}}, \tag{4.10}$$

Since the individual Gaussian lobes are isotropic, we propose using this property and precomputation to perform fast on-the-fly rotation.

4.4.3.2 On-the-fly Rotation

Given ZH coefficients for a canonically oriented Gaussian lobe, we rotate them on-the-fly to remove discretization error (fig. 4.7, center). Furthermore, the storage requirement is reduced drastically from 114 MB to 240 B for order-60 SH, using 32-bit floats.

To perform the fast rotation, we need to evaluate the SH basis functions at a given $\overline{\boldsymbol{\omega_j}}$ direction (eq. (4.10)). To accelerate this operation, we precompute the value of Legendre polynomials densely for different θ, and we bake in the normalization factors as well (see eq. (2.10)). For a resolution of 2048 and order-60, the precomputed data requires a storage space of 30 MB. Notice that this texture is common for the entire scene and remains independent of the roughness or any other scene parameter.

4.4.3.3 Spatially-Varying Roughness

A benefit of our approach is that the reduced memory footprint allows us to precompute and store the ZH coefficients for many roughness values, enabling support for spatially varying roughness. In practice, we precompute the ZH coefficients for roughness values from $0 - 1$ at 1000 uniformly distributed points. At runtime, we linearly interpolate the ZH coefficients from the two nearest σ_s and then perform fast rotation to obtain the desired NDF_{SH}. Storing the ZH values for 1000 σ_s values for order-60 takes roughly 240 KB.

4.5 Implementation

We implement neural networks using TinyCudaNN [42] and PyTorch [48] for training. All experiments are conducted on an NVIDIA RTX 4090 GPU. Training samples are generated by sampling the footprint center $X \sim U(0, 1)$ and a random normalized footprint size $\sigma \sim \lambda e^{-\lambda x}$. Experimentally, we found that setting $\lambda = 16$ works well. We clamp the resulting footprint size to a minimum of 8×8 and train each feature pyramid/decoder independently. We use the Adam [32] optimizer with a learning rate of 10^{-3} for both the network weights and learnable feature vectors. We generate 2^{18} samples per

iteration and train each network for $20k$ iterations, leaving us with ~ 5 billion training samples generated online. Each feature pyramid/decoder pair takes ~ 10 mins to train, which leaves us with a total training time of ~ 1.5 hours. Latent code and network weights are stored in half-precision because it does not affect the quality of our generated images

We train the network to minimize the KL Divergence $D_{KL}(P^*||P)$ between the partial-histogram prediction P^* and ground truth partial-histogram P. The ground truth is obtained efficiently on-the-fly while training uses the SAT proposed by Gamboa et al. [20] with our adaptive binning (section 4.4.1). Note that this SAT is only needed for training and doesn't contribute to the memory footprint during runtime.

4.5.1 Rendering Pipeline

Our method is implemented completely on the GPU and uses Mitsuba 3 [30]. After getting the initial intersection points, we build a G-Buffer consisting of the uv-coordinates and footprint size. We then query our Neural Histogram to predict the histogram at each intersection point. Finally, the histograms are projected to SH while lights SH are calculated using bilinear interpolation in half-vector space. We use a custom CUDA kernel to calculate the double integral product from these coefficients and obtain the final shading. Since we use small MLPs, our method can be implemented in a single mega-kernel with inline MLPs similar to Vaidyanathan et al. [59].

Like Gamboa et al. [20], we also use Ratio Estimators [25] to add visibility information. This introduces some variance noise in penumbra regions, to mitigate this, our images are denoised with Optix AI Denoiser. We naively apply temporal denoising for animated sequences by setting the motion vector to zero as we expect small camera movement. More advanced denoising methods are orthogonal to our technique.

Our work analytically integrates complex lighting (e.g., an environment map) reflected on a normal-mapped surface. No directional integration is required for delta lights, such as point or directional lights. These delta lights can be easily handled using a pruning approach [20] or directly evaluating our neural histogram. We use the former approach as its overhead is negligible, i.e., $\sim$ 5-20 milliseconds.

