Diss. ETH No. 26050

Efficient Light-Transport Simulation Using Machine Learning

A thesis submitted to attain the degree of
Doctor of Sciences of ETH Zürich
(Dr. sc. ETH Zürich)

presented by
Thomas Müller
MSc ETH in Computer Science, ETH Zürich
born on 04.04.1994
citizen of Germany

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner
Dr. Jan Novák, co-examiner
Prof. Dr. Matthias Zwicker, co-examiner

2019

Abstract

The goal in this dissertation is the efficient synthesis of photorealistic images on a computer. Currently, by far the most popular approach for photorealistic image synthesis is path tracing, a Monte Carlo simulation of the integral equations that describe light transport. We investigate several data-driven approaches for improving the convergence of path tracing, leveraging increasingly sophisticated machine-learning models. Our first approach focuses on the specific setting of "multiple scattering in translucent materials" whereas the following approaches operate in the more general "path-guiding" framework.

The appearance of bright translucent materials is dominated by light that scatters beneath the material surface hundreds to thousands of times. We sidestep an expensive, repeated simulation of such long light paths by precomputing the large-scale characteristics of material-internal light transport, which we use to accelerate rendering. Our method employs "white Monte Carlo", imported from biomedical optics, to precompute in a single step the exitant radiance on the surface of large spherical shells that can be filled with a wide variety of translucent materials. Constructing light paths by utilizing these shells is similarly efficient as popular diffusion-based approaches while introducing significantly less error. We combine this technique with prior work on rendering granular materials such that heterogeneous arrangements of polydisperse grains can be rendered efficiently.

The computational cost of path construction is not the only factor in rendering efficiency. Equally important is the distribution of constructed paths, because it determines the stochastic error of the simulation. We present two path-guiding techniques that aim to improve this distribution by systematically guiding paths towards scene regions with large energy contribution. To this end, we introduce a framework that learns a path construction scheme on line during rendering while optimally balancing the computational rendering and learning cost. In this framework, we use two novel path-generation models: a performance-optimized spatio-directional tree ("SD-tree") and a neural-network-based generative model that utilizes normalizing flows. Our SD-tree is designed to learn the 5-D light field in a robust manner, making it suitable for production environments. Our neural networks, on the other hand, are able to learn the full 7-D integrand of the rendering equation, leading to higher-quality path guiding, albeit at increased computational cost. Our neural architecture generalizes beyond light-transport simulation and permits importance sampling of other high-dimensional integration problems.

Zusammenfassung

Das Ziel dieser Dissertation ist die effiziente Synthese fotorealistischer Bilder auf einem Computer. Momentan ist der bei Weitem beliebteste Ansatz für fotorealistische Bildsynthese Path Tracing, eine Monte-Carlo-Simulation der Integralgleichungen welche Lichttransport beschreiben. Es werden mehrere datengesteuerte Ansätze untersucht, die die Konvergenz von Path Tracing verbessern, in welchen zunehmend fortgeschrittenere maschinelle Lernverfahren wirksam angewendet werden. Der erste Ansatz richtet sich spezifisch auf die "mehrfache Lichtstreuung in durchscheinenden Materialien" wohingegen die folgenden Ansätze in der generelleren "Path Guiding" Rahmenstruktur agieren.

Das Aussehen heller, durchscheinender Materialien besteht hauptsächlich aus Licht, das hunderte bis tausende Male unter der Materialoberfläche gestreut wird. Es wird eine teure, wiederholte Simulation von solchen langen Lichtpfaden vermieden, indem die grossräumigen Charakteristiken des materialinternen Lichttransports vorberechnet werden, welche zur Beschleunigung des Renderings verwendet werden. Diese Methode bedient sich des "weissen Monte Carlo", welches aus der biomedizinischen Optik übernommen wurde, um in einem einzelnen Schritt die Strahldichte zu berechnen, die die Oberfläche von grossen, kugelförmigen Schalen verlässt, welche mit einer breit gefächerten Menge an durchscheinenden Materialien gefüllt sein können. Die Konstruktion von Lichtpfaden mit Hilfe dieser Schalen ist ähnlich effizient wie beliebte Ansätze, die auf der Diffusionstheorie des Lichtes beruhen, jedoch weitaus akkurater. Dieser Ansatz wird schlussendlich mit Vorarbeiten zum Rendering körniger Materialien kombiniert, sodass inhomogene Anordnungen polydisperser Körner effizient gerendert werden können.

Die Berechnungskosten der Konstruktion von Lichtpfaden sind allerdings nicht der einzige Bestandteil der Renderingeffizienz. Ebenso wichtig ist die Verteilung der konstruierten Lichtpfade, da diese den stochastischen Fehler der Simulation bestimmt. Es werden zwei Path-Guiding-Methoden vorgestellt, welche darauf abzielen, diese Verteilung zu verbessern, indem Lichtpfade systematisch in Richtung von Orten mit hohem Energiebeitrag gelenkt werden. Dazu wird eine Rahmenstruktur vorgestellt, in welcher ein Lichtpfadkonstruktionsschema mit dem Rendering mitlaufend maschinell erlernt wird, während die Berechnungskosten des Renderings und des Lernens optimal ausbalanciert werden. In dieser Rahmenstruktur werden zwei neue Lichtpfadkonstruktionsmodelle vorgestellt: eine leistungsoptimierte räumlich und richtungsabhängige Baumstruktur ("SD-tree") und ein auf neuronalen Netzwerken basierendes generatives Modell, welches normalisierende Flüsse verwendet. Der SD-tree ist darauf ausgelegt, das 5-D Lichtfeld

robust zu erlernen, wodurch dieser geeignet für Produktionsumgebungen ist. Die neuronalen Netzwerke können andererseits den kompletten 7-D Integranden der Renderinggleichung erlernen, was zu hochwertigerem Path Guiding führt, wenngleich mit erhöhten Berechnungskosten. Die neuronale Architektur generalisiert über die Lichttransportsimulation hinaus und erlaubt auch die Stichprobenentnahme nach Wichtigkeit anderer hochdimensionaler Integrationsprobleme.

Acknowledgments

I am greatly indebted to a large number of people both at Disney and at the Computer Graphics Laboratory of ETH without whom the work leading up to and including this dissertation would not have been possible.

I want to begin by thanking my ETH advisor, Markus Gross, for welcoming me into the team, for his constant strategic guidance, and for enabling the unbelievably comfortable research environment that joint doctoral students at Disney Research and ETH students can be part of. I further want to thank Jan Novák, my Disney Research advisor, who went above and beyond in mentoring my work. His constant readiness to provide helpful feedback, constructive ideas, and insightful discussions, even during times where most people would sleep, was invaluable.

I also want to thank Marios Papas, who sparked and cultivated my interest in research throughout countless brainstorming sessions during coffee breaks. Without Marios I would not have decided to pursue a doctorate.

I owe further thanks to Thijs Vogels and Brian McWilliams, who were instrumental in my studies of the field of machine learning, always willing to provide helpful guidance. Their expertise in the field was invaluable, going as far as sparking the core idea behind one of the chapters of this dissertation.

Next, I want to thank Disney Research's digital artists Alessia Marra and Maurizio Nitti who created award-winning assets that were specifically tailored to my research projects. Their work resulted in a level of polish that would otherwise have been unattainable.

Additionally, I am incredibly thankful to all the people I was fortunate to collaborate on research with: Ana Dodik, Benedikt Bitterli, Brian McWilliams, Fabrice Rousselle, Jan Novák, Marios Papas, Markus Gross, Magnus Wrenninge, Simon Kallweit, Srinath Ravichandran, Steve Marschner, Wojciech Jarosz, Xianyao Zhang, and Yu-Chen Tsai. It was always a pleasure to work alongside these incredibly smart individuals and I wish many more fruitful collaborations with them to be possible in the future.

Outside of the pure research environment I was in, I want to thank everybody I met in the Hyperion team at the Walt Disney Animation Studios: Andrew Fisher, Brent Burley, Dan Teece, Darren Robinson, David Adler, Joseph Schutte, Mark Lee, Matt Chiang, Patrick Kelly, Peter Kutz, Rajesh Sharma, Ralf Habel, Wei-Feng Huang, and Yining Karl Li. They welcomed me into their group multiple times, allowing me to experiment with my research projects in the context of top-tier

movie production. During my time there, I gained numerous profound insights into the intersection of art and technology that would have been impossible to obtain in a pure laboratory setting.

I am also thankful to all unaffiliated individuals who publicly released tools and assets that were used within this dissertation. These are Wenzel Jakob, for creating the open-source Mitsuba renderer in which all rendering algorithms in this dissertation were implemented and tested, Vorba et al. [2014] and Dinh et al. [2016] for releasing the source code of their work, everybody who worked on the TensorFlow machine-learning platform [Abadi et al. 2015], and the following people and entities who created and/or contributed to virtual scene descriptions that were used throughout this dissertation: Benedikt Bitterli, Jaakko Lehtinen, Jay-Artist, Johannes Hanika, Marko Dabrocić, Miika Aitala, Nacimus, Olesya Jakob, Onrřej Karlík, Samuli Laine, SlykDrako, thecali, Tiziano Portenier, Wig42, and the Stanford 3D Scanning Repository.

Finally, I want to thank my parents, Jochen and Susanne Müller, for always being by my side and supporting me selflessly throughout my entire life. There was never a moment where I could not count on them, to the degree where their support implicitly was the enabling factor for this dissertation. I therefore dedicate this dissertation to them.

To mom and dad

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

Introduction

Our goal in this dissertation is the *efficient* synthesis of photorealistic images (and by extension: videos) on a computer, a process that is known as "photorealistic rendering". The applications of photorealistic rendering are manifold: they currently range from animated feature films and live-action visual effects to product and architectural visualization, all while the video-game industry demonstrates a rising demand as well.

The problem to be solved in image synthesis can be described in a simple manner: *in a virtual 3-D world, light is emitted by light sources, interacts with matter, and then strikes a camera.* ***How much light does each pixel of the camera's sensor observe?*** The entire dissertation is devoted to answering this question computationally as efficiently as possible.

1.1 Problem Definition

To answer the above question with precision, we take a physics-inspired approach where we model the interactions of light with the virtual world, "light transport", in a physically based manner and solve the resulting equations to *compute* photorealistic images. Light transport can be defined in several different ways, the most accurate (currently known to humans) following the photon wave-particle duality as described by quantum mechanics. Since quantum and wave effects are often insignificant at human-visible scales, it is common in the field of computer graphics to follow geometric optics, where light travels in straight lines and only changes direction as it interacts with matter. We also follow geometric optics in this dissertation.

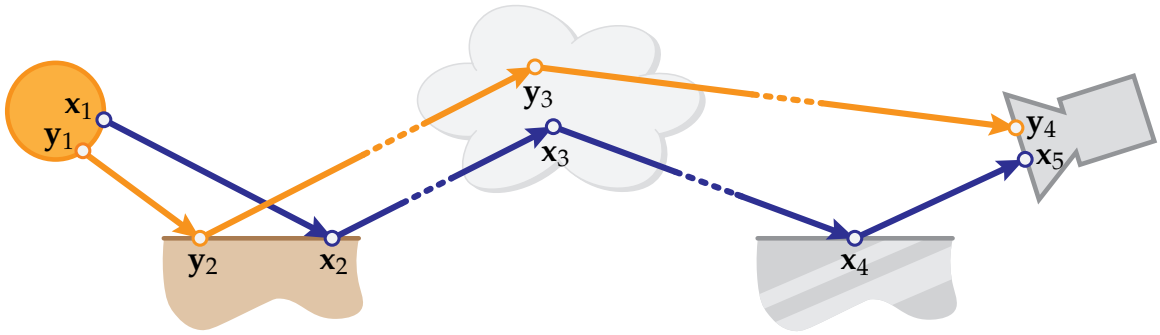


Figure 1.1: *Two paths carrying light from a light source (left) to a camera (right). A light path consists of a number of ordered 3-D vertices, where the ordering determines the direction of light flow (illustrated as directed edges). The light observed by a camera is the sum of the light contributions from all (infinitely many) paths that exist. Typical path-tracing algorithms begin at the camera and trace towards light sources.*

In the framework of geometric optics, the total amount of light observed by a camera pixel is given by the sum of the light carried along *every* possible path of lines that connects a light source with the pixel. We illustrate two examples of such paths in Figure 1.1. Mathematically, summing over the light contributibutions of each path to a given pixel with coordinates x, y is concisely expressed by an integral over the space of paths $\mathcal{P}$

$$I_{xy} = \int_{\mathcal{P}} f_{xy}(\bar{\mathbf{x}}) \, \mathrm{d}\Pi(\bar{\mathbf{x}}) \,, \tag{1.1}$$

where $\bar{\mathbf{x}} = \mathbf{x}_1 \ldots \mathbf{x}_k$ is a path consisting of k 3-D vertices, $f_{xy}(\bar{\mathbf{x}})$ is the light contribution of the path to the given pixel, and Π is a measure on the space of paths. We will define the terms in more detail eventually; for now, we will focus on the aspect that the light reaching each pixel is an integral over paths at all.

Unfortunately, it is generally impossible to express the integral in closed form and we must therefore resort to solving the integral numerically. For several reasons that we will review shortly, the movie and visual effects industry's current numerical technique of choice is Monte Carlo integration, where the integral is approximated by the carefully weighted average contribution of a finite number N of randomly generated paths

$$\int_{\mathcal{P}} f_{xy}(\bar{\mathbf{x}}) \, \mathrm{d}\Pi(\bar{\mathbf{x}}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_{xy}(\bar{\mathbf{x}}_i)}{q_{xy}(\bar{\mathbf{x}}_i)} \,, \tag{1.2}$$

where $q_{xy}(\bar{\mathbf{x}}_i)$ is the probability density of generating the path $\bar{\mathbf{x}}_i$ with respect to the path measure Π.

This computational approach of generating paths of light and evaluating their contribution to the rendered image is one of the most popular techniques for the simulation of light transport.

The most common approach to path generation is an incremental one: vertices are appended after another by repeated ray-casting operations (towards randomly chosen directions) until some termination criterion is fulfilled and the path is considered complete. This kind of approach is well known under the name "path tracing" and became the movie industry's standard over the last decade, a process that is by some referred to as the "path-tracing revolution" [Keller et al. 2015].

1.2 The Path-Tracing Revolution

At the current point in time, every major movie studio produces feature films that are either entirely rendered with path tracing or contain special effects that are. For a long time in the past, however, photorealistic imagery was instead rendered with simpler, more approximate techniques while the burden of creating pleasing visuals rested on artists' abilities to fake realistic lighting in carefully arranged virtual scenes. Path tracing, which *by construction* produces realistically-lit images, promised to alleviate this burden by allowing artists to focus more on the actual content of their scenes.

However, path tracing was simply too slow to be practical, which was largely caused by the enormous number of paths that needed to be traced (on the order of *billions* per image) to achieve acceptable approximation error. But despite of this major limitation, movie studios still adopted path tracing eventually. This major paradigm shift was driven by two factors: the steady exponential increase in compute power following Moore's Law [Moore 1965], which allowed paths to be traced more quickly, and, significant advances in path-tracing algorithms [Christensen and Jarosz 2016], which reduced the number of paths that needed to be traced in the first place.

However, despite of the surge in popularity and practicality of path tracing (or, perhaps, *because* of it), the demand for more efficient algorithms lives on. Using the latest GPU technology, path tracing begins to be used even in the realm of real-time graphics [Burnes 2018], creating renewed interest in advancing the current state of the art towards the metaphorical "holy-grail" which is *real-time* photorealistic rendering.

The contributions in this dissertation—even though they are tailored to off-line production rendering—were conceived with the goal of eventually contributing to real-time rendering. In the remainder of this chapter, we elaborate

on the advantages of path tracing over alternative approximate integration techniques, introduce the role of machine learning in path tracing, list our contributions to the field, and outline the structure of this dissertation.

1.3 Alternatives to Monte Carlo Integration

In the previous section, we explained how path tracing is a Monte Carlo approach for estimating the amount of light being recorded in a virtual camera. What we did not explain is why Monte Carlo would be a desirable method to use despite of its large computational cost in the first place. After all, there exist many alternative approximate integration techniques that could be applied to rendering instead. In the following, we name some examples and explain how they compare against Monte Carlo to illustrate their benefits and drawbacks.

Closed-Form Solutions. Rather than employing a numerical integration technique such as path tracing, one might be tempted to try finding closed-form solutions to an approximate, simplified setting of the problem. While this is a popular approach in real-time redering, it is used rarely in production rendering due to the (sometimes severe) limitations imposed on the admissible virtual scenes.