4.6 Results and Analysis

We compare our technique to the histogram representation and SH computation of Gamboa et al. [20]. All techniques have been implemented on GPU using a 9×32 resolution histogram. We will release our implementation upon acceptance. All references are calculated using Monte-Carlo with high sample counts. It is important to note that while Gamboa et al. [20] supports rectangular queries, we are restricted to a square footprint, however, this limitation is more apparent at grazing angles or other highly anisotropic footprints, where both are inefficient. Finally, we evaluate the results using the LDR-ꟻLIP metric [1]. We display our results with Troy James Sobotka's AgX tone mapping function.

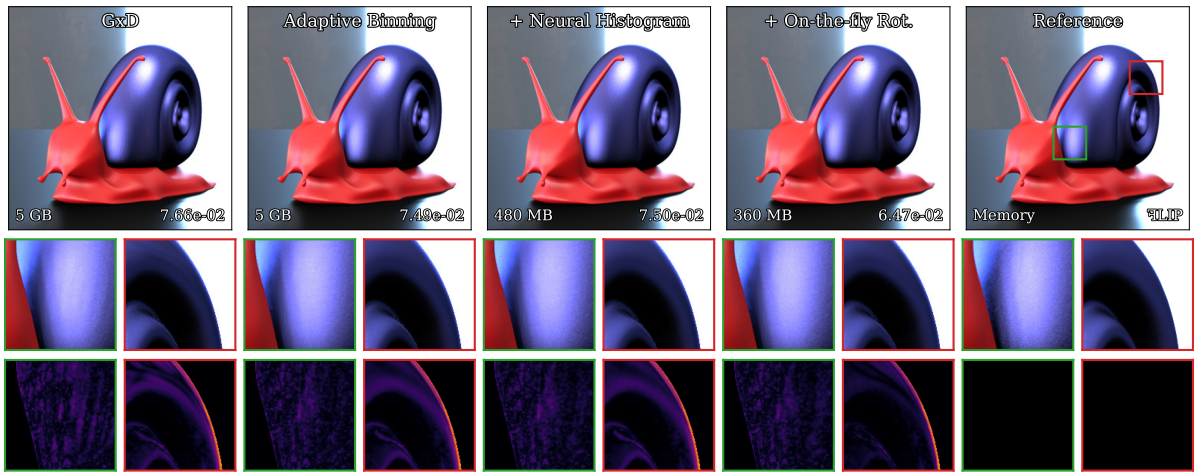


Figure 4.8: From left to right: GxD [20], our adaptive binning, neural network representation, on-the-fly rotation, and the $\mathcal{P}$-NDF reference. The reference is calculated using Monte-Carlo with high sample counts. We observe an improvement in both memory usage and rendering quality with our approach. We use ꟻLIP for the difference image.

Table 4.1: Comparison of memory usage and runtime of our method with GxD [20], using a 9×32 histogram resolution on a $2K \times 2K$ normal map with SH-Order of 60. Environment map coefficient storage is excluded, as it remains the same for both methods. The runtimes are calculated assuming the glinty materials fills the entire screen with resolution 1920×1080. All timings are measured on an NVIDIA RTX 4090 GPU.

	Memory				Runtime (s)		
	Histogram	BSDF Coeffs	$K_l^m P_l^m$	Total	Histogram	Dot Product	Total
GxD	4.8 GB	114 MB	–	4.91 GB	0.69	9.15	9.84
Ours	55 MB	240 B	30 MB	85 MB	0.40	7.42	7.82
Ratio	87×	475k×	–	60×	1.7×	1.2×	1.25×

Empirically design study Figure 4.8 shows an ablation study of our design choices compared to the Gamboa et al. [20] technique. The scene uses the isotropic normal map and an intrinsic roughness of 0.01. Our adaptive histogram representation and SH computation techniques have improved rendering quality, while our neural network has significantly reduced memory usage. Even though we need to evaluate MLPs to obtain the histogram, this process is still faster than the SAT NDF representation [20]. This can be attributed to GPUs being generally more bandwidth-limited than compute-limited. Table 4.1 shows the compression ratio achieved compared to prior work except for the light SH coefficients and the runtime comparisons of histogram query and dot product. The light SH coefficients take 342 MB for order 60 SH in both methods. Overall, our technique is faster and one order of magnitude more compact compared to Gamboa et al. [20]'s approach.

Comparison against Gamboa et al. Figure 4.9 shows additional scenes with different normal maps. All scenes have an environment map and a point light. Each scene contains different normal and en-

Table 4.2: We isolate the rendering cost of our materials (1 spp) shown in fig. 4.1. Notice how most rendering time is spent on the background objects and the ratio estimator for shadows ($1k$ spp Monte Carlo).

Object	Normal Map	SH Order	Time (s)	
			GxD	Ours
Spoon	Scratches	60	0.16	0.13
Mug	Flakes	40	0.26	0.29
Handle	Isotropic	40	0.51	0.51
Kettle	Brushed Metal	70	2.04	1.65
Total			2.97	2.58
Ratio Estimator + BG			3.26	

vironment maps. Our technique consistently produces more accurate results than Gamboa et al. [20] while using less memory. These results show that our neural network component does not introduce visible artifacts.

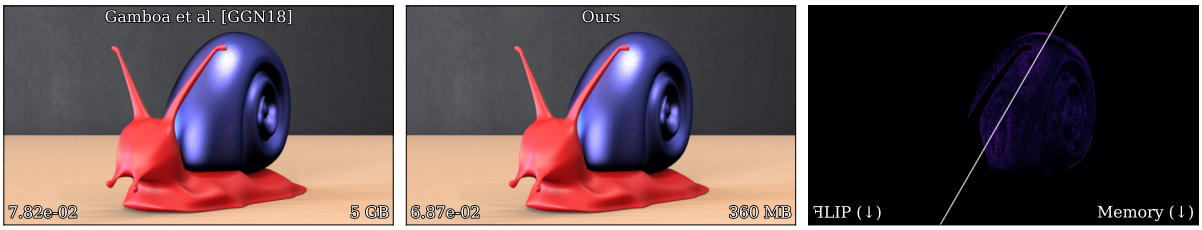
Different histogram resolution Figure 4.10 shows the impact of a higher-resolution histogram on our technique. Increasing the histogram resolution helps to reduce some discretization errors. Our compact representation enables design choices compared to impractical memory requirements by SAT NDF representations [20], which will require 18.4 GB at higher resolution. Higher histogram resolution impacts the storage of the latent code, which is proportional to the number of histogram bins. We must also increase our decoder network from an MLP of size 64×2 to 256×3 to maximize rendering quality. The runtime increase due to a bigger network is negligible compared to the $4\times$ increase in SH dot product computation (Table 4.2).

Spatially-varying roughness Figure 4.1 compares Gamboa et al. [20] (GxD) and our technique with spatially varying roughness. Supporting this is critical for realistic appearances as can be observed by the differences in the kettle object. Our method is more compact and closer to the reference image. Table 4.2 shows that our technique is slightly more efficient than GxD, where most of our computational budget is spent on dot SH computations and the ratio estimator used to compute the shadow of the environment map. Note that our method supports live-editing of the roughness texture. Editing the roughness texture only requires a rendering a new image and not any additional computations, thanks to our roughness-independent NDF representation and our fast and compact SH projections.