For example, the reflected light off certain surface materials due to polygonally shaped light sources can be computed in closed form [Heitz et al. 2016a]. Despite the great usefulness of this approach in real-time environments such as video games, true photorealism often demands more sophisticated material models that are not supported by this approach.

Another widely used class of analytic approaches are diffusion-based expressions for multiply scattered light in translucent materials [Stam 1995; Jensen et al. 2001; d'Eon and Irving 2011; d'Eon 2013]. Although such diffusion-based approaches are sometimes orders of magnitude faster to compute than path-traced random walks, they require idealizing assumptions about the underlying surface geometry of the translucent material. For example, the "dipole" approach treats the underlying material as a semi-infinite slab, which is an insufficient approximation of, for instance, a human finger. For this reason, and due to advances in efficient path-traced volume rendering, production renderers gradually move away from diffusion towards pure path tracing [Chiang et al. 2016; Fascione et al. 2018].

Finite-Element Methods. Perhaps the most straightforward approach to numeric integration is the discretization of the continuous domain to a finite set of discrete elements. The solution of the discretized integral—now a Riemann sum—can then either be found in closed form, or, at the very least, approximated more efficiently. Finite-element methods unfortunately come with the drawbacks of finite resolution. The discretization of a continuous integral inherently causes error that only vanishes as the number of discrete elements approaches infnity, with the consequence that convergence to the true solution is only possible using an infinite amount of memory and computational resources. To make matters worse, the performance of many finite element methods scales much worse than linearly in resolution, leading to even stronger practical constraints than limited memory alone leads to. Lastly, high-dimensional integration problems are especially problematic for some finite-element methods because of the "curse of dimensionality": the number of elements required to cover the integration domain with sufficient density grows exponentially in the number of dimensions.

In spite of these limitations, there have been various applications of finite-element methods to physically based rendering in the past, most prominently the "radiosity method" imported from heat-transfer literature by Goral et al. [1984]. The radiosity method subdivides surfaces of the virtual scene into a finite number of patches, allowing—under a simplified material model—to cast the integration problem as linear system of equations. By virtue of subdividing the virtual scene instead of the entire integration domain, the radiosity method avoids the curse of dimensionality to an extent: its discretization covers only the 2-dimensional object surfaces. Unfortunately, the radiosity method can not easily handle materials with complex reflection properties, leading to difficulties when applying it to general light transport.

In contrast to finite-element methods, Monte Carlo integration algorithms work by repeatedly point-sampling the integrand, which requires only a constant amount of memory (the memory needed for the current sample). Furthermore, the asymptotic convergence rate of Monte Carlo algorithms is always guaranteed: the error (here the standard deviation from the true solution) of a Monte Carlo estimator evolves as $\mathcal{O}(1/\sqrt{N})$ in the number of samples. This convergence rate is independent of the dimensionality of the integrand, which is a decisive advantage over finite-element approaches in the domain of light-transport simulation because of the theoretically infinite dimensionality of paths.

One might think, that the samples of a Monte Carlo path tracer are—like finite-element techniques—limited to a low-resolution approximation of the virtual scene to remain efficient. However, this is usually not the case. The

samples of a path tracer are light paths that are generated via successive ray-casting operations, the cost of which is only logarithmic in the amount of scene detail, making memory constraints rather than computational ones the resolution bottleneck. Modern production scenes are sometimes over a hundred giga bytes in size [Walt Disney Animation Studios 2016], which are feasible to path trace because of the logarithmic cost of ray casting. It is worth noting that more efficient point-sampling techniques than ray-casting exist (for example rasterization which is frequently used in real-time graphics), which are, however, not versatile enough to solve the path integral (1.1) in its general form. Nevertheless, it is often worthwhile to consider hybrid techniques to gain performance at additional implementation complexity.

1.4 Machine Learning and Path Tracing

The basic path-tracing algorithm is often referred to as "embarrassingly parallel", owing to the fact that paths are traced completely independently from each other. Although this is an advantage for multiprocessor scalability—each processing unit can trace its own set of paths independently from the other units—there exists untapped potential in the lack of information sharing across paths. To name a simple example: if a path with a larger contribution to the image was randomly generated, then it would be reasonable to assume that a similarly shaped path also contributes significantly to the image. It would therefore make sense to locally explore path space around previously found high-contribution paths.

Past Usage of Machine Learning. The idea of sharing information across paths is not new; it has been practiced in the rendering community for a long time. Some of the most popular applications include irradiance caching [Ward et al. 1988], photon mapping [Jensen 2001], bidirectional path tracing [Lafortune and Willems 1993; Veach and Guibas 1994], Metropolis light transport [Veach and Guibas 1997], and path guiding [Jensen 1995] which has lately found renewed interest [Vorba et al. 2014; Vorba and Křivánek 2016; Herholz et al. 2016, 2018; Guo et al. 2018; Dahm and Keller 2017, 2018; Zheng and Zwicker 2018].

In general, any precomputation that estimates part of the path integral (1.1) ahead of time in such a way that it can be re-used during rendering, is an information-sharing technique (although the reverse is not necessarily the case [Veach and Guibas 1994, 1997]). We argue, that even though computer

graphics researchers typically do not refer to them this way, these precomputation approaches are all, in a way, *machine-learning* approaches. Although some may be considered primitive (e.g. tabulation-based approaches based on weighted histogram density estimation), others use sophisticated optimization machinery (e.g. weighted maximum-a-posteriori mixture of gaussians fitting and non-linear optimization). Inspired by these past successes of machine learning in rendering—and indeed, also their limitations—we strive in this dissertation to develop new machine-learning-based rendering techniques that expand the state of the art.

Deep Learning. During the last couple of years, deep-learning-based techniques became dominant in many fields. Deep learning has emerged as a state-of-the-art technology in image recognition [Simonyan and Zisserman 2014; He et al. 2016], machine translation [Wu et al. 2016], generative modeling of raw audio and natural images [van den Oord et al. 2016a,b], and many more fields. However, until recently, deep learning has seen relatively little usage in light-transport simulation. Although there have been great results in denoising [Bako et al. 2017; Vogels et al. 2018; Chaitanya et al. 2017; Lehtinen et al. 2018], there have been only few applications of deep learning to rendering [Ren et al. 2013; Dahm and Keller 2017; Nalbach et al. 2017]. In this dissertation, our goal is to take another step towards combining deep learning with path tracing to reap the benefits of deep learning that were previously observed in other fields.

1.5 Original Contributions

Motivated by the appeal of the combination of path tracing and machine learning that was outlined above, we focus our work on precisely this topic. In particular, we focus on the machine-learning-based acceleration of *unidirectional* path tracing, where paths originate from the camera and seek out light sources as is most frequently done in movie production [Fascione et al. 2018].

We investigate several data-driven approaches for improving the convergence of path tracing, leveraging increasingly sophisticated machine-learning models. In our first project, we focus on the acceleration of a specific aspect of volumetric path tracing: the rendering of multiple-scattering effects in translucent materials. We then apply our findings to the even narrower context of rendering *granular* materials. Next, we investigate improving the efficiency of more general path-tracing algorithms by proposing a reinforcement-learning-inspired "path-guiding" framework that is tasked with learning to *importance*

sample the path integral. After successfully applying a tabulation-based model to the problem, we shift our focus to the utilization of neural networks for the same task, leading to a technique than can learn to importance sample general high-dimensional target functions.

In the following, we outline our contributions in more detail.

1.5.1 Multiple Scattering in Translucent Materials

We provide a data-driven algorithm for accelerating the rendering of dense translucent materials such as snow, skin, milk, butter, and many more. Such materials are expensive to render with path tracing, because their appearance is dominated by light that scatters many times beneath the material surface—known as high-order scattering—often leading to excessively long paths (sometimes hundreds to thousands of vertices). Our work is inspired by Moon et al. [2007] and Lee and O'Sullivan [2007], who concurrently presented a tabulation-based technique for side-stepping the expensive simulation of long light paths. Although their work achieves impressive speed-up (competitive with popular diffusion approximations), it is applicable only to homogeneous materials (i.e. with spatially constant appearance properties) and requires an expensive precomputation for each material to be rendered. We overcome these limitations by tabulating a large collection of materials at once using a method called "white Monte Carlo" [Alerstam et al. 2008] that was discovered in biomedical optics. Our resulting technique reaps a large portion of the accuracy benefits of the original work [Moon et al. 2007; Lee and O'Sullivan 2007] while only requiring a single, relatively cheap precomputation independent of the scene to be rendered. By sacrificing some additional accuracy, our method is able to handle heterogeneous materials, i.e. those with spatially varying material parameters.

Granular Materials. Furthermore, we apply and evaluate our aforementioned technique to the specific task of rendering high-order scattering in *granular* materials such as sand, snow, and sugar. To this end, we extend existing mechanisms to accelerate high-order scattering in previous work on the topic [Meng et al. 2015; Müller 2016]. We additionally propose an automatic combination of existing low-order scattering (short light paths) [Müller 2016] with our high-order acceleration techniques to obtain a parameterless algorithm that can efficiently render dynamic, heterogeneous granular materials.

1.5.2 Path Guiding

In general, Monte Carlo algorithms can be made more efficient by generating bigger proportions of samples in places where the integrand is large. Since path tracing *is* a Monte Carlo algorithm, this translates to the goal of focusing computational effort towards the construction of those paths that carry most light. "Path guiding" refers to data-driven approaches to *learning* how to direct paths towards high-contribution regions. Perhaps the biggest advantage of path guiding approaches is that they increase efficiency without breaking the correctness of the algorithm: a guided path tracer still converges to the true solution. In this dissertation, we propose two novel path guiding algorithms.

Practical Path Guiding. Our first algorithm uses a tree data structure that we specially designed for learning and subsequently sampling the 5-dimensional light field of the virtual scene on line during rendering. The algorithm subsequently uses the learned light field to *guide* paths towards brightly-lit regions of space during tracing. In our algorithm, we trade off computational effort between training our data structure and simulating light paths in a principled manner that minizes the final approximation error. This approach not only increases the overall efficiency of path tracing, but does so without introducing an expensive precomputation—due to the on-line learning—to leave interactive artist workflows intact.

Neural Path Guiding. Our second algorithm borrows several core algorithmic components from our first algorithm but utilizes deep neural networks rather than a tree data structure to learn how to guide paths. The large modeling capacity of neural networks allows them to not only learn the 5-dimensional light field, but the entire 7-dimensional product of the light field and the material reflectivity. Path guiding according to this product, if it was learned perfectly, would be provably optimal, thereby leading to a *theoretical* best-case scenario of a constant-time zero-error path tracing algorithm. In practice, the neural networks can not perfectly learn a function within finite time and memory constraints, but, nonetheless, result in state of the art quality.

Our algorithm hinges on three key contributions to the sub-field of neural-network-based generative modeling that is based on so-called "normalizing flows". These contributions are independent from path guiding (and light-transport simulation as a whole) and are generally applicable to high-dimensional Monte Carlo integration problems. First, we derive a gradient-

descent-based optimization technique for minimizing the KL and χ^2 divergences of the learned distribution using only estimates of an unnormalized Monte Carlo integrand that were drawn according to an arbitrary probability distribution. Second, we preprocess the inputs of our neural networks using a novel one-blob encoding—a generalization of one-hot encoding [Harris and Harris 2013]—to stimulate localization of computation. Lastly, we introduce a new class of normalizing flows based on piecewise-polynomial functions that yield superior performance compared to previously used flows.

1.6 Dissertation Structure

Prerequisites. We assume that the reader is familiar with basic multi-dimensional calculus, measure theory, and linear algebra. Chapters 2–5 are devoted to introducing all further fundamental theory that is required for understanding our contributions to light-transport simulation.

Part 1: Fundamentals (Chapters 2–5). Chapter 2 covers basic probability theory and Chapter 3 introduces Monte Carlo integration, both of which are fundamental for the understanding of light transport and machine learning. Chapter 4 then covers the geometric-optics view of light transport, some existing applications of Monte Carlo simulation to its solution, and mathematical material models that are used in the virtual scenes that we evaluate our contributions on. Lastly, Chapter 5 introduces a small subset of the field of machine learning that our contributions build on.

Part 2: Technical Contributions (Chapters 6–8). Chapter 6 covers our data-driven algorithm for accelerating the simulation of multiple scattering in heterogeneous translucent materials, and our application of it to the specific use-case of granular materials. Next, in Chapter 7, we present our path-guiding algorithm that learns the 5-dimensional light field on line within a tree data structure. Lastly, in Chapter 8, we describe our neural-network-based sampling contributions to general Monte Carlo integration, which we subsequently apply to path guiding by extending our algorithm from Chapter 7.

Conclusion (Chapter 9). We finally conclude the dissertation in Chapter 9, where we summarize our results and provide an outlook towards interesting opportunities for future work in the intersection of light-transport simulation and machine learning.

Table 1.1: *Publications that some chapters of this dissertation are based on.*

Chapter	Publication
6	**Efficient Rendering of Heterogeneous Polydisperse Granular Media** Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, Jan Novák *ACM Trans. Graph. (Proc. SIGGRAPH Asia), vol. 35, no. 6, pp. 168:1–168:14, Nov 2016*
7	**Practical Path Guiding for Efficient Light-Transport Simulation** Thomas Müller, Markus Gross, Jan Novák *Computer Graphics Forum (Proc. EGSR), vol. 36, no. 4, pp. 91–100, Jun 2017*
8	**Neural Importance Sampling** Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, Jan Novák *ACM Trans. Graph. (to appear)*

1.7 Published Work

Chapters 6–8 and Appendices B–D are based on previously published work [Müller et al. 2016, 2017, 2018] that has been partially reproduced verbatim with explicit permission from the respective co-authors. The aforementiond publications are listed in Table 1.1.

During his doctoral studies, the author has also contributed to the following publications [Kallweit et al. 2017; Bitterli et al. 2018] that are not part of this dissertation:

Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks
Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, Jan Novák
ACM Trans. Graph. (Proc. SIGGRAPH Asia), vol. 36, no. 6, pp. 231:1–231:11, Nov 2017

A Radiative Transfer Framework for Non-Exponential Media
Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, Wojciech Jarosz
ACM Trans. Graph. (Proc. SIGGRAPH Asia), vol. 37, no. 6, pp. 225:1–225:17, Nov 2018

Part I

Fundamentals

CHAPTER 2

Probability Theory

In this chapter we briefly introduce the relevant components of probability theory that are required for understanding the Monte Carlo and machine-learning algorithms that we will present in this dissertation.

2.1 Probabilities and Probability Densities

Let X be a "random variable" taking values in a topological space $\mathcal{D}$ and let

$$P(S) = \text{Prob}(X \in S) \tag{2.1}$$

be the corresponding *probability* that the value of X lies in a measurable subset $S \subseteq \mathcal{D}$; its measure being $\mu(S)$. This probability $P(S)$ is formally called "probability measure" and must satisfy the (intuitive) property of taking values in $[0, 1] \subseteq \mathbb{R}$ and being additive over disjoint sets, i.e.

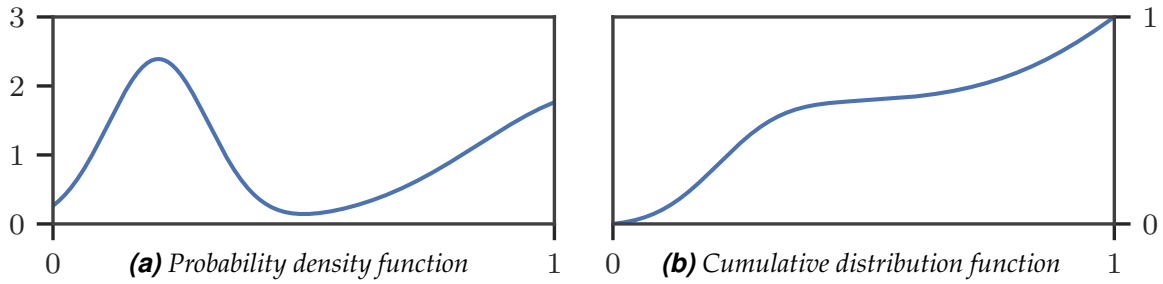
$$\forall S_1, S_2 \in \Sigma(\mathcal{D}) : S_1 \cap S_2 = \emptyset \implies P(S_1 \cup S_2) = P(S_1) + P(S_2)\,, \tag{2.2}$$

where $\Sigma(\mathcal{D})$ are all Borel sets[1] in $\mathcal{D}$.