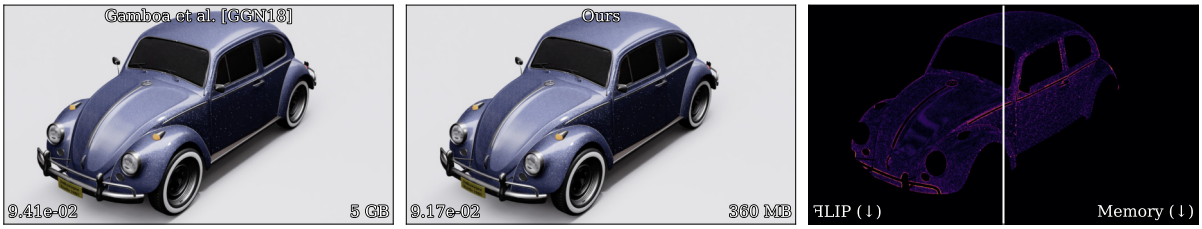
4.7 Discussion

In this work we presented a novel method to capture complex material appearances arising from spatially-varying roughness and high-resolution normal maps.

(a) Cutlery (Scratches)



(b) Snail (Isotropic)



(c) Car (Flakes)

Figure 4.9: Comparison between Gamboa et al. [20] and our method on various complex scenes. The third column shows ꟻLIP error of Gamboa et al. (left) and our method (right) computed against a Monte-Carlo reference with 16k spp. We have a lower ꟻLIP score in all the scenes with a much lower memory footprint.

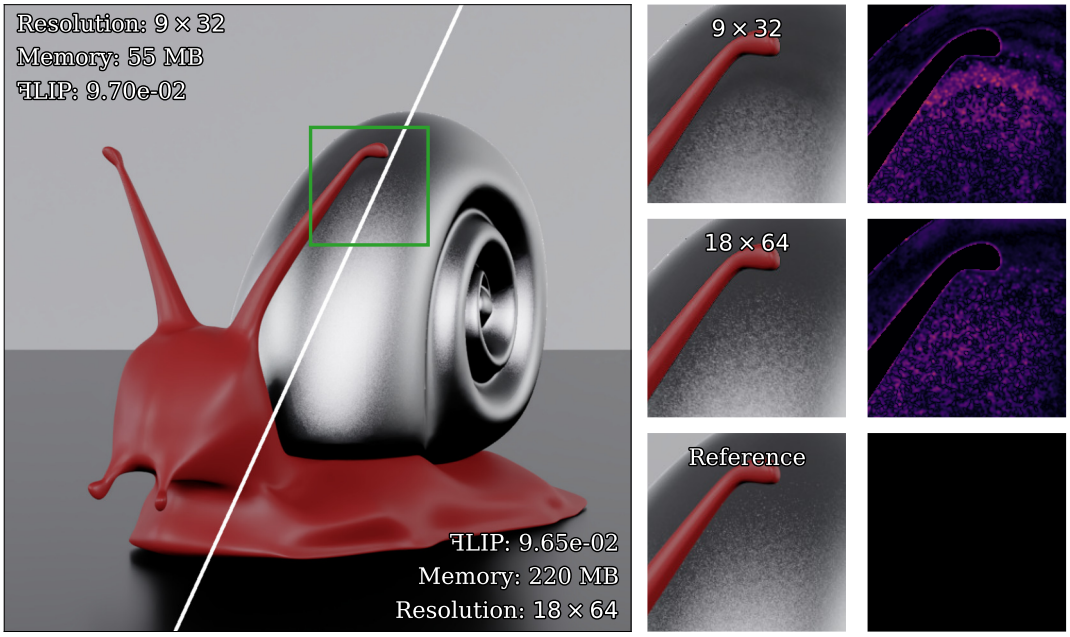


Figure 4.10: Comparison of different histogram resolutions. Increasing the resolution by a factor of two has minor local improvements in discretization error at the expense of increased memory usage. Memory figures are only for our neural histogram storage.

Our solution relies on a neural representation histogram with an improved adaptive binning strategy. We compute the complex $\mathcal{P}$-NDF - Light integral in a single dot product of SH vectors. We believe our technique pushes the realism of materials and opens a venue for future research.

Our method uses square footprints relying on a MIP-mapping style representation. To obtain a better $\mathcal{P}$-NDF, we could use multiple queries, similar to anisotropic filtering, at an added cost. As future work, we want to explore other feature space designs to support anisotropic filtering in a single query, similar to recent work by Deliot et al. [13].

The cost of our neural histogram depends on the normal map and $\mathcal{P}$-NDF resolutions. We experimented with differentiable codebooks [56] and differentiable indirection [12], but found difficulties balancing compression and keeping sharp features of the $\mathcal{P}$-NDF, essential to faithful surface appearance.

Like all SH-based techniques, we suffer from SH order constraints. We can observe that SH computation is one of the biggest bottlenecks to our technique (table 4.1). The fast rotation array for the light SH is 342 MB, which is 4 times more than the memory usage for all our other components. Additionally, our technique only supports environmental lights. As future work, we want to explore alternative bases [71] to overcome these issues.

Chapter 5

Conclusion

In this thesis, we have provided an overview of various approaches for solving physically based rendering. We conclude this thesis by summarizing our contributions.

We first present an improvement to the state-of-the-art method for direct lighting with many area lights. Our use of analytic method (LTC) to evaluate the total illumination from an area light leads to better estimation of the weights in RIS, improving the overall algorithm's convergence rate. We showed better quality results in equal time with the state-of-the-art methods. Our method is more beneficial in the case of glossy material, but it never performs worse than the state-of-the-art, even in the case of diffuse surfaces.

In our second work, we present an efficient way to render glinty appearance. We propose a combination of neural and analytic methods to achieve better results than previous techniques while also requiring less memory. We use neural networks to compress the data structures required for our algorithm. We also propose improvements to the spherical harmonics-based analytic integration to further reduce the memory requirements of the method and improve the quality of the results. Additionally, to the best of our knowledge, our method is the first method to support glinty surfaces with spatially varying roughness.

For future work, we would like to explore the extension of both of our methods to Global illumination. For our first work, one can trivially extend it to global illumination by applying our method at each path vertex. Similarly, for the second work, we can use secondary ray differentials [6] to obtain the footprint of secondary bounces and apply our algorithm to it. Another interesting avenue is to explore rendering glinty surfaces under the influence of area lights using LTCs. The explicit NDF can be represented as a weighted sum of lobes which can be approximated by LTCs and then analytically integrated. The computational complexity and its impact on the runtime remain to be evaluated. Finally, it would also be interesting to use the Spherical Harmonics double product only for guiding the sampling process, similar to our first work [53] and Diolatzis et al. [15]. Doing so would avoid the approximation error due to insufficient SH order, giving us an unbiased estimator with potentially better equal-time convergence.

List of Related Publications

[P1] **Ishaan Shah**[*], Aakash KT[*], and P. J. Narayanan, **"Combining Resampled Importance and Projected Solid Angle Samplings for Many Area Light Rendering"**, *SIGGRAPH Asia Technical Communications*, 2023.

[P2] **Ishaan Shah**, Luis Gamboa, Adrien Gruson, and P. J. Narayanan, **"Neural Histogram-Based Glint Rendering of Surfaces With Spatially Varying Roughness"**, Under Review, 2024.

Other Papers:

[R1] Aryamaan Jain, Jyoti Sunkara, **Ishaan Shah**, Avinash Sharma, and K. S. Rajan **"Automated tree generation using grammar & particle system"**, *ICVGIP*, 2021.

[R2] Chandradeep Pokhariya, **Ishaan Shah**[*], Angela Xing[*], Zekun Li, Kefan Chen, Avinash Sharma, and Srinath Sridhar, **"MANUS: Markerless Grasp Capture using Articulated 3D Gaussians"**, *CVPR*, 2024.