By taking the Radon-Nikodym derivative of this probability measure we obtain a so-called "probability density function" (or PDF in short)

$$p(x) = \frac{\mathrm{d}P}{\mathrm{d}\mu}(x)\,, \tag{2.3}$$

[1] The exact definition of Borel sets is not important throughout the remainder of this chapter. For intuitive understanding, it is sufficient to treat $\Sigma(\mathcal{D})$ as the set of measurable subsets of $\mathcal{D}$.



(a) *Probability density function* ***(b)*** *Cumulative distribution function*

Figure 2.1: ***(a)*** *An exemplar probability density function (PDF) defined over* $[0,1] \subseteq \mathbb{R}$ *and* ***(b)*** *its corresponding cumulative distribution function (CDF). The PDF must be non-negative everywhere and must integrate to* 1 *over its domain. The CDF—being the indefinite integral of the PDF over the PDF's domain—therefore increases monotonically and ranges from* 0 *to* 1.

where $x \in \mathcal{D}$. If the PDF is known up-front but its corresponding probability measure is not, then the probability measure can conversely be obtained by Lebesgue integration:

$$\text{Prob}(X \in S) = P(S) = \int_S p(x)\,\mathrm{d}\mu(x)\,. \tag{2.4}$$

These general definitions over measurable sets rather than subsets of $\mathbb{R}^D$ are necessary, because we will later deal with probability distributions over very particular sets, such as the surfaces in a virtual 3-D scene.

To make the general concepts easier to grasp intuitively, let us briefly look at the special case where $\mathcal{D}$ is the real number line. Then, the probability measure P is nothing other than the "cumulative distribution function" (or CDF in short)

$$P(x) = \text{Prob}(X \leq x) = \text{Prob}\Big(X \in (-\infty, x]\Big)\,, \tag{2.5}$$

where notation is slightly abused by giving P real-valued input rather than an interval on the number line. The PDF is then

$$p(x) = \frac{\mathrm{d}P(x)}{\mathrm{d}x}\,, \tag{2.6}$$

where the regular derivative (as opposed to Radon-Nikodym) suffices, and the probability of X lying in any given interval $[a,b] \subseteq \mathbb{R}$ is again obtained by the integral

$$\text{Prob}(a \leq X \leq b) = \int_a^b p(x)\,\mathrm{d}x = P(b) - P(a)\,. \tag{2.7}$$

Figure 2.1 illustrates an exemplar PDF and CDF pair for a random variable over $[0,1] \subseteq \mathbb{R}$.

2.1.1 Joint, Marginal and Conditional Probability Density

Suppose the domain of a random variable is the Cartesian product of multiple sets. Without loss of generality, we only treat the case of the two-fold Cartesian product, i.e. $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$; the higher-order product case can be obtained by induction. The random variable can then be split according to the dimensions of $\mathcal{D}$: let X take values of $\mathcal{D}_1$ and Y values of $\mathcal{D}_2$. In this case, the probability measure from before corresponds to the integral over the so-called "joint probability density" $p(x, y)$ of x and y:

$$P(S) = \int_S p(x, y)\, \mathrm{d}\mu_1(x)\, \mathrm{d}\mu_2(y)\,. \tag{2.8}$$

If one of the random variables (e.g. Y) is fixed, then the PDF of the other random variable (in this case X) is the "conditional probability density" $p(x|y)$:

$$p(x|y) = \frac{p(x, y)}{p(y)}\,, \tag{2.9}$$

where $p(y)$ is the "marginal probability density"

$$p(y) = \int_{\mathcal{D}_1} p(x, y)\, \mathrm{d}\mu_1(x)\,, \tag{2.10}$$

which corresponds to an *aggregate* of the joint probability density over all possible values of x. Of course, $p(x, y)$ can also be marginalized over y (or any other axis for higher-dimensional distributions), yielding

$$p(x) = \int_{\mathcal{D}_2} p(x, y)\, \mathrm{d}\mu_2(y)\,. \tag{2.11}$$

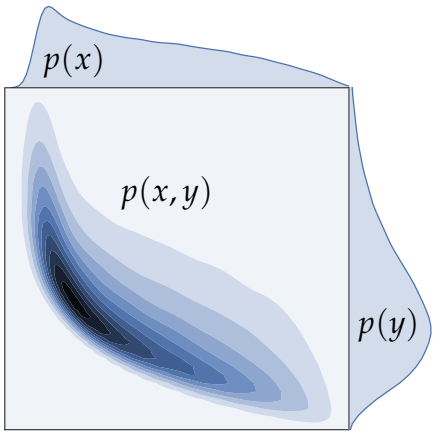


Figure 2.2: *2-d PDF.*

We illustrate the relationship between the joint and its corresponding marginal probability densities for an exemplar PDF in Figure 2.2. While *marginal* densities correspond to *integrals* along the dimensions that are marginalized out, *conditional* densities correspond to *slices* through the joint density that are re-normalized to become valid lower-dimensional PDFs themselves.

Using the definition of the conditional, it is often useful to write the joint as

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x)\,. \tag{2.12}$$

Rearrangement of the above formula yields an identity that is widely known as Bayes' rule

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}\,. \tag{2.13}$$

2.2 Emergent Properties of Random Variables

There exist a number of useful identities on random variables that characterize their overall statistical behavior in certain ways.

2.2.1 Expectation

In this section, let $\mathcal{D}$ be the set of vectors of a vector space over $\mathbb{R}$. The "expected value" (or expectation) of a random variable X living in $\mathcal{D}$ is defined as

$$\mathbb{E}[X] = \int_{\mathcal{D}} x p(x)\, \mathrm{d}\mu(x)\,. \tag{2.14}$$

The intuition behind it is to capture the *mean* of an infinitely large number of realizations of X. This intuition is formalized by the strong law of large numbers

$$\mathrm{Prob}\left(\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} X_i = \mathbb{E}[X]\right) = 1\,, \tag{2.15}$$

where X_i are independent and identically distributed random variables following the same distribution as X.

The expectation is linear for *arbitrary* random variables X and Y, i.e.

$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y] \tag{2.16}$$

$$\mathbb{E}[aX] = a\mathbb{E}[X]\,, \tag{2.17}$$

where a is a constant.

It is also possible to compute partial expectations of multi-dimensional random variables, integrating out only some of the dimensions. Using the same definition of X and Y as in Section 2.1.1, the "conditional expectation" of a

function $f(X, Y)$ on Y is itself a random variable (a function of Y) which is defined as

$$\mathbb{E}\big[f(X,Y)|Y\big] = \int_{\mathcal{D}_1} f(x,Y)p(x|Y)\,\mathrm{d}\mu_1(x) \tag{2.18}$$

$$= \frac{\int_{\mathcal{D}_1} f(x,Y)p(x,Y)\,\mathrm{d}\mu_1(x)}{\int_{\mathcal{D}_1} p(x,Y)\,\mathrm{d}\mu_1(x)}\,. \tag{2.19}$$

In the following, we will sometimes denote $\mathbb{E}\big[f(X,Y)|Y\big]$ as $\mathbb{E}_X\big[f(X,Y)\big]$ to capture the dimensions that are being integrated out. Note, how the regular expectation is recovered by consecutively applying the conditional expectation:

$$\mathbb{E}\big[f(X,Y)\big] = \mathbb{E}_X\Big[\mathbb{E}_Y\big[f(X,Y)\big]\Big]\,. \tag{2.20}$$

2.2.2 Variance

The variance $V[X]$ of a random variable $X \in \mathbb{R}$ is defined as the expected squared difference of X from its expected value[2]:

$$\mathbb{V}[X] = \mathbb{E}\Big[\big(X - \mathbb{E}[X]\big)^2\Big]\,. \tag{2.21}$$

Unlike the expectation, the variance is *not* linear, but has several interesting properties nonetheless. For *independent* random variables X and Y, the variance separates under addition. Furthermore, constant scalars can be moved out by squaring them:

$$\mathbb{V}[X+Y] = \mathbb{V}[X] + \mathbb{V}[Y] \tag{2.22}$$

$$\mathbb{V}[aX] = a^2\mathbb{V}[X]\,. \tag{2.23}$$

These identities allow rewriting the definition of the variance to another form that is often useful to work with

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}\Big[\big(X - \mathbb{E}[X]\big)^2\Big]\\ &= \mathbb{E}\Big[X^2 - 2X\,\mathbb{E}[X] + \mathbb{E}[X]^2\Big]\\ &= \mathbb{E}[X^2] - \mathbb{E}\big[2X\,\mathbb{E}[X]\big] + \mathbb{E}[X]^2\\ &= \mathbb{E}[X^2] - 2\,\mathbb{E}[X]\,\mathbb{E}[X] + \mathbb{E}[X]^2\\ &= \mathbb{E}[X^2] - \mathbb{E}[X]^2\,.\end{aligned} \tag{2.24}$$

[2] Variance is sometimes defined over vector-valued domains, in which case either the variance of individual vector components or of the vector length is meant.

Similar to the expectation, variance can also be defined conditionally. The conditional variance of a function $f(X, Y)$ on Y is itself a random variable (a function of Y) which is defined in terms of the conditional expectation analogously to regular variance:

$$\mathbb{V}\big[f(X,Y)|Y\big] = \mathbb{E}\big[f(X,Y)^2|Y\big] - \mathbb{E}\big[f(X,Y)|Y\big]^2, \tag{2.25}$$

or in different notation

$$\mathbb{V}_X\big[f(X,Y)\big] = \mathbb{E}_X\big[f(X,Y)^2\big] - \mathbb{E}_X\big[f(X,Y)\big]^2. \tag{2.26}$$

The overall variance can be expressed in terms of conditional variances and expectations. Denoting $Z = f(X, Y)$ for brevity, the following identities hold:

$$\begin{aligned}
\mathbb{V}[Z] &= \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 \\
&= \mathbb{E}_X\Big[\mathbb{E}_Y[Z^2]\Big] - \mathbb{E}_X\Big[\mathbb{E}_Y[Z]\Big]^2 \\
&= \mathbb{E}_X\Big[\mathbb{E}_Y[Z^2] - \mathbb{E}_Y[Z]^2\Big] + \mathbb{E}_X\Big[\mathbb{E}_Y[Z]^2\Big] - \mathbb{E}_X\Big[\mathbb{E}_Y[Z]\Big]^2 \\
&= \mathbb{E}_X\Big[\mathbb{V}_Y[Z]\Big] + \mathbb{V}_X\Big[\mathbb{E}_Y[Z]\Big] && (2.27) \\
&= \mathbb{E}_Y\Big[\mathbb{V}_X[Z]\Big] + \mathbb{V}_Y\Big[\mathbb{E}_X[Z]\Big]. && (2.28)
\end{aligned}$$

2.2.3 Covariance and Correlation

Covariance and correlation are useful for quantifying the dependence between random variables. Analogously to Section 2.1.1, suppose the domain of random variables has the form $\mathcal{D} = \mathbb{R}^2$. The "covariance" between a pair of random variables $(X, Y) \in \mathbb{R}^2$ is defined as

$$\operatorname{cov}(X,Y) = \mathbb{E}\Big[\big(X - \mathbb{E}[X]\big)\big(Y - \mathbb{E}[Y]\big)\Big]. \tag{2.29}$$

Intuitively, the covariance captures the degree to which X and Y vary in tandem. A positive value means, that values of X tend to be accompanied with values of the same sign of Y. A negative value has the same property, only with flipped signs. Independent X and Y must have $\operatorname{cov}(X, Y) = 0$, but note that the reverse is not necessarily true. This is the case, because the covariance only captures *linear* relationships between X and Y, but not higher-order ones. It is worth noting that the covariance of a random variable with itself reduces to the variance

$$\operatorname{cov}(X,X) = \mathbb{E}\Big[\big(X - \mathbb{E}[X]\big)^2\Big] = \mathbb{V}[X]. \tag{2.30}$$

Table 2.1: *Various popular f-divergences.*

Name	$f(t)$	$D_f(p \,\|\, q)$
KL-divergence	$t \log t$	$\int_{\mathcal{D}} p(x) \log \frac{p(x)}{q(x)} \,\mathrm{d}\mu(x)$
Pearson χ^2-divergence	$t^2 - 1$	$\int_{\mathcal{D}} \frac{p(x)^2}{q(x)} \,\mathrm{d}\mu(x) - 1$
Total variation distance (L_1 norm)	$\lvert t-1 \rvert / 2$	$\int_{\mathcal{D}} \frac{1}{2} \lvert p(x) - q(x) \rvert \,\mathrm{d}\mu(x)$
Hellinger distance	$\left(\sqrt{t} - 1\right)^2$	$\int_{\mathcal{D}} \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 \mathrm{d}\mu(x)$

The covariance can therefore also be viewed as a *generalization* of variance.

Another property of covariance is that it not only captures the degree to which X and Y vary in tandem, but also the individual magnitudes of variation of X and Y. This leads to possible values of the covariance along the entire real number line $(-\infty, \infty)$, which is often undesirable. When only the relationship between X and Y is of interest, but not their individual variation, it is useful to scale covariance to the interval $[-1, 1]$, producing "correlation":

$$\operatorname{corr}(X, Y) = \frac{\operatorname{cov}(X, Y)}{\sqrt{\mathbb{V}[X]\,\mathbb{V}[Y]}}\,. \tag{2.31}$$

In practice, correlation is a useful indicator of a hidden causal relationship between X and Y, but caution must be taken: although causation implies correlation, the reverse is not necessarily true. To give a real-world example: even though a cloudy sky and rain are correlated, rain does not cause cloudy skies.

2.3 Divergences

Often it is useful to quantify the amount by which two probability distributions differ from each other. "Divergences" are a class of functions that do precisely this. They take two probability distributions as input and return a real output, where larger output values correspond to a bigger differences and a zero output corresponds to both probability distributions being equal.

More formally, let $P, Q \in \mathcal{S}$ be two probability measures with common support $\mathcal{D}$, a divergence $D(P \,\|\, Q)$ is defined to satisfy

$$D(P \,\|\, Q) \geq 0 \quad \forall P, Q \in \mathcal{S} \tag{2.32}$$

$$D(P \,\|\, Q) = 0 \quad \text{if and only if } P = Q\,. \tag{2.33}$$

Perhaps the most commonly used type of divergence are f-divergences, which are defined in terms of the expected odds-ratio P/Q weighted by a convex function f satisfying $f(1) = 0$:

$$D_f(P \parallel Q) = \int_{\mathcal{D}} f\left(\frac{\mathrm{d}P}{\mathrm{d}Q}\right) \mathrm{d}Q\,. \tag{2.34}$$

The constraints on f ensure that D_f satisfies the above requirements for being a valid divergence. We can also express the f-divergence in terms of probability densities

$$D_f(p \parallel q) = \int_{\mathcal{D}} f\left(\frac{p(x)}{q(x)}\right) q(x)\, \mathrm{d}\mu(x)\,. \tag{2.35}$$

Table 2.1 lists several commonly used f-divergences.

2.4 Distribution Models

So far, we defined probability distributions very generally. However, in practice we are often confronted with (or desire) simple distributions that have known convenient properties. Many such distributions were discovered in the history of statistics; far too many to cover here. We therefore limit ourselves to the "uniform" and the "normal" distribution as we will make use of both throughout this dissertation.

2.4.1 Uniform Distribution

The "uniform distribution" is arguably the simplest distribution model. A random variable $X \sim \mathcal{U}(a, b)$ (read X distributed uniformly in $[a, b]$) with $a, b \in \mathbb{R}$; $a < b$ has piecewise-constant PDF and piecewise-linear CDF

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise,} \end{cases} \qquad P(x) = \begin{cases} 0 & \text{if } x \in x < a \\ \frac{x-a}{b-a} & \text{if } x \in [a, b] \\ 1 & \text{otherwise.} \end{cases} \tag{2.36}$$

The uniform distribution nicely generalizes across multiple dimensions: the D-dimensional uniform distribution corresponds to a constant probability density within the prescribed hypervolume and zero probability density everywhere else. To give a uniformly distributed 2-dimensional example, $X, Y \sim \mathcal{U}(a_X, b_X, a_Y, b_Y)$ has marginal distributions $X \sim \mathcal{U}(a_X, b_X)$ and $Y \sim \mathcal{U}(a_Y, b_Y)$ with a joint density of $p(X)p(Y)$ (i.e. X and Y are independent).

2.4.2 Normal Distribution

The "normal distribution" is another important model as it naturally arises in several places of probability theory. It is also sometimes referred to as "Gaussian" distribution, named after Carl Friedrich Gauss.