[R3] Cheng-You Lu, Peisen Zhou, Angela Xing, Chandradeep Pokhariya, Arnab Dey, **Ishaan Shah**, Rugved Mavidipalli, Dylan Hu, Andrew Comport, Kefan Chen, and Srinath Sridhar, **"DiVa-360: The Dynamic Visual Dataset for Immersive Neural Fields"**, *CVPR*, 2024.

Bibliography

[1] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, "ꟻLIP: A Difference Evaluator for Alternating Images," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2020.

[2] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007.

[3] J. Arvo, "Stratified sampling of spherical triangles." Association for Computing Machinery, 1995.

[4] A. Atanasov, A. Wilkie, V. Koylazov, and J. Křivánek, "A Multiscale Microfacet Model Based on Inverse Bin Mapping," *Computer Graphics Forum (Proceedings of Eurographics)*, 2021.

[5] P. Beckmann and A. Spizzichino, *The Scattering of Electromagnetic Waves from Rough Surfaces*. Pergamon Press, 1963.

[6] L. Belcour, L.-Q. Yan, R. Ramamoorthi, and D. Nowrouzezahrai, "Antialiasing complex global illumination effects in path-space," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2017.

[7] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz, "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2020.

[8] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila, "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2017.

[9] S. Chandrasekhar, *Radiative Transfer*. Dover Publications, 1960.

[10] X. Chermain, B. Sauvage, J.-M. Dischler, and C. Dachsbacher, "Procedural Physically-based BRDF for Real-Time Rendering of Glints," *Comput. Graph. Forum (Proc. Pacific Graphics)*, 2020.

[11] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *Computer Graphics (Proceedings of SIGGRAPH)*, 1981.

[12] S. Datta, C. Marshall, Z. Dong, Z. Li, and D. Nowrouzezahrai, "Efficient Graphics Representation with Differentiable Indirection," in *SIGGRAPH Asia 2023 Conference Papers*. Association for Computing Machinery, 2023.

[13] T. Deliot and L. Belcour, "Real-Time Rendering of Glinty Appearances using Distributed Binomial Laws on Anisotropic Grids," *Computer Graphics Forum*, 2023.

[14] H. Deng, Y. Liu, B. Wang, J. Yang, L. Ma, N. Holzschuch, and L.-Q. Yan, "Constant-Cost Spatio-Angular Prefiltering of Glinty Appearance Using Tensor Decomposition," *ACM Transactions on Graphics*, 2022.

[15] S. Diolatzis, A. Gruson, W. Jakob, D. Nowrouzezahrai, and G. Drettakis, "Practical product path guiding using linearly transformed cosines," *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2020.

[16] J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov, "Linear efficient antialiased displacement and reflectance mapping," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 2013.

[17] A. C. Estevez and C. Kulla, "Importance sampling of many lights with adaptive tree splitting," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2018.

[18] J. Fan, B. Wang, M. Hasan, J. Yang, and L.-Q. Yan, "Neural layered BRDFs," in *ACM SIGGRAPH Conference Papers*. Association for Computing Machinery, 2022.

[19] L. Fascione, J. Hanika, M. Leone, M. Droske, J. Schwarzhaupt, T. Davidovič, A. Weidlich, and J. Meng, "Manuka: A batch-shading architecture for spectral path tracing in movie production," *ACM Transactions on Graphics*, 2018.

[20] L. E. Gamboa, J.-P. Guertin, and D. Nowrouzezahrai, "Scalable appearance filtering for complex lighting effects," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 2018.

[21] M. N. Gamito, "Solid angle sampling of disk and cylinder lights," *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2016.

[22] A. Gauthier, R. Faury, J. Levallois, T. Thonat, J.-M. Thiery, and T. Boubekeur, "MIPNet: Neural normal-to-anisotropic-roughness MIP mapping," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 2022.

[23] T. Hachisuka and H. W. Jensen, "Stochastic progressive photon mapping," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 2009.

[24] E. Heitz, J. Dupuy, S. Hill, and D. Neubelt, "Real-time polygonal-light shading with linearly transformed cosines," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2016.