The PDF and CDF of a normal-distributed random variable $X \sim \mathcal{N}(\mu, \sigma)$ with prescribed mean and standard deviation $\mu, \sigma \in \mathbb{R}$ are

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\left(-\frac{x-\mu}{\sigma\sqrt{2}}\right)^2\right], \quad P(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], \tag{2.37}$$

where $\operatorname{erf}(x)$ is the error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t \,, \tag{2.38}$$

which has no closed-form expression in terms of elementary functions. In practice, numerical approximation of $\operatorname{erf}(x)$ are used.

Like the uniform distribution, the normal distribution generalizes to higher dimensions, in which case it is also referred to as "multivariate normal distribution". In this dissertation, we refer to 1-dimensional (univariate) *and* higher dimensional (multivariate) normal distributions simply as "normal distribution" for convenience. The multivariate normal distribution $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is parameterized by vector-valued mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and matrix-valued covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$:

$$p(x) = \frac{1}{\det(\boldsymbol{\Sigma})\sqrt{(2\pi)^2}} \exp\left[\frac{1}{2}(x-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right]. \tag{2.39}$$

Each entry in the covariance matrix $\boldsymbol{\Sigma}_{ij}$ is the covariance between the i-th and the j-th dimension. Since covariance is symmetric, $\boldsymbol{\Sigma}$ must also be symmetric w.r.t. its diagonal. Furthermore, because the elements on the diagonal $\boldsymbol{\Sigma}_{ii}$ are the variances along the i-th dimensions (covariance with themselves), they must be non-negative. Lastly, the covariance matrix must be positive definite[3].

The normal distribution has a number of remarkable properties that makes it convenient to use in practice. Interesting for us is that various operations on normally distributed random variables $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ produce yet another normally distributed random variable $X' \sim \mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ with easily computable $\boldsymbol{\mu}'$ and $\boldsymbol{\Sigma}'$. These operations include

[3] If we allow degenerate covariance matrices (i.e. Dirac-delta PDFs), this constraint is lifted to positive semi-definiteness.

- marginalization,
- conditioning,
- the density product,
- the sum of random variables (even when they are correlated), and
- affine transformations.

Furthermore, the KL-divergence between two normal distributions can be computed in closed form. Modeling a distribution as normal therefore permits a number of cheap, exact operations that would otherwise either be approximate and expensive, or would require specialized parameterizations. This is especially convenient in the field of Bayesian statistics on which many machine-learning techniques are built on.

Another occurance of the normal distribution is in the central limit theorem: the mean of N independent, equally distributed random variables tends to be normally distributed as $N \to \infty$, regardless of the distribution of the individual random variables. This theorem is useful in the context of Monte Carlo integration, where integrals are approximated by such sums of N independent random variables. It is therefore reasonable to assume that Monte Carlo estimators with large sample counts are approximately Gaussian, which opens the door for a number of analyses and algorithms.

2.5 Converting Probability Densities

2.5.1 Converting Between Measures

It is often useful to express a probability density function in a different measure. Suppose we would like to express a probability density $p(x) = \frac{\mathrm{d}P}{\mathrm{d}\mu}(x)$ not with respect to measure μ, but with respect to μ' instead. Expanding the definition of the desired probability density $p'(x)$ we arrive at the relation

$$p'(x) = \frac{\mathrm{d}P}{\mathrm{d}\mu'}(x) = \frac{\mathrm{d}P}{\mathrm{d}\mu}\frac{\mathrm{d}\mu}{\mathrm{d}\mu'}(x) = p(x)\frac{\mathrm{d}\mu}{\mathrm{d}\mu'}(x)\,, \tag{2.40}$$

which will become useful in the next section.

2.5.2 Converting Between Parameterizations

Another operation that is important in practice is changing the parameterization of a probability density $p(x)$. In this section we investigate changing the

parameterization of x via *bijective* and *absolutely continuous* transformations $h : S_1 \to S_2;\ S_1, S_2 \subseteq \mathcal{D}$ with $x \in S_1$ and $z = h(x) \in S_2$. The transformation h induces a new probability measure

$$P_h(S) = P\big(h^{-1}(S)\big)\,;\ S \subseteq S_2\,, \tag{2.41}$$

the density of which $p_h(z)$ we are interested in knowing. Expanding the definition of $P_h(S)$ we get

$$\begin{aligned} P_h(S) = P\big(h^{-1}(S)\big) &= \int_{h^{-1}(S)} p(x)\,\mathrm{d}\mu(x) \\ &= \int_S p\big(h^{-1}(z)\big)\,\mathrm{d}(h^{-1}\circ\mu)(z) \\ &= \int_S p\big(h^{-1}(z)\big)\frac{\mathrm{d}(h^{-1}\circ\mu)}{\mathrm{d}\mu}(z)\,\mathrm{d}\mu(z)\,, \end{aligned} \tag{2.42}$$

and therefore

$$p_h(z) = p\big(h^{-1}(z)\big)\frac{\mathrm{d}(h^{-1}\circ\mu)}{\mathrm{d}\mu}(z)\,. \tag{2.43}$$

In the special case of $\mathcal{D} = \mathbb{R}^D$, this identity becomes the well known change-of-variable formula

$$p_h(z) = p\big(h^{-1}(z)\big)\left|\det\left(\frac{\partial h^{-1}(z)}{\partial z^T}\right)\right|, \tag{2.44}$$

often expressed in terms of x

$$p_h(z) = p_h\big(h(x)\big) = p(x)\left|\det\left(\frac{\partial h(x)}{\partial x^T}\right)\right|^{-1}, \tag{2.45}$$

where $\frac{\partial h(x)}{\partial x^T}$ is the Jacobian matrix of h:

$$\frac{\partial h(x)}{\partial x^T} = \begin{bmatrix} \frac{\partial h_1(x)}{\partial x_1} & \dots & \frac{\partial h_1(x)}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_D(x)}{\partial x_1} & \dots & \frac{\partial h_D(x)}{\partial x_D} \end{bmatrix}. \tag{2.46}$$

Intuitively, due to the absolute continuity of h, the Jacobian matrix describes the *local* behavior of h perfectly via a linear approximation. Its absolute determinant then captures by how much the D-dimensional hypervolume is scaled. Since $p(x)$ is in units of *inverse* D-dimensional hypervolume, it needs to be scaled by the reciprocal of the above scale factor.

2.6 Sampling from Probability Density Functions

It is often useful to generate data points that follow a certain prescribed PDF. For many seemingly simple PDFs, this operation is surprisingly challenging. In this section, we describe a few techniques that are frequently used in practice and discuss their individual benefits and drawbacks; Table 2.2 summarizes these. All the following techniques are based on an underlying "source of randomness" that permits drawing samples from the canonical uniform distribution $X \sim \mathcal{U}(0,1)$. Usually, this source of randomness is implemented directly on top of a pseudo-random number generator; the Mersenne Twister [Matsumoto and Nishimura 1998] and, more lately, the PCG family [O'Neill 2014] are popular choices due to their high performance and reasonable statistical qualities.

2.6.1 The Inversion Method

The inversion method is applicable to PDFs that are defined on the real number line $\mathbb{R}$. It can also be used on PDFs that are defined on *subsets* of $\mathbb{R}$, i.e. $\mathcal{S} \in \mathbb{R}$, by expressing such PDFs over the entirety of $\mathbb{R}$ and setting their value to zero wherever they were originally not defined, i.e. $x \in (\mathbb{R} - \mathcal{S}) \implies p(x) = 0$.

The inversion method works as follows: given a desired PDF $p(x)$, compute its CDF $P(x)$ via integration

$$P(x) = \int_{-\infty}^{x} p(t)\,\mathrm{d}t\,, \tag{2.47}$$

then invert the CDF to obtain $x = P^{-1}(\xi)$. Passing a uniformly distributed $\xi \sim \mathcal{U}(0,1)$ (which can be obtained from (pseudo)random number generators as mentioned before) into $X = P^{-1}(\xi)$ then produces X that are distributed proportional to the prescribed $p(x)$.

Using the formulas from Section 2.5, it is simple to show that the above steps—the inversion method—indeed produce $X \sim p(x)$. Let $\xi \sim \mathcal{U}(0,1)$ and $x = h(\xi) = P^{-1}(\xi)$, then

$$p_h(x) = p_{\mathcal{U}}(\xi)\left|\frac{\partial h(\xi)}{\partial \xi}\right|^{-1} = 1\left|\frac{\partial P(x)}{\partial x}\right| = \frac{\partial P(x)}{\partial x} = p(x)\,. \tag{2.48}$$

Although the inversion method is an invaluable tool for sampling from desired PDFs, it is only useful in a limited number of settings. For the inversion method to be useful, it must be possible to evaluate the inverse

CDF P^{-1}. In most cases, this amounts to being able to—either numerically or analytically—integrate and invert the desired density $p(x)$, which for many densities is not easy.

To given an example: evaluating the inverse CDF of the normal distribution involves evaluating the inverse error function, which can not be done in closed form. Although numeric approximations of the inverse error function exist, alternative techniques for sampling from the normal distribution—such as the Box-Muller transform [Box and Muller 1958]—are often preferred.

Another limitation of the inversion method is that it only works for 1-dimensional PDFs. This does not mean, however, that the inversion method is useless for higher-dimensional PDFs. If a higher-dimensional PDF can be conveniently expressed as the product of 1-dimensional *conditional* PDFs, then the inversion method can be applied to sample from *each of the conditional PDFs* in sequence. For example, consider $x = (x_1 \dots x_D) \in \mathbb{R}^D$ with the goal of sampling from $p(x)$. Then, if $p(x)$ can be expressed as $p(x_1)p(x_2|x_1) \cdots p(x_D|x_{D-1} \dots x_1)$, the inversion method can be used to sample $p(x_1)$, then $p(x_2|x_1)$, and so on.

Example Use Case: Uniform Distribution. A simple application of the inversion method is the sampling of the uniform distribution over arbitrary intervals $[a, b]$. Using the inverse of the uniform CDF (2.36) directly results in $X = \xi(b - a) + a$. Since the multi-dimensional uniform PDF is by definition the product of 1-dimensional uniform PDFs, one can sample from it by independently sampling its individual dimensions according to the same formula.

2.6.2 Rejection Sampling

"Rejection sampling" hinges on the observation that it is possible to transform existing samples from one PDF $\hat{p}(x)$ to a smaller number of samples that follow another PDF $p(x)$ by probabilistically discarding some of them. The discarded samples are referred to as "rejected" whereas all other samples are "accepted".

If the goal is to produce samples according to a given PDF $p(x)$, then $\hat{p}(x)$ must be proportional to a majorant $\hat{f}(x)$ that bounds $p(x)$ from above.

$$\hat{p} \propto \hat{f}\,, \tag{2.49}$$

$$\hat{f}(x) \geq p(x)\,, \quad \forall x \in \mathcal{D}\,. \tag{2.50}$$

Algorithm 2.1: Rejection-sampling. Samples from $p(x)$ by probabilistically accepting and rejecting samples from a tractable $\hat{p}(x) \propto \hat{f}(x)$.

```
function rejectionSample(p, p̂, f̂):
    repeat
        x ← sample(p̂)                          // Generate proposal x
        ρ ← p(x)/f̂(x)                  // Compute acceptance probability
        ξ ← sample(U(0,1))
    until ξ < ρ
    return x                                                  // Accept
```

The rejection-sampling algorithm to draw from $X \sim p$ then works as follows: draw x from $X \sim \hat{p}$ and evaluate $\rho(x) = p(x)/\hat{f}(x)$; with probability $\rho(x)$ return x; otherwise repeat. This algorithm is formalized in Algorithm 2.1.

The correctness of the algorithm can be proven by evaluating the probability density of accepted samples. Since all accepted samples are drawn independently from the same distribution (regardless of how many rejected samples they are preceeded by), we only have to show $\hat{p}(x \mid x \text{ accepted}) = p(x)$.

Proof.

$$\begin{aligned}
\hat{p}(x \mid x \text{ accepted}) &= \frac{\hat{p}(x)\rho(x)}{\int_{\mathcal{D}} \hat{p}(x)\rho(x)\,\mathrm{d}\mu(x)} && \text{(def. of conditional)} \\
&= \frac{\hat{p}(x)}{\hat{f}(x)} \cdot \frac{p(x)}{\int_{\mathcal{D}} p(x)\,\hat{p}(x)/\hat{f}(x)\,\mathrm{d}\mu(x)} && \text{(def. of } \rho\text{)} \\
&= c \cdot \frac{p(x)}{\int_{\mathcal{D}} c \cdot p(x)\,\mathrm{d}\mu(x)} && (\hat{p} \propto \hat{f}) \\
&= p(x) && \square
\end{aligned}$$

It is often simple to find a convenient majorant $\hat{f}(x)$ and respective sampleable $\hat{p}(x)$ for a given $p(x)$. In particular, when $\mathcal{D}$ is bounded (e.g. a finite interval on $\mathbb{R}$), then uniform $\hat{f}(x)$ is often a reasonable choice. However, the runtime of the rejection sampling algorithm heavily depends on the tightness of $\hat{f}(x)$. If $\hat{f}(x)$ is too loose, then rejection sampling can be arbitrarily slow due to a small $\rho(x)$ causing most samples to be rejected.

To make matters worse, many use cases that require random sample generation require a predictable number of consumed (pseudo)random numbers, which rejection sampling inherently can not guarantee due to *probabilistic* rejections. Examples of applications that require a predictable number of consumed samples are mission-critical software with tight run-time bounds and

Table 2.2: *Tradeoffs between sampling strategies. Although the Metropolis-Hastings algorithm can be used to sample from any PDF, the resulting samples are (often highly) correlated and require a burn-in period to diminish bias. When this is undesirable, rejection sampling and the inversion method are attractive alternatives.*

Name	Admissible PDFs	Independent	Bounded cost
Markov chain	All		✓
Rejection sampling	Bounded by sampleable majorant	✓	
Inversion method	Product of 1-dimensional conditionals with tractable inverse CDF	✓	✓

quasi-Monte Carlo algorithms. Nonetheless, rejection sampling is a valuable tool with pareto-optimal tradeoffs in many situations.

2.6.3 The Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm [Hastings 1970] can be utilized to sample from almost any desired PDF. Remarkably, this is possible, even if the desired PDF is only known in unnormalized form (i.e. up to an arbitrary constant factor). Metropolis-Hastings is built on so-called "Markov chains".

Markov Chains. Markov chains are defined by a set of states $\mathcal{D}$, a current state $x \in \mathcal{D}$, and a transition probability measure $P(S|x)$ that determines the probability of the current state advancing to any other state. In Markov chains, the transition probability only depends on the current state[4] x; we illustrate an example in Figure 2.3, where the numbers indicate the transition probability from every state to every other state. We are interested in a specific kind of Markov chains: those that are "ergodic". Ergodic Markov chains never get "stuck" in any particular state (i.e. every state must be reachable from any other state with finite probability in a finite number of

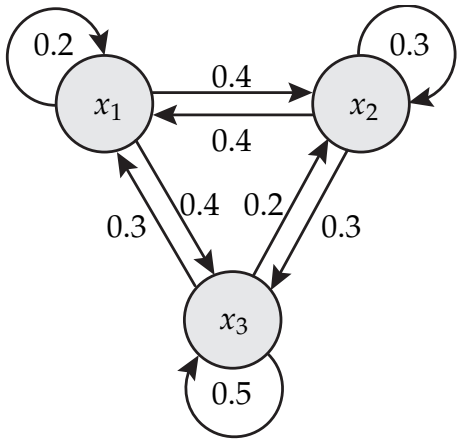


Figure 2.3: *3-state Markov chain.*

[4]Some Markov-chain definitions also allow $P(S|x)$ to depend on the number of state transitions that happened in the past. We are only interested in Markov chains that are *independent* of this number, which are referred to as "stationary" Markov chains in that context.

Algorithm 2.2: Metropolis-Hastings. Samples from $p(x) \propto f(x)$ by mutating the previous sample x and probabilistically accepting or rejecting the result.

```
function metropolisHastingsSample(f, p̂, x):
    x' ← sample( p̂( · |x) )              // Generate proposal x' by mutating x
    ρ ← min(1, f(x')p̂(x|x') / f(x)p̂(x'|x))    // Compute acceptance probability
    ξ ← sample( U(0,1) )
    if ξ < ρ then
        return x'                                              // Accept
    else
        return x                                               // Reject
```

transitions) and (informally) are free from deterministic cyclic behavior. We are interested in one particular useful property of ergodic Markov chains: the marginal probability distribution of the state of an ergodic Markov chain converges to a *unique* distribution that is independent from the initial state and only depends on the transition probability measure. This distribution is called the Markov chain's "stationary distribution".

Metropolis-Hastings. The Metropolis-Hastings algorithm [Hastings 1970] is a recipe for simulating a Markov chain that has a stationary distribution with desired PDF $p(x)$ that is only known up to a constant of proportionality $f(x) \propto p(x)$. The algorithm works by repeatedly mutating the current state x to obtain x' and probabilistically replacing x with x' (called "acceptance"); see Algorithm 2.2. Mutations are performed according to a "proposal PDF" $\hat{p}(x'|x)$ that can be sampled from and may (but need not) depend on x. The sequence of generated x's, when using the acceptance probability

$$\rho(x', x) = \min\left(1, \frac{f(x')\hat{p}(x|x')}{f(x)\hat{p}(x'|x)}\right), \tag{2.51}$$

has the desired stationary PDF $p(x)$.

Unfortunately, despite of the applicability of Metropolis-Hastings to a wide range of functions, there are a number of disadvantages associated with the algorithm. First, even though the underlying Markov chain has the desired stationary PDF $p(x)$, the *actual* distribution of states only *converges* to the stationary PDF as the sample count approaches infinity and is therefore always slightly incorrect. In the application that interests us, Monte Carlo integration for light-transport simulation, this limitation can thankfully be overcome by careful sample weighting [Veach and Guibas 1997]. The second limitation is, however, more problematic for us: the samples produced by

the Metropolis-Hastings algorithm are positively correlated with each other. This is an undesirable property, because many applications, including Monte Carlo integration, have a preference for independent (or negatively correlated) samples that cover the domain more uniformly. Although the positive correlation of Metropolis-Hastings can be reduced by choosing $\hat{p}(x'|x)$ that approximates $p(x')$ as much as possible, doing so in an effective manner is a challenging open research problem.

CHAPTER 3

Monte Carlo Integration

Monte Carlo integration is useful in any application that requires the evaluation of integrals for which closed-form solutions either do not exist or are difficult to obtain. In this dissertation, our goal is the efficient synthesis of photorealistic images by means of solving the path integral. To this end, Monte Carlo integration is our method of choice. In this chapter, we provide a brief overview over the practice of Monte Carlo integration. Although we focus on the aspects of the technique that will become relevant later in this dissertation, we attempt to explain in more generality whenever it is reasonable.

3.1 Origin of Monte Carlo

The Monte Carlo method was originally conceived to solve a problem not so different from light transport: neutron transport. In the 20th century, physicists working on the hydrogen bomb in Los Alamos were unable to predict the propagation of neutrons with traditional analytical methods. Stanisław Ulam, one of these physicists, was inspired to approach the problem numerically via repeated random sampling by the then recently completed ENIAC computer. In 1946, Ulam proposed this idea to his colleague John von Neumann, which led to the development of the Monte Carlo method [Metropolis 1987]. The name "Monte Carlo" was coined by Nicholas C. Metropolis, who co-published the first declassified document on the method [Metropolis and Ulam 1949], after Ulam's Uncle's habit to gamble in the Monacan city "Monte Carlo".

Interestingly, physicist Enrico Fermi independently discovered the Monte Carlo method around 15 years earlier, which he used to astonish his then-colleagues with unbelievably accurate predictions of experimental results [Metropolis 1987]. However, at that point in time, he elected not to publish the technique.

3.2 Definition and Basic Properties

Given an integral over domain $\mathcal{D}$ with measure μ

$$F = \int_{\mathcal{D}} f(x) \, \mathrm{d}\mu(x) \,, \tag{3.1}$$

the idea of Monte Carlo integration is to express the integral as an expectation over some probability density $q(x)$

$$F = \int_{\mathcal{D}} \frac{f(x)}{q(x)} \, q(x) \, \mathrm{d}\mu(x) = \mathbb{E}\left[\frac{f(X)}{q(X)}\right] , \tag{3.2}$$

which is valid if $q(x)$ is non-zero wherever $f(x)$ is non-zero, i.e. $f(x) \neq 0 \implies q(x) > 0, \forall x \in \mathcal{D}$. The expectation and therefore the integral can then be estimated as a sum of random variables, which itself is a random variable

$$F = \mathbb{E}\left[\frac{f(X)}{q(X)}\right] \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i)} = \langle F \rangle_N \,, \tag{3.3}$$

where $X_1, \ldots, X_N \in \mathcal{D}$ are independent and distributed identically proportional to $q(x)$. The strong law of large numbers tell us, that $\langle F \rangle_N$ converges to F as N approaches infinity. We call the random variable $\langle F \rangle_N$ a "Monte Carlo estimator" of F and any particular *realization* of the random variable a "Monte Carlo estimate". We further define $\langle F \rangle = \langle F \rangle_1 = \frac{f(X)}{q(X)}$ as the "primal Monte Carlo estimator", which is notationally convenient in derivations that do not depend on N.

Variance of Monte Carlo Estimators. The variance of any Monte Carlo estimator is proportional to $1/N$, regardless of the choice of $f(x)$ or $q(x)$.[1]

[1] Certain pathological Monte Carlo estimators admit infinite variance, in which case it is independent of the number of samples N. In this dissertation, we ignore the existence of such estimators and always assume finite variance.

Proof.

$$\begin{aligned}
\mathbb{V}\left[\langle F\rangle_N\right] &= \mathbb{V}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] && \text{(definition)}\\
&= \frac{1}{N^2}\mathbb{V}\left[\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] && \text{(property of variance)}\\
&= \frac{1}{N^2}\sum_{i=1}^{N}\mathbb{V}\left[\frac{f(X_i)}{p(X_i)}\right] && (X_1,\ldots,X_N \text{ independent})\\
&= \frac{1}{N}\mathbb{V}\left[\langle F\rangle\right] && (X,\ldots,X_N \text{ same distribution})
\end{aligned}$$ □

This means: to half the variance of a given Monte Carlo estimator—also known as "mean squared error" (MSE) if the estimator is unbiased—the number of random samples N needs to be doubled. Similarly, to half the "standard deviation" $\sqrt{\mathbb{V}\left[\langle F\rangle_N\right]}$—the "root mean squared error" (RMSE) if the estimator is unbiased—the number of random samples N must be *quadrupled*. This convergence rate of $\mathcal{O}(1/\sqrt{N})$ is commonly considered to be both a blessing *and* a curse at the same time. It is a blessing, because, unlike many numerical quadrature rules, it does not suffer from the curse of dimensionality. That is, the convergence rate is the same, regardless of the dimensionality of $\mathcal{D}$. We will see later, that computing light transport involves solving a recursive integral, making the problem infinitely-dimensional in its most general form. Therefore, a reasonable convergence rate for large numbers of dimensions is certainly desired.

The convergence rate is also widely considered to be a curse, because it is relatively slow. The variance being proportional to $1/N$ means, that *if* the variance of the primal Monte Carlo estimator $\langle F\rangle$ is large, then to achieve a desired variance for which we consider the Monte Carlo estimate to be "accurate", a large number of samples N needs to be drawn (proportional to $\mathbb{V}\left[\langle F\rangle\right]$), and therefore the total computation time—being proportional to the number of samples N—is equally large.

This leads to the question whether the variance can be reduced by other means than simply drawing a larger number of samples. This is indeed possible; in the following section, we explore several approaches to do so.

3.3 Variance Reduction

This section provides a brief overview over Monte Carlo variance-reduction techniques that are used and/or discussed within this dissertation. In partic-

ular, we will discuss analytic integration, importance sampling, and control variates.

There exist several other variance-reduction techniques that we omit for brevity. For a more thorough introduction to variance-reduction techniques we refer the interested reader to Eric Veach's excellent thesis [Veach 1997].

3.3.1 Use of Expectation

Perhaps the most straightforward approach to reducing variance is the explicit integration over some dimensions of the integration domain and then performing Monte Carlo integration only over the remaining dimensions. For example, consider the following Monte Carlo estimator of an integral over the domain $\mathcal{D} = \mathbb{R}^2$:

$$\langle F \rangle = \frac{f(X,Y)}{q(X,Y)}\,. \tag{3.4}$$

Integrating over some dimensions of the domain in *both* the numerator *and* the denominator (for example along Y) yields another Monte Carlo estimator of the same integral: it is the conditional expectation of the original estimator with respect to the integrated-out dimensions

$$\langle F \rangle' = \frac{\int_{\mathbb{R}} f(X,y)\,\mathrm{d}y}{\int_{\mathbb{R}} q(X,y)\,\mathrm{d}y} = \int_{\mathbb{R}} \frac{f(X,y)}{q(X,y)} q(y|X)\,\mathrm{d}y = \mathbb{E}_Y\left[\frac{f(X,Y)}{q(X,Y)}\right]. \tag{3.5}$$

The new Monte Carlo estimator $\langle F \rangle'$ therefore has the same expectation as $\langle F \rangle$ (i.e. preserves unbiasedness) but has lower variance, because variation caused by some random variables (in this case Y) has been integrated out.

This intuitive notion has been formalized in a series of works by Rao, Blackwell, and Kolmogorov as the Rao–Blackwell–Kolmogorov theorem. The use of expectation is therefore also often referred to as Rao–Blackwellization.

3.3.2 Importance Sampling

Importance sampling is one of the most well known variance reduction techniques that exist. The key insight is, that the variance of $\langle F \rangle_N$ heavily depends on the choice of $q(x)$. The closer the shape of $q(x)$ follows the shape of normalized $f(x)$ (which is $p(x) = f(x)/F$), the lower the variance becomes. Conversely, if $q(x)$ deviates from the shape of $p(x)$, then variance increases. In the limiting case of $q(x) = p(x), \forall x \in \mathcal{D}$ the variance becomes

zero, i.e. the Monte Carlo estimator produces the correct answer with a single sample:

$$\langle F \rangle = \frac{f(X)}{p(X)} = \frac{f(X)}{f(X)/F} = F\,. \tag{3.6}$$

Unfortunately, knowing $p(x)$ requires knowledge of the integral F which we are trying to solve in the first place, so we can not use $p(x)$ directly in practice. However, it is often possible to reduce variance by sampling from a distribution that either approximates the shape of $p(x)$ in general, or matches it well only in some regions of the integration domain $\mathcal{D}$.

More precisely, given some sampling density $q(x)$, the variance can be expressed as a function of $p(x)$ and $q(x)$:

$$\begin{aligned}
\mathbb{V}[\langle F \rangle] &= \mathbb{E}\left[\frac{f(X)^2}{q(X)^2}\right] - \mathbb{E}\left[\frac{f(X)}{q(X)}\right]^2 \\
&= \int_{\mathcal{D}} f(x)\frac{f(x)}{q(x)}\,\mathrm{d}\mu(x) - F^2 \\
&= F^2\left(\int_{\mathcal{D}} p(x)\frac{p(x)}{q(x)}\,\mathrm{d}\mu(x) - 1\right).
\end{aligned} \tag{3.7}$$

This formula not only illustrates once more that $\mathbb{V}[\langle F \rangle]$ approaches zero as $q(x)$ approaches $p(x)$, but it also highlights the failure case: whenever $q(x)$ approaches zero in a region where $p(x) \neq 0$, the variance tends to infinity. In this context it helps to think of the integral in the parentheses as either the expected ratio $p(X)/q(X)$ for $X \sim p$ or, alternatively, as the expected *squared* ratio $p(X)^2/q(X)^2$ for $X \sim q$.

In summary, variance can be reduced by drawing Monte Carlo samples from a probability density $q(x)$ that is *as proportional as possible* to the integrand $f(x)$.

3.3.3 Quantifying the Quality of a Sampling Distribution

In Section 2.3 we briefly introduced the concept of "divergences" for quantifying the difference between a given pair of probability distributions. Since for all divergences $D(p \,\|\, q) = 0$ if and only if $p = q$, a divergence of zero is both a necessary and sufficient condition for q being *optimal* for importance sampling, i.e. resulting in a variance of zero.

However, beyond this basic property it can be desirable for $D(p \,\|\, q)$ to quantify the "efficiency" of the Monte Carlo estimator $\langle F \rangle$. One particularly

interesting f-divergence in this context is the Kullback-Leibler (KL) divergence:

$$D_{\mathrm{KL}}(p \,\|\, q) = \int_{\mathcal{D}} p(x) \log \frac{p(x)}{q(x)} \,\mathrm{d}\mu(x)\,. \tag{3.8}$$

It is also known as the "relative entropy" between p and q, since it can be written as the difference between the cross entropy $H(p, q)$ and the entropy $E(p)$

$$\begin{aligned}\int_{\mathcal{D}} p(x) \log \frac{p(x)}{q(x)} \,\mathrm{d}x &= \underbrace{-\int_{\mathcal{D}} p(x) \log q(x) \,\mathrm{d}x}_{\text{Cross entropy}} + \underbrace{\int_{\mathcal{D}} p(x) \log p(x) \,\mathrm{d}x}_{\text{Negative entropy}} \\ &= H(p, q) - E(p)\,. \end{aligned} \tag{3.9}$$

Information-theoretically, the KL-divergence therefore measures the *wasted* amount of information when sub-optimally encoding samples from p with the optimal encoding of q. While this view is very abstract in the context of Monte Carlo estimation, one can also analyze the terms from a numerical standpoint: the ratio $p(x)/q(x)$ ensures, that the divergence approaches infinity as $q(x)$ approaches zero where $p(x) \neq 0$, corresponding to the behavior of the variance of a Monte Carlo estimator of F that is importance sampled by $q(x)$. The logarithm around the ratio can be viewed as making the divergence more numerically stable by suppressing values close to infinity or zero. These properties make the KL-divergence interesting for measuring how "good" any given q is for importance sampling in a Monte Carlo context.

Anothing interesting divergence in the context of Monte Carlo integration is the Pearson χ^2-divergence

$$D_{\chi^2}(p \,\|\, q) = \int_{\mathcal{D}} \frac{p(x)^2}{q(x)} \,\mathrm{d}\mu(x) - 1\,, \tag{3.10}$$

which is proportional to the variance of $\langle F \rangle$ by a factor of $\frac{1}{F^2}$ (see Equation 3.7). One might think that this close correspondence to the variance makes the χ^2-divergence the obvious divergence of choice for measuring the quality of q, but when estimating the χ^2-divergence in practice (there typically exists no closed-form), numerical instabilities may arise due to the squared ratio $p(x)^2/q(x)^2$. This makes the robust usage of the χ^2-divergence in practical algorithms difficult, especially when it drives a machine-learning optimization objective. In fact, we found the KL-divergence to yield superior usability in all our experiments (see Chapter 8).

3.3.4 Multiple Importance Sampling

Often times, a collection of candidate densities $q_i(x), i \in \mathbb{N}$ exists, where each candidate is a good choice for importance sampling a *different, potentially overlapping* region of the integration domain $\mathcal{D}$. In other words, multiple densities may exist, each producing Monte Carlo estimators with low variance in *some* parts of $\mathcal{D}$ and (potentially) high variance in others. "Multiple importance sampling" is a recipe for combining the strengths of these densities $q_i(x)$ to achieve provably low variance in those regions where at least one of the densities results in low variance.

To motivate the benefits of multiple importance sampling, let us first consider the naïve approach of simply averaging multiple independent Monte Carlo estimators $\langle F \rangle_N^i, i \in \mathbb{N}$, each using the corresponding density $q_i(x)$ for importance sampling. Given several uncorrelated random variables, the variance of their sum is equal to the sum of their variances, and thus

$$\mathbb{V}\left[\frac{1}{M}\sum_{i=1}^{M}\langle F\rangle_N^i\right] = \sum_{i=1}^{M}\mathbb{V}\left[\frac{1}{M}\langle F\rangle_N^i\right] \geq \frac{1}{M}\mathbb{V}\left[\langle F\rangle_N^{\hat{i}}\right] = \mathbb{V}\left[\langle F\rangle_{MN}^{\hat{i}}\right], \tag{3.11}$$

where $\langle F \rangle_N^{\hat{i}}$ is the estimator corresponding to the density that produces the lowest variance. This formula says, that combining M strategies never produces lower variance than choosing the most optimal of the strategies and instead investing the same total sample count MN into it. In other words: given a fixed total number of samples, a linear combination of estimators can never outperform the best available one, even if that overall "best" available estimator performs much worse in some regions of the integration domain than the other estimators.