[25] E. Heitz, S. Hill, and M. McGuire, "Combining analytic direct illumination and stochastic shadows," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*. ACM Press, 2018.

[26] S. Hill and E. Heitz, "Advances in real-time rendering - real-time area lighting: a journey from research to production," in *ACM SIGGRAPH 2016 Courses*, 2016.

[27] B. Hu, J. Guo, Y. Chen, M. Li, and Y. Guo, "DeepBRDF: A Deep Representation for Manipulating Measured BRDF," *Computer Graphics Forum (Proceedings of Eurographics)*, 2020.

[28] H. Igehy, "Tracing ray differentials," in *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, 1999.

[29] W. Jakob, M. Hašan, L.-Q. Yan, J. Lawrence, R. Ramamoorthi, and S. Marschner, "Discrete stochastic microfacet models," *ACM Transactions on Graphics*, 2014.

[30] W. Jakob, S. Speierer, N. Roussel, M. Nimier-David, D. Vicini, T. Zeltner, B. Nicolet, M. Crespo, V. Leroy, and Z. Zhang, "Mitsuba 3 renderer," 2022.

[31] J. T. Kajiya, "The rendering equation," *Computer Graphics (Proceedings of SIGGRAPH)*, 1986.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs]*, 2014.

[33] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, 2009.

[34] I. Kondapaneni, P. Vevoda, P. Grittmann, T. Skřivan, P. Slusallek, and J. Křivánek, "Optimal multiple importance sampling," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2019.

[35] A. Kt, A. Jarabo, C. Aliaga, M. J.-Y. Chiang, O. Maury, C. Hery, P. J. Narayanan, and G. Nam, "Accelerating hair rendering by learning high-order scattered radiance," *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2023.

[36] A. Kt, P. Sakurikar, and P. J. Narayanan, "Fast Analytic Soft Shadows from Area Lights," in *Eurographics Symposium on Rendering - DL-only Track*. The Eurographics Association, 2021.

[37] A. Kuznetsov, K. Mullia, Z. Xu, M. Hašan, and R. Ramamoorthi, "Neumip: multi-resolution neural materials," *ACM Trans. Graph.*, 2021.

[38] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman, "Generalized resampled importance sampling: Foundations of ReSTIR," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2022.

[39] D. Lin and C. Yuksel, "Real-time stochastic lightcuts," *Proc. ACM Comput. Graph. Interact. Tech.*, 2020.

[40] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, 2021.

[41] P. Moreau, M. Pharr, and P. Clarberg, "Dynamic many-light sampling for real-time ray tracing," in *Proceedings of High Performance Graphics – Short Papers*. Eurographics Association, 2019.

[42] T. Müller, "tiny-cuda-nn," 4 2021. [Online]. Available: https://github.com/NVlabs/tiny-cuda-nn

[43] T. Müller, F. Rousselle, J. Novák, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2021.

[44] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, 2022.

[45] R. Ng, R. Ramamoorthi, and P. Hanrahan, "Triple product wavelet integrals for all-frequency relighting," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2004.

[46] M. Olano and D. Baker, "LEAN mapping," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2010.

[47] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni, "ReSTIR GI: Path resampling for real-time path tracing," *Computer Graphics Forum*, 2021.

[48] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[49] C. Peters, "BRDF importance sampling for polygonal lights," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2021.

[50] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[51] G. Rainer, A. Ghosh, W. Jakob, and T. Weyrich, "Unified Neural Encoding of BTFs," *Computer Graphics Forum (Proceedings of Eurographics)*, 2020.

[52] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi, "Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination," in *Proceedings of High Performance Graphics*. Association for Computing Machinery, 2017.

[53] I. N. Shah, A. KT, and P. J. Narayanan, "Combining resampled importance and projected solid angle samplings for many area light rendering," in *SIGGRAPH Asia 2023 Technical Communications*. Association for Computing Machinery, 2023.

[54] P. Shirley, C. Wang, and K. Zimmerman, "Monte Carlo techniques for direct lighting calculations," *ACM Transactions on Graphics*, 1996.

[55] P.-P. Sloan, B. Luna, and J. Snyder, "Local, deformable precomputed radiance transfer," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2005.

[56] T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler, "Variable Bitrate Neural Fields," in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*. ACM, 2022.

[57] J. F. Talbot, D. Cline, and P. Egbert, "Importance resampling for global illumination," in *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics Association, 2005.

[58] C. Ureña, M. Fajardo, and A. King, "An area-preserving parametrization for spherical rectangles," *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2013.

[59] K. Vaidyanathan, M. Salvi, B. Wronski, T. Akenine-Moller, P. Ebelin, and A. Lefohn, "Random-access neural compression of material textures," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2023.

[60] E. Veach, "Robust Monte Carlo methods for light transport simulation," Ph.D. dissertation, 1997.

[61] E. Veach and L. J. Guibas, "Optimally combining sampling techniques for Monte Carlo rendering," in *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, 1995.

[62] ——, "Metropolis light transport," in *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, 1997.

[63] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Trans. Math. Softw.*, 1977.

[64] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: A scalable approach to illumination," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2005.

[65] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics Association, 2007.

[66] B. Wang, H. Deng, and N. Holzschuch, "Real-Time Glints Rendering With Pre-Filtered Discrete Stochastic Microfacets," *Computer Graphics Forum*, 2020.

[67] L. Williams, "Pyramidal parametrics," *Computer Graphics (Proceedings of SIGGRAPH)*, 1983.

[68] C. Wyman, *Weighted Reservoir Sampling: Randomly Sampling Streams*. Apress, 2021.

[69] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. sitzmann, and S. Sridhar, "Neural fields in visual computing and beyond," *Computer Graphics Forum (Proceedings of Eurographics State of the Art Reports)*, 2022.

[70] B. Xu, L. Wu, M. Hasan, F. Luan, I. Georgiev, Z. Xu, and R. Ramamoorthi, "NeuSample: Importance sampling for neural materials," in *ACM SIGGRAPH Conference Papers*. ACM Press, 2023.

[71] Z. Xu, Z. Zeng, L. Wu, L. Wang, and L.-Q. Yan, "Lightweight neural basis functions for all-frequency shading," in *ACM SIGGRAPH Asia Conference Papers*. Association for Computing Machinery, 2022.

[72] L.-Q. Yan, M. Hašan, W. Jakob, J. Lawrence, S. Marschner, and R. Ramamoorthi, "Rendering glints on high-resolution normal-mapped specular surfaces," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2014.

[73] L.-Q. Yan, M. Hašan, B. Walter, S. Marschner, and R. Ramamoorthi, "Rendering specular microgeometry with wave optics," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2018.

[74] L.-Q. Yan, M. Hašan, S. Marschner, and R. Ramamoorthi, "Position-normal distributions for efficient rendering of specular microstructure," *ACM Transactions on Graphics*, 2016.

[75] C. Yuksel, "Stochastic lightcuts," in *Proceedings of High Performance Graphics – Short Papers*. Eurographics Association, 2019.

[76] T. Zeltner, F. Rousselle, A. Weidlich, P. Clarberg, J. Novák, B. Bitterli, A. Evans, T. Davidovič, S. Kallweit, and A. Lefohn, "Real-Time Neural Appearance Models," 2023.

[77] J. Zhu, Y. Bai, Z. Xu, S. Bako, E. Velázquez-Armendáriz, L. Wang, P. Sen, M. Hašan, and L.-Q. Yan, "Neural complex luminaires: Representation and rendering," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2021.

[78] T. Zirr and A. S. Kaplanyan, "Real-time rendering of procedural multiscale materials," in *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. Association for Computing Machinery, 2016.