To get around this limitation, the key idea behind multiple importance sampling is to combine *the individual sample estimates themselves* rather than their means [Veach and Guibas 1995]. Consider the general case of linearly combining the samples of M strategies

$$\langle F\rangle_N^M = \sum_{i=1}^{N}\sum_{j=1}^{M} w_j(X_{i,j})\frac{f(X_{i,j})}{q_j(X_{i,j})} \tag{3.12}$$

via sample- and strategy dependent weights $w_j(x)$. This estimator is unbiased under the mild conditions that the weights sum up to 1 whenever $f(x)$ is non-zero

$$\sum_{j=1}^{M} w_j(x) = 1, \quad \forall x \in \{x \in \mathcal{D} \mid f(x) \neq 0\} \tag{3.13}$$

and the weight of strategy j is zero if the PDF of said strategy is zero

$$w_j(x) = 0, \quad \forall x \in \{x \in \mathcal{D} \mid q_j(x) = 0\}. \tag{3.14}$$

Given these conditions, unbiasedness follows.

Proof.

$$\begin{aligned}
\mathbb{E}[\langle F \rangle_N^M] &= \sum_{j=1}^{M} \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} w_j(X_{i,j})\frac{f(X_{i,j})}{q_j(X_{i,j})}\right] \\
&= \sum_{j=1}^{M} \int_{\mathcal{D}} w_j(x)\frac{f(x)}{q_j(x)} q_j(x)\,\mathrm{d}\mu(x) \\
&= \sum_{j=1}^{M} \int_{\mathcal{D}} w_j(x) f(x)\,\mathrm{d}\mu(x) \\
&= \int_{\mathcal{D}} \sum_{j=1}^{M} w_j(x) f(x)\,\mathrm{d}\mu(x) \\
&= \int_{\mathcal{D}} f(x)\,\mathrm{d}\mu(x)
\end{aligned}$$

□

In the multiple-importance-sampling framework one particular choice of weights stands out: the so-called balance heuristic.

$$w_j(x) = \frac{q_j(x)}{\sum_{i=1}^{M} q_i(x)}. \tag{3.15}$$

It performs *provably* well in the sense that any other combination strategy can not significantly reduce the variance further: let $\langle \hat{F} \rangle_N^M$ be a multiple-importance-sampling estimator that uses the balance heuristic and let $\langle F \rangle_N^M$ be *any* other multiple-importance-sampling estimator, then

$$\mathbb{V}[\langle \hat{F} \rangle_N^M] - \mathbb{V}[\langle F \rangle_N^M] \leq \frac{M-1}{MN} F^2. \tag{3.16}$$

For a proof and more thorough treatment of multiple importance sampling, we refer to the paper by Veach and Guibas [1995].

3.3.5 Control Variates

The framework of control variates requires approximating $f(x)$ with a function $g(x)$ with a *known* integral $G = \int_{\mathcal{D}} g(x)\,\mathrm{d}\mu(x)$. The integral F we would

like to solve for can then be rewritten as

$$F = G + F - G \tag{3.17}$$
$$= G + \int_{\mathcal{D}} f(x)\,\mathrm{d}\mu(x) - \int_{\mathcal{D}} g(x)\,\mathrm{d}\mu(x) \tag{3.18}$$
$$= G + \int_{\mathcal{D}} f(x) - g(x)\,\mathrm{d}\mu(x)\,, \tag{3.19}$$

and Monte Carlo integration can be performed *only* on the *difference* $f(x) - g(x)$ rather than $f(x)$ alone. Just like importance sampling, we can easily show that $g(x) = f(x), \forall x \in \mathcal{D}$ results in a zero-variance estimator

$$\mathbb{V}\left[\frac{f(X) - g(X)}{q(X)}\right] = \mathbb{V}\left[\frac{f(X) - f(X)}{q(X)}\right] = \mathbb{V}[0] = 0\,. \tag{3.20}$$

Let us investigate in more detail how the variance relates to the choice of $g(x)$:

$$\mathbb{V}\left[\frac{f(X) - g(X)}{q(X)}\right] = \mathbb{E}\left[\frac{\big(f(X) - g(X)\big)^2}{q(X)^2}\right] - \mathbb{E}\left[\frac{f(X) - g(X)}{q(X)}\right]^2 \tag{3.21}$$
$$= \int_{\mathcal{D}} \frac{\big(f(x) - g(x)\big)^2}{q(x)}\,\mathrm{d}\mu(x) - (F - G)^2\,. \tag{3.22}$$

The first important insight is, that the method of control variates is *orthogonal* to that of importance sampling. Regardless of the choice of $g(x)$, one can still reduce the variance arbitrarily much by choosing $q(x)$ to be as proportional to the integrand (in this case $f(x) - g(x)$) as possible.

However, secondly, the variance depends on the *mean squared difference* (weighted by $1/q(x)$) between $f(x)$ and $g(x)$. Note, though, that $f(x)$ does not necessarily have to equal $g(x)$ for the variance to reach zero; it is sufficient for $\big(f(x) - g(x)\big)^2/q(x)$ to be a constant.

This nicely illustrates how the variance characteristics of the method of control variates differ from those of importance sampling. While importance sampling reduces variance by *sampling as proportional as possible* to $f(x)$, control variates reduce variance when using a function $g(x)$ with a known integral G, which is as close as possible to $f(x)$ *in a weighted-least-squares sense*.

3.3.6 Viewing Variance Reduction as a Smoothing Operation

Let us interpret a Monte Carlo estimator $\langle F \rangle$ of F not as a random variable but as a deterministic function of x

$$\langle F \rangle(x) = G + \frac{f(x) - g(x)}{q(x)}\,. \tag{3.23}$$

It is then worthwhile to view the application of control variates and (multiple) importance sampling as picking $g(x)$ and $q(x)$ such that $\langle F \rangle(x)$ becomes *maximally smooth* (i.e. constant). Indeed, minimizing $\mathbb{V}[\langle F \rangle(X)]$ is nothing more than a formalization of smoothing $\langle F \rangle(x)$.

Although this view may seem trivial at first, it becomes interesting in certain situations. In particular, consider the case, where the random variable $X \sim q$ is a function of another random variable $Z \sim p$, where $X = x(Z)$. This setup may seem contrived at first glance, but it occurs frequently in practice when using one of the techniques described in Section 2.6 to sample from a desired PDF $q(x)$.

For example: when using the inversion method. Let $Z \sim \mathcal{U}(0,1)$ be uniformly distributed and $x(Z) = Q^{-1}(Z)$ be the inverse CDF of $q(x)$. Then, let us rewrite the integral F in terms of z

$$\begin{aligned} F &= \mathbb{E}\big[\langle F \rangle(X)\big] \\ &= \mathbb{E}\big[\langle F \rangle(x(Z))\big] \\ &= G + \int_{\mathcal{D}_z} \Big(f\big(x(z)\big) - g\big(x(z)\big)\Big) \frac{p(z)}{q\big(x(z)\big)} \,\mathrm{d}\mu(z) \\ &= G + \int_{\mathcal{D}_z} f_z(z) \,\mathrm{d}\mu(z)\,. \end{aligned} \tag{3.24}$$

Assuming that $g(x)$ and $q(x)$ were chosen such that they reduce the variance of $\langle F \rangle(x)$, the above formula is a *reformulation* of the original integrand in a smoother parameterization $f_z(z)$. One possible application of this view is the recursive usage of variance reduction techniques on estimating $\int_{\mathcal{D}_z} f_z(z) \,\mathrm{d}\mu(z)$ in addition to the ones that were originally applied to $\int_{\mathcal{D}} f(x) \,\mathrm{d}\mu(x)$.

3.4 Properties of Estimators

Any statistical estimator—not just a Monte Carlo estimator—has certain properties that may or may not be desirable. In this section we describe two particular properties that are widely regarded as desirable in Monte Carlo integration: consistency and unbiasedness.

3.4.1 Consistency

A consistent estimator converges to the correct answer as the number of samples approaches infinity. Let E_N be an N-sample estimator of some true

value E, then E_N is consistent if and only if it approaches E as N approaches infinity

$$\lim_{N\to\infty} \text{Prob}\Big(|E_N - E| > \epsilon \Big) = 0\,. \tag{3.25}$$

In the case of the Monte Carlo estimators we discussed before, the weak law of large numbers guarantees consistency out-of-the-box.

However, in practice it is often desirable to approximate F with a simpler-to-solve integral F', for instance, when applying level-of-detail approaches to computer graphics problems. In this case, a Monte Carlo estimator of F' would be consistent w.r.t. F', but not with respect to F.

3.4.2 Unbiasedness

The bias of an estimator E_N is the deviation of its expectation $\mathbb{E}[E_N]$ from the true value E:

$$\text{Bias}[E_N] = \mathbb{E}[E_N] - E\,. \tag{3.26}$$

It can be understood as *systematic* error that does not vanish when averaging multiple independent estimates. In the particular example that we gave in the preceeding section, the bias of estimator $\langle F' \rangle$ when using it to estimate F would be the difference between F' and F

$$\text{Bias}[\langle F' \rangle] = \mathbb{E}[\langle F' \rangle] - F = F' - F\,, \tag{3.27}$$

that is, the approximation error of F' versus F.

An estimator is called *unbiased* if its bias is zero, i.e. its expectation matches the true value. This leads to an interesting connection between unbiasedness and consistency: the bias of any consistent estimator must vanish as N approaches infinity. However, the reverse is not always true. An unbiased estimator is *not* necessarily consistent, because unbiasedness does not guarantee, that the estimator converges in probability to any value at all as N approaches infinity. In the case of Monte Carlo estimators, convergence is guaranteed due to the strong law of large numbers and unbiasedness *does* imply consistency.

3.5 Nested Integrals

It is sometimes required to solve not a single integral but a nested sequence of integrals. Such nested integrals are common when solving transport

problems of the kind that we are interested in. Suppose we have $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2 \times \cdots \times \mathcal{D}_n$ and nested integrals over $\mathcal{D}_1, \ldots, \mathcal{D}_n$ of the form

$$\begin{aligned} F &= \int_{\mathcal{D}_1} t_1(x_1) + f_1(x_1) \int_{\mathcal{D}_2} t_2(x_1, x_2) + f_2(x_1, x_2) \int_{\mathcal{D}_3} \ldots \, \mathrm{d}\mu(x_3) \, \mathrm{d}\mu(x_2) \, \mathrm{d}\mu(x_1) \\ &= \int_{\mathcal{D}} \sum_{i=1}^{n} \left(t_i(x_1, \ldots, x_i) \prod_{j=1}^{i-1} f_j(x_1, \ldots, x_j) \right) \prod_{i=1}^{n} \mathrm{d}\mu(x_i) \,. \end{aligned} \tag{3.28}$$

We can estimate these nested integrals using a single Monte Carlo estimator

$$\langle F \rangle = \frac{\sum_{i=1}^{n} t_i(X_1, \ldots, X_i) \prod_{j=1}^{i-1} f_j(X_1, \ldots, X_j)}{q(X_1, \ldots, X_n)} \,. \tag{3.29}$$

via the use of expectation (Section 3.3.1), we can replace this Monte Carlo estimator with a lower-variance estimator—a product-sum of nested 1-D Monte Carlo estimators—converging to the same value

$$\begin{aligned} \langle F \rangle' &= \sum_{i=1}^{n} \frac{t_i(X_1, \ldots, X_i) \prod_{j=1}^{i-1} f_j(X_1, \ldots, X_j)}{q(X_1, \ldots, X_i)} \\ &= \sum_{i=1}^{n} \frac{t_i(X_1, \ldots, X_i)}{q(X_i | X_1, \ldots, X_{i-1})} \prod_{j=1}^{i-1} \frac{f_j(X_1, \ldots, X_j)}{q(X_j | X_1, \ldots, X_{j-1})} \\ &= \sum_{i=1}^{n} \langle T_i \rangle \prod_{j=1}^{i-1} \langle F_j \rangle = \sum_{i=1}^{n} \langle T_i \rangle \langle P_i \rangle = \sum_{i=1}^{n} \langle S_i \rangle \,, \end{aligned} \tag{3.30}$$

where we defined $\langle P_i \rangle = \prod_{j=1}^{i-1} \langle F_j \rangle$ and $\langle S_i \rangle = \langle T_i \rangle \langle P_i \rangle$. Note that the computational cost of evaluating $\langle F \rangle'$ is *linear* in n (as opposed to quadratic) because each $\langle S_i \rangle$ shares its prefix of $\langle F_j \rangle$'s with all other $\langle S_i \rangle$.

Special care must be taken when choosing the sample count of the individual estimators to avoid a prohibitively expensive exponential growth in computational cost: due to the nesting, if N samples were used in each estimator $\langle T_i \rangle$ and $\langle F_i \rangle$, then the total sample count would be $\mathcal{O}(N^n)$.

Unfortunately, there is no single optimal allocation of samples to the nested estimates; the optimal allocation heavily depends on the nested integrands and the quality of their individual corresponding variance-reduction techniques. In practice, a common approach is simply to use primal estimators everywhere, except for the outermost estimator $\langle F \rangle'$.

3.5.1 Russian Roulette

Consider the case where there are *infinitely many* nested integrals, i.e. $n \to \infty$, leading to

$$\langle F \rangle' = \sum_{i=1}^{\infty} \langle S_i \rangle . \tag{3.31}$$

This situation may seem contrived at first glance, but it actually occurs frequently in practice. The specific application we are interested in is solving Fredholm integral equations of the second kind, which describe the equilibrium solutions of transport problems, including light transport.

Evaluating this Monte Carlo estimator requires an infinite amount of computation due to its infinitely many terms. However, using a technique called "russian roulette", it is possible to modify the above Monte Carlo estimator to achieve *finite* run time at the cost of additional variance without introducing bias.

The key idea is to replace each Monte Carlo estimator $\langle S_i \rangle$ with another one that is *zero* with some probability $b < 1$ and $\langle S_i \rangle / (1 - b)$ otherwise[2]. This can be expressed by introducing a uniformly distributed random variable $\xi \sim \mathcal{U}(0, 1)$:

$$\langle S_i \rangle' = \begin{cases} 0 & \text{if } \xi < b \\ \frac{\langle S_i \rangle}{1-b} & \text{otherwise.} \end{cases} . \tag{3.32}$$

It is easy to show that if $\langle S_i \rangle$ is an unbiased estimator, then $\langle S_i \rangle'$ is unbiased as well:

Proof.

$$\begin{aligned} \mathbb{E}\left[\langle S_i \rangle'\right] &= \mathbb{E}\left[\mathbb{E}_\xi\left[\langle S_i \rangle'\right]\right] && \text{(conditional expectation)} \\ &= \mathbb{E}\left[b \cdot 0 + (1-b)\frac{\langle S_i \rangle}{1-b}\right] && \text{(definition of } \langle S_i \rangle'\text{)} \\ &= S_i && \text{(}\langle S_i \rangle \text{ unbiased)} \end{aligned}$$

□

[2] This is essentially the *reverse* of variance reduction by the use of expection (Section 3.3.1).

We can also quantify the increased variance of $\langle S_i \rangle'$ when compared to $\langle S_i \rangle$:

$$\begin{aligned}
\mathbb{V}\left[\langle S_i \rangle'\right] &= \mathbb{E}\left[\mathbb{E}_\xi\left[\langle S_i \rangle'^2\right]\right] - S_i^2 \\
&= \mathbb{E}\left[b \cdot 0^2 + (1-b)\left(\frac{\langle S_i \rangle}{1-b}\right)^2\right] - S_i^2 \\
&= \frac{1}{1-b}\,\mathbb{E}\left[\langle S_i \rangle^2\right] - S_i^2 \\
&= \frac{b}{1-b}\,\mathbb{E}\left[\langle S_i \rangle^2\right] + \mathbb{V}\left[\langle S_i \rangle\right]. \qquad (3.33)
\end{aligned}$$

The russian-roulette approach to making the infinite sum $\langle F \rangle' = \sum_{i=1}^{\infty} \langle S_i \rangle$ tractable is then to replace all $\langle S_i \rangle$ with corresponding $\langle S_i \rangle'$ with appropriately chosen termination probabilities b_i that approach 1 sufficiently quickly.

In the absence of perfect information, the goal of obtaining russian-roulette probabilities b_i that optimally balance variance and computational cost is a difficult problem, although there exist data-driven approaches that *learn* approximately optimal russian-roulette probabilities [Vorba and Křivánek 2016]. Alternatively, a popular heuristic is choosing $b_i = 1 - \prod_{j=1}^{i-1} \langle F_i \rangle$, but its efficiency heavily depends on the characteristics of the individual estimators.

3.6 Overview of Advanced Techniques

Although, due to the limited scope of this dissertation, we can not cover every variant of Monte Carlo integration that was published in the past, we list several advanced techniques that we consider to be of high interest for our use case of path-traced Monte Carlo light-transport simulation.

3.6.1 Quasi Monte Carlo

Perhaps surprisingly, the error of Monte Carlo estimators can converge to zero even when driven by *deterministic* sequences of samples, rather than random sequences. These types of deterministic estimators are referred to as "quasi-Monte Carlo" estimators. Even though quasi-Monte Carlo estimators are inherently biased due to their determinism, when they are carefully designed they can converge at faster asymptotic rates than random Monte Carlo estimators. In this dissertation, we focus on image synthesis using random Monte Carlo estimators, which in some cases can be extended to the quasi-Monte Carlo setting with little effort if required.

Quasi Monte Carlo is quite popular in path tracing: most renderers support it in some form. We refer the interested reader to the summary of quasi-Monte Carlo methods for image synthesis by Keller [2013].

3.6.2 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) techniques draw samples from an *unnormalized* target PDF $f(x) = c \cdot p(x)$ using an underlying Markov chain, for example by using the Metropolis-Hastings algorithm (see Section 2.6.3 for a definition of Markov chains and the Metropolis-Hastings algorithm). Since, unlike traditional importance-sampling PDFs, the stationary PDF of a Markov chain can not be readily evaluated, the class of integrals that can be estimated using MCMC are limited to those of the form

$$H = \int_{\mathcal{D}} h(x)p(x)\,\mathrm{d}\mu(x) = \mathbb{E}\big[h(X)\big]\,, \tag{3.34}$$

because their corresponding Monte Carlo estimator

$$\langle H \rangle_N = \frac{1}{N}\sum_{i=1}^{N} h(X_i) \tag{3.35}$$

does not require evaluating $p(x)$. In other words: MCMC is applicable for estimating *expectations* over probability densities that are available only in unnormalized form.

Note, that if $p(x)$ is available in *normalized* form, that is, the Markov chain is used purely to draw samples from $p(x)$ and not to avoid normalization, then MCMC is also applicable in the general integration setting.

3.6.3 Adaptive Monte Carlo

The idea behind adaptive Monte Carlo is to gradually refine importance-sampling distributions based on the outcome of previously drawn Monte Carlo samples.

Sequential Monte Carlo. The need for adaptation initially arose in the context of *dynamic* systems where the integrand itself continually changes and therefore requires constantly updated sampling distributions. This led to "sequential Monte Carlo" [Liu and Chen 1998], also known as "particle filters" [Del Moral 1996].

Population Monte Carlo. More interesting to us, however, is the case of *static* integrands. Building on the observation of Chopin [2002] that sequential Monte Carlo becomes an iterated importance-sampling scheme in the static setting, Cappé et al. [2004] propose "population Monte Carlo" (PMC), which, like particle filters, simulates a finite number of particles to approximate the importance-sampling distribution. Several variations of PMC exist, for example M-PMC [Cappé et al. 2008] which models the importance-sampling distribution as a mixture of gaussians that are centered around the simulated particle positions.

Particle-less Methods. In parallel to the development of the above particle-based approaches, several attempts have been made to learning and refining importance-sampling distributions that are represented in different ways, such as piecewise-constant functions [Jensen 1995; Lafortune and Willems 1995], mixture models that resemble M-PMC [Hey and Purgathofer 2002; Vorba et al. 2014], and, lately, neural networks [Dahm and Keller 2017; Zheng and Zwicker 2018].

Our work in this dissertation falls into this category. We present adaptive Monte Carlo methods that use piecewise-constant functions in Chapters 6 and 7, and neural networks in Chapter 8.

CHAPTER 4

Light Transport in Computer Graphics

The complex appearance of images of the real world stems from the fact that light interacts with different materials in a wide variety of ways, resulting in a multitude of interesting visual phenomena. Light consists of discrete packages of energy that propagate through space as a particle and as a wave at the same time. These energy packages—photons—interact with matter and get absorbed or reflected, only to interact again with the next body of matter they encounter.

Due to the complexity of simulating the full underlying physics of light as dictated by quantum mechanics, computer graphics literature usually follows the simplified setting of geometric optics, where the energy of light is continuous, and where light travels through vacuum in straight lines as opposed to waves. This simplified setting is usually a good approximation of the true nature of light at human scales, and it is therefore most often sufficient for producing photorealistic results[1].

In this chapter, we will therefore provide a brief introduction to the geometric-optics-view that underlies computer graphics.

For a detailed survey of the various possible models for light transport and their relation to other fields (e.g. neutron transport, heat transfer, and acoustics), we refer the interested reader to the the introductory chapter of Veach's dissertation [Veach 1997].

[1] There are exceptions to this rule which—while being interesting—are out of the scope of this dissertation.

Table 4.1: *Light-transport-related quantities, sets, measures, elements, and transport kernels.*

Symbol	Name	Description
Φ	(Radiant) Flux	Power carried by light $[\mathrm{W} = \mathrm{J\,s^{-1}}]$.
E	Irradiance	Incident power per unit area $[\mathrm{W\,m^{-2}}]$.
R	Radiosity	Exitant power per unit area $[\mathrm{W\,m^{-2}}]$.
I	Intensity	Power per unit solid angle $[\mathrm{W\,sr^{-1}}]$.
L	Radiance	Power per unit projected area per solid angle $[\mathrm{W\,m^{-2}\,sr^{-1}}]$. Subscripts L_i, L_o, and L_e denote incident, outgoing, and emitted radiance, respectively. Other suffixes may be used to further differentiate.
W	Importance	Unitless sensor response per unit projected area per solid angle $[\mathrm{m^{-2}\,sr^{-1}}]$.
σ_s	Scattering coeff.	Number density of scattering particles on a line $[\mathrm{m^{-1}}]$.
σ_a	Absorbtion coeff.	Number density of absorbing particles on a line $[\mathrm{m^{-1}}]$.
$\sigma_\mathrm{t} = \sigma_\mathrm{s} + \sigma_\mathrm{a}$	Extinction coeff.	Number density of participating particles on a line $[\mathrm{m^{-1}}]$.
$\alpha = \sigma_\mathrm{s}/\sigma_\mathrm{t}$	Albedo	Fraction of scattering vs. total number of interactions. Informally: color of material.
$\mathcal{M}$	Space	3D space in which light transport is simulated.
$\partial\mathcal{V} \subseteq \mathcal{M}$	Surface	2D surface positions that participate in light transport.
$\mathcal{V} = \mathcal{M} - \partial\mathcal{V}$	Volume	Space that does not lie on a surface. May or may not participate in light transport.
$\mathcal{P}_k = \mathcal{M}^k$	Path space	Space of light paths with length k.
$\mathcal{P} = \bigcup_{k=2}^{\infty} \mathcal{P}_k$	Path space	Space of light paths of all lengths.
$\mathcal{S}^2$	Unit sphere	Set of unit-length 3D vectors.
$A(\partial\mathcal{V}')$	Area	Surface area measure of a subset $\partial\mathcal{V}' \subseteq \partial\mathcal{V}$.
$V(\mathcal{V}')$	Volume	Volume measure of a subset $\mathcal{V}' \subseteq \mathcal{V}$.
$M(\mathcal{M}')$	Volume or area	Volume or area measure of a subset $\mathcal{M}' \subseteq \mathcal{V}$ or $\mathcal{M}' \subseteq \partial\mathcal{V}$.
$\Pi(\mathcal{P}')$	Product	Product measure defined as $\int_{\mathcal{P}'} \prod_{i=1}^{k} \mathrm{d}M(\mathbf{x}_i)$ for $\mathbf{x}_1 \ldots \mathbf{x}_k = \bar{\mathbf{x}} \in \mathcal{P}' \subseteq \mathcal{P}$.
$\Omega(\mathcal{S}^{2\prime})$	Solid-angle	Solid angle measure of a subset $\mathcal{S}^{2\prime} \subseteq \mathcal{S}^2$.
$\mathbf{x} \in \mathcal{M}$	Position	3D vector. Subscripts $\mathbf{x}_\mathrm{i}$ and $\mathbf{x}_\mathrm{o}$ denote association with incident and outgoing radiance, respectively.
$\bar{\mathbf{x}} \in \mathcal{P}$	Light path	Light path consisting of any number of vertices ≥ 2.
$\omega \in \mathcal{S}^2$	Direction	Unit 3D vector. Subscripts ω_i and ω_o denote association with incident and outgoing radiance, respectively.
$f_\square$	Scattering function	Relates incident radiance to scattered (outgoing) radiance. In this dissertation: bidirectional scattering distribution function f_s (BSDF; Section 4.6), phase function f_p (Section 4.7), grain scattering distribution function f_g (GSDF, Section 6.3), shell transport function f_stf (STF, Section 6.2).
$\tau(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$	Transmittance	Fraction of unscattered light on a line between $\mathbf{x}_1$ and $\mathbf{x}_2$.
$G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$	Geometry term	Radiance change due to geometry between $\mathbf{x}_1$ and $\mathbf{x}_2$.
$E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$	Edge throughput	Product of transmittance and the geometry term.
$T(\bar{\mathbf{x}})$	Path throughput	Fractional radiance carried by a given light path $\bar{\mathbf{x}}$.
$f(\bar{\mathbf{x}})$	Path contribution	Radiance measured by a sensor through $\bar{\mathbf{x}}$.

4.1 Physical Quantities

We begin by introducing the physical quantities that underlie light transport. Those quantities are defined in terms of spatial and directional coordinates, which we define as follows; all notation in this chapter is summarized in Table 4.1.

Let $\mathcal{M} = \mathbb{R}^3$ be the set of all valid 3-dimensional positions. In this dissertation, whenever we refer to a *position* $\mathbf{x} \in \mathcal{M}$ we use the symbol $\mathbf{x}$. Light-matter interactions on 2-dimensional surfaces embedded in 3-dimensional space are treated differently from volumetric interactions, which leads us to partition $\mathcal{M}$ into the set of surface positions $\partial\mathcal{V} \subseteq \mathcal{M}$ and the complementary set of volume positions $\mathcal{V} = \mathcal{M} - \partial\mathcal{V}$.

Let $\mathcal{S}^2 \subset \mathbb{R}^3$ be the set of 3-dimensional unit-length vectors, i.e. the set of vectors on the unit sphere. In this dissertation, we refer to *directions* $\omega \in \mathcal{S}^2$ using the symbol ω. Whenever we model light-particle interactions, we refer to directions of *incident* light as ω_i and directions of *outgoing* light as ω_o, where, by convention, *both* ω_i and ω_o point *away* from the scattering event[2].

4.1.1 Flux

"Radiant flux" or just "flux", denoted Φ, is the *power* carried by light. It describes an amount of energy Q being carried by light *per unit time* and therefore has units of watts $[\mathrm{W} = \mathrm{J\,s^{-1}}]$:

$$\Phi(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t}\,. \tag{4.1}$$

An example for flux is the number of photons hitting a solar panel *per second*, multiplied by the energy carried by each photon.

In practice we are most often concerned with the steady-state solution of light-transport and we therefore omit the time dimension from the following derivations. Time-dependent effects on the human scale, such as motion blur, are usually treated as orthogonal to light-transport in the sense that they are approximated as a linear average of the steady-state light-transport over the effect duration. This approximation is valid when the time scales

[2] In our illustrations, such as in Figure 4.1, the direction of arrows follows the direction of light to aid in intuitive understanding. As a consequence, the direction of incident illumination is shown as $-\omega_\mathrm{i}$ (labeled as such), whereas the direction of outgoing illumination is shown as ω_o.

of interest are much longer than the time that light requires to propagate between objects[3].

Flux can be differentiated according to the wavelength of light, yielding *spectral* flux $[\mathrm{W} = \mathrm{J\,s^{-1}\,nm^{-1}}]$:

$$\Phi'(t,\lambda) = \frac{\mathrm{d}Q(t,\lambda)}{\mathrm{d}t\,\mathrm{d}\lambda}\,. \tag{4.2}$$

Since our goal is to produce photorealistic images, it is important to model color and therefore the wavelength of light. Most light-transport phenomena, however, are independent from the wavelength of light, which allows us to omit it in the following derivations and definitions. A rendering system that solves the equations of light transport can then simply run a monochromatic (wavelength independent) algorithm on multiple wavelengths independently to produce a colored image.

It is worth noting, that there are some visual phenomena where the above approximation breaks down. Flourescence in particular, where photons of one wavelength are absorbed and re-emitted at another wavelength, plays a role in the appearance of many human-scale objects and requires modeling of the interaction between spectral flux at different wavelengths. In this dissertation, however, we ignore this effect.

In the following, our goal will be to mathematically model the transport of flux throughout space and its interactions with materials. To do this, we must be able to formulate the spatio-directional distribution of flux, which requires the usage of its spatial, directional, and spatio-directional derivatives. These derivatives have established names and definitions in geometric-optics literature, which we will introduce next.

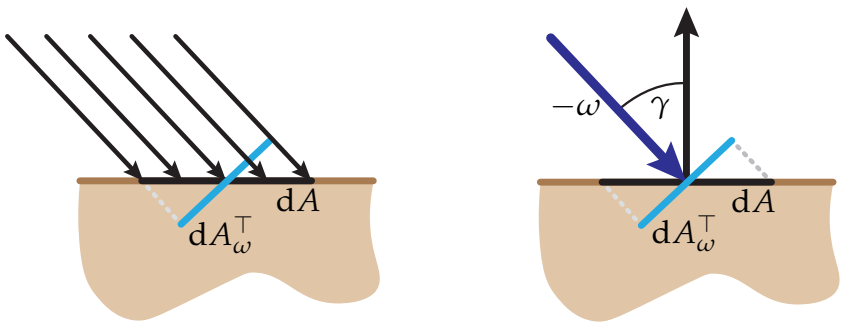
4.1.2 Irradiance and Radiosity

"Irradiance", denoted E, is the surface area density of flux. It is therefore measured in units of watts per square meter $[\mathrm{W\,m^{-2}}]$, which are obtained by differentiating flux with respect to the surface-area measure A:

$$E(\mathbf{x}) = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A(\mathbf{x})}\,. \tag{4.3}$$

To give an example: if flux corresponds to the number of photons hitting a solar panel per second, then irradiance corresponds to the number of photons

[3]There actually exists work that investigates the propagation of light at small time scales [Jarabo et al. 2014].



(a) *Foreshortening* ***(b)*** *Differential geometry*

Figure 4.1: *Parallel rays of light that arrive at a surface at an angle γ are spread by a factor of $\cos\gamma$, called the "foreshortening term" **(a)**. This spread of energy yields the differential relationship $\mathrm{d}A^{\top}_{\omega}(\mathbf{x}) = \mathrm{d}A(\mathbf{x})\,|\cos\gamma|$ found in the definition of radiance **(b)**.*

hitting a differential surface patch of the solar panel, divided by the size of the differential patch.

By convention, irradiance only refers to energy *arriving* at a surface. The complementary quantity (with the same units) referring to energy *leaving* a surface is called "radiosity", denoted B.

4.1.3 Intensity

It is also interesting to consider the *directional* density of flux. This quantity—flux per solid angle—is called "intensity" I and has units of watts per steradian $[\mathrm{W\,sr^{-1}}]$. We obtain intensity by differentiating flux with respect to the solid-angle measure Ω:

$$I(\omega) = \frac{\mathrm{d}\Phi(\omega)}{\mathrm{d}\Omega(\omega)}\,. \tag{4.4}$$

4.1.4 Radiance

"Radiance", denoted L, is defined as flux per solid angle *and* per area, resulting in units of watts per square meter per solid angle $[\mathrm{W\,m^{-2}\,sr^{-1}}]$. Radiance is obtained by differentiating flux with respect to the solid-angle measure Ω and projected area measure $A^{\top}_{\omega}$

$$L(\mathbf{x},\omega) = \frac{\mathrm{d}^2\Phi(\mathbf{x},\omega)}{\mathrm{d}\Omega(\omega)\,\mathrm{d}A^{\top}_{\omega}(\mathbf{x})}\,, \tag{4.5}$$

where the projected area measure $A^{\top}_{\omega}$ is defined as the surface area measure A projected onto a plane perpendicular to ω

$$\mathrm{d}A^{\top}_{\omega}(\mathbf{x}) = \mathrm{d}A(\mathbf{x})\,|N(\mathbf{x}) \cdot \omega| = \mathrm{d}A(\mathbf{x})\,|\cos\gamma|\,, \tag{4.6}$$

which we illustrate in Figure 4.1. The term $\cos\gamma = N(\mathbf{x}) \cdot \omega$ is called the "foreshortening term", referring to its correspondence to the stretching of light across slanted surfaces.

The choice of differentiating flux with respect to $A^{\top}_{\omega}$ rather than A may seem arbitrary at first, but there is a good reason for it: due to the differentiation with respect to an area that is perpendicular to the propagation direction ω, radiance is *independent* from the orientation of the surface that it originated from, which paves the way for modeling radiance-volume and radiance-surface interactions in the future.

Radiance has a core property that makes it convenient for modeling light transport: it remains constant along straight lines through vacuum. This property allows for an intuitive interpretation of radiance propagation through space as *rays of light*, which we will make use of in the following.

4.2 The Measurement Equation

Before we begin modeling the transport of light, we first define how light is *observed*. In the real world, us humans perceive light in the form of photons with varying wavelength (color) entering our eyes and stimulating the photoreceptive cells on our retinas. Photography devices are nothing else than machines that mimic the characteristics of human perception: instead of recording photons via cell stimulation, analog cameras record images on a sheet of transparent film coated with a material that darkens as it is exposed to light, and digital cameras employ microscopic sensors that convert incident photons into electricity.

In the virtual world, no such physical mechanisms for sensing light are required. Instead of having to carefully craft a physical object that responds to incident light in a convenient manner, we can simply model the sensitivity of a virtual sensor as a function. Suppose there is incident radiance $L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}})$ at a position *on* the sensor $\mathbf{x} \in \mathcal{I} \subseteq \mathcal{M}$ (or *in* the sensor; nothing prevents $\mathcal{I}$ from being a volume rather than a surface) from direction $\omega_{\mathrm{i}} \in \mathcal{S}^2$, then let the "importance" $W(\mathbf{x}, \omega_{\mathrm{i}})$ be the differential response of the sensor per incident radiance:

$$W(\mathbf{x}, \omega_{\mathrm{i}}) = \frac{\mathrm{d}^2 S(\mathbf{x}, \omega_{\mathrm{i}})}{\mathrm{d}^2 \Phi_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}})} = \frac{\mathrm{d}^2 S(\mathbf{x}, \omega_{\mathrm{i}})}{L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}})\,\mathrm{d}\Omega(\omega_{\mathrm{i}})\,\mathrm{d}M^{\top}_{\omega}(\mathbf{x})}\,, \tag{4.7}$$

where $M_\omega^\top(\mathcal{I})$ is either the projected surface area measure or the volume measure, depending on whether $\mathcal{I} \subseteq \partial\mathcal{V}$ or $\mathcal{I} \subseteq \mathcal{V}$:

$$M_\omega^\top(\mathcal{I}) = \begin{cases} A_\omega^\top(\mathcal{I}) & \text{if } \mathcal{I} \subseteq \partial\mathcal{V} \\ V(\mathcal{I}) & \text{otherwise} \end{cases} \iff \mathrm{d}M_\omega^\top(\mathbf{x}) = \begin{cases} \mathrm{d}A_\omega^\top(\mathbf{x}) & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ \mathrm{d}V(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4.8}$$

For notational convenience, we define $W(\mathbf{x}, \omega_\mathrm{i})$ not only over $\mathbf{x} \in \mathcal{I}$, but over all positions in space, setting its value to 0 whenever $\mathbf{x} \notin \mathcal{I}$.

Following from the definition of importance, its units are $[\mathrm{S/W}]$, where S (the sensor response) is a placeholder for any quantity of our choosing. If the sensor models a digital camera, then its response S might be voltage; if the sensor models an analog film, then S could represent darkening of the film. In our idealized virtual world, we make S unitless and leave its eventual interpretation to the user.

The measurement equation captures the total amount of measurement over the sensor domain

$$I = \int_{\mathcal{M}} \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) W(\mathbf{x}, \omega_\mathrm{i}) \, \mathrm{d}\Omega(\omega_\mathrm{i}) \, \mathrm{d}M(\mathbf{x}) \, . \tag{4.9}$$

Note that this equation produces a scalar value, which makes it—by itself—an unsuitable model for the light captured by a physical camera or by a human eye. It can, however, be used to model the light captured by *a single pixel* or *a single cell on a human eye's retina*. Because the regular-grid structure of the pixels in a camera is more convenient than the irregular arrangement of photoreceptive cells in a human eye, we model virtual sensors after cameras in this dissertation. We define a virtual image as a set of measurements I_{xy}, $x \in \{1, \ldots, m\}$, $y \in \{1, \ldots, n\}$, each of which corresponds to the light observed by a pixel in the y-th row and the x-th column of a virtual camera with a resolution of $m \times n$ pixels:

$$I_{xy} = \int_{\mathcal{M}} \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) W_{xy}(\mathbf{x}, \omega_\mathrm{i}) \, \mathrm{d}\Omega(\omega_\mathrm{i}) \, \mathrm{d}M(\mathbf{x}) \, . \tag{4.10}$$

In order to synthesize a virtual image, our goal is to numerically solve the measurement integral via Monte Carlo integration. In order to do so, we must be able to quantify incident radiance $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$, which depends on the geometry of the virtual scene, on the placement of virtual emitters (light sources), and on the interactions between the emitted light and the aforementioned geometry. In the remainder of this chapter, we provide a brief overview of the predominant models of $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ in graphics literature and techniques to compute them.

Surface and Volume Light Transport. Researchers developed mathematical models of light-matter interactions that can be broadly divided into two categories: light-surface interactions and light-volume interactions (hence the distinction between $\partial\mathcal{V}$ and $\mathcal{V}$). From a purely physical standpoint, both models are fundamentally the same; translucent and opaque matter is made out of the same fundamental particles, after all. It is, however, *mathematically* convenient to model interactions of light with optically dense matter on idealized, infinitely thin surfaces rather than volumetrically.

Since this dissertation contains contributions to volume rendering (Chapter 6) as well as surface rendering (Chapters 7 and 8), we introduce both models, including their combination, within this chapter. We begin with surface light transport in Section 4.3, then proceed to volumetric light transport in Section 4.4, and eventually combine both.

4.3 Surface Light Transport

Recall, that in order to solve the measurement equation (4.9) we require a mathematical formulation of the incident light $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ for a given position and direction, where, for now, we will constrain ourselves to positions on surfaces, i.e. $\mathbf{x} \in \partial\mathcal{V}$. We further assume that the space *between* surfaces $\mathcal{V}$ is empty (i.e. a vacuum) and does not interact with light. It follows, that radiance is conserved along straight lines. To make use of this fact, let us define the ray-tracing operator $r(\mathbf{x}, \omega)$ that finds the first ray-surface intersection along the ray prescribed by $\mathbf{x} + t \cdot \omega$ with $t \in (0, \infty)$:

$$\begin{aligned} r(\mathbf{x}, \omega) &= \mathbf{x} + \hat{t} \cdot \omega \\ \hat{t} &= \inf \left\{ t \in (0, \infty) \,\middle|\, (\mathbf{x} + t \cdot \omega) \in \partial\mathcal{V} \right\}, \end{aligned} \tag{4.11}$$

where, for simplicity, we assume that $\hat{t}$ always exists[4]. Using the ray-tracing operator and the conserversion of radiance along straight lines, we get the relationship

$$L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) = L_\mathrm{o}\big(r(\mathbf{x}, \omega_\mathrm{i}), -\omega_\mathrm{i}\big), \tag{4.12}$$

where $L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o})$ is defined to be the outgoing radiance at $\mathbf{x}$ towards ω_o, i.e. radiance *leaving* a surface at $\mathbf{x}$. It is important to note, that here we are making the implicit assumption that the speed of light is infinite. If it was

[4] By convention, the infimum of the empty set is ∞. To avoid having to deal with $\hat{t} = \infty$, let us simply assume there exists a (arbitrarily big) finitely sized sphere encompassing the virtual scene.

not (as in reality), then the above equation would include an additional *time* variable that captures the delay between light departing at $\mathbf{x}$ and arriving at $r(\mathbf{x}, \omega)$. There is, however, a reasonable justification to be made for the infinite speed-of-light assumption: us humans rarely perceive light in-flight, but rather only after it (very rapidly) reached an equilibrium state. The above equation—without a time variable—helps us capture precisely this equilibrium.

4.3.1 The Rendering Equation

We can now express incident light at one location $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ in terms of outgoing light at another location $L_\mathrm{o}(\mathbf{x}', \omega_\mathrm{o})$, where, in this case $\omega_\mathrm{o} = -\omega_\mathrm{i}$. This allows us to express a virtual image not in terms of the light *arriving* at a virtual camera, but in terms of the light *leaving* those points $\mathbf{x}' = r(\mathbf{x}, \omega_\mathrm{i})$ in the virtual scene that the virtual camera observes.

$$\begin{aligned} I_{xy} &= \int_{\mathcal{I}} \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) W_{xy}(\mathbf{x}, \omega_\mathrm{i}) \, \mathrm{d}\Omega(\omega_\mathrm{i}) \, \mathrm{d}M(\mathbf{x}) \\ &= \int_{\mathcal{I}} \int_{\mathcal{S}^2} L_\mathrm{o}(\mathbf{x}', \omega_\mathrm{o}) W_{xy}(\mathbf{x}, \omega_\mathrm{i}) \, \mathrm{d}\Omega(\omega_\mathrm{i}) \, \mathrm{d}M(\mathbf{x}) \, . \end{aligned} \tag{4.13}$$

The outgoing light $L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o})$ consists of two components: emitted light $L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o})$ and scattered light $L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o})$

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o}) \, . \tag{4.14}$$

Although there are many physically plausible models for emission $L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o})$ (e.g. combustion, flourescence, and chemiluminescence to name a few), in computer graphics we typically bake $L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o})$ into virtual scenes as a material parameter. Scattered light $L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o})$, on the other hand, is a function of *incident* light $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ and the surface properties that define the fraction of the incident light that is scattered from any given incident direction ω_i towards the outgoing direction of interest ω_o.

The Scattering Equation. The scattering equation is a cornerstone of physically based rendering. It formalizes the aforementioned interaction between incident and scattered radiance at any given surface location $\mathbf{x}$:

$$L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o}) = \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) \, f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}| \, \mathrm{d}\Omega(\omega_\mathrm{i}) \, . \tag{4.15}$$

The scattering equation contains an new term, $f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o})$, the so-called "bidirectional scattering distribution function" (BSDF). It describes the amount

of incident radiance from ω_i that is scattered towards ω_o (hence the arrow notation). The BSDF is surface-material specific, effectively capturing the material's appearance at $\mathbf{x}$. We will expand on the BSDF later in Section 4.6 and we will for now continue with our pursuit of expanding the measurement equation.

The Rendering Equation. Inserting the scattering equation (4.15) into Equation 4.14 yields the famous rendering equation [Kajiya 1986]

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}| \, \mathrm{d}\Omega(\omega_\mathrm{i}) \,. \tag{4.16}$$

Using the relationship in Equation 4.12, we can rewrite the rendering equation as a self-recursive Fredholm integral equation of the second kind

$$\begin{aligned} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) &\underset{4.12}{=} L_\mathrm{o}(\mathbf{x}', \omega_\mathrm{o}) \\ &\underset{4.16}{=} L_\mathrm{e}(\mathbf{x}', \omega_\mathrm{o}) + \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}', \omega_\mathrm{i}') f_\mathrm{s}(\mathbf{x}', \omega_\mathrm{i}' \to \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}'| \, \mathrm{d}\Omega(\omega_\mathrm{i}') \,. \end{aligned} \tag{4.17}$$

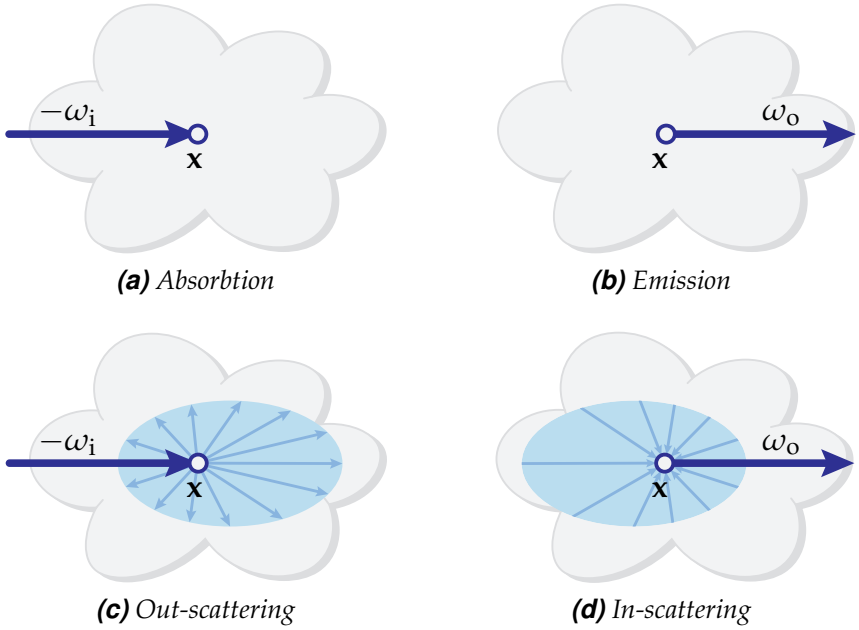
Each recursive integral over $L_\mathrm{i}(\mathbf{x}', \omega_\mathrm{i}')$ corresponds to scattering of light. The $L_\mathrm{e}(\mathbf{x}', \omega_\mathrm{o})$ terms *within* each recursive integral capture the emitted light at $\mathbf{x}'$ that is propagated towards the outermost $\mathbf{x}$ through all layers of recursion.

As it turns out, applying Monte Carlo integration (3.3) to the recursive rendering equation directly results in the unidirectional path-tracing algorithm that we introduced in the beginning of this dissertation. We will explain this in more detail later, in Section 4.5, and focus, for now, on incorporating volumetric light transport into the rendering equation.

4.4 Volumetric Light Transport

Until here, we investigated light-transport simulation in the presence of only idealized light-surface interactions. However, light interacts not only with surfaces—i.e. (almost) infinitely dense matter—but also with sparser volumetric arrangements of particles. In rendering literature, we refer to such volumetrically scattering media that participate in light transport as "participating media". Light transport within participating media is characterized by two largely separate properties:

1. the arrangement and density of scatterers (i.e. small particles interacting with light), governing spatial propagation of light, and



(a) *Absorbtion* **(b)** *Emission*

(c) *Out-scattering* **(d)** *In-scattering*

Figure 4.2: *The radiative transfer equation consists of four additive terms: absorbtion* ***(a)****, emission* ***(b)****, out-scattering* ***(c)****, and in-scattering* ***(d)****. Absorbtion and out-scattering are always negative, i.e. they remove radiance from a ray of light, whereas their respective counterparts emission and in-scattering are positive, thereby introducing radiance.*

2. the characteristics of light-particle interactions themselves, governing directional propagation of light and energy dissipation.

Due to difficulties associated with modeling the aforementioned properties in full generality, we focus on the framework of "radiative transfer" [Chandrasekar 1960], where particles are assumed to be infinitesimally small and arranged independently (i.e. there is no mutual correlation in the position of particles). The following sections will introduce the basic mathematics that describe radiative transfer.

4.4.1 Radiative Transfer

Recall, that in purely surface-based light transport, incident radiance $L_i(\mathbf{x}, \omega_i)$ at some surface position $\mathbf{x} \in \partial\mathcal{V}$ is related to outgoing radiance at another surface position $\mathbf{x}' = r(\mathbf{x}, \omega_i)$ via the ray-tracing operator (Equation 4.11)

$$L_i(\mathbf{x}, \omega_i) = L_o(\mathbf{x}', -\omega_i)\,.$$

This relationship hinges on the assumption that radiance is invariant as it traverses the space between surfaces, which is *not* the case in the presence of participating media: in such media, radiance may vary along straight lines.

In the absence of surfaces (i.e. sharp discontinuies), however, we can postulate that radiance only varies *continuously*. The notion of continuous variation is captured by the property that incident and outgoing radiance at the *same* position are equal, i.e. there are no discontinuities

$$L_{\mathrm{i}}(\mathbf{x}, -\omega_{\mathrm{o}}) = L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}) \,, \quad \forall \mathbf{x} \in \mathcal{V} \,. \tag{4.18}$$

Due to this equivalence of incident and outgoing radiance, we will refer to radiance in volumes simply as $L(\mathbf{x}, \omega) = L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}})$, where $\omega = \omega_{\mathrm{o}}$.

The continuous *change* in radiance as the volume $\mathcal{V}$ between surfaces $\partial\mathcal{V}$ is traversed is then characterized by the differential "radiative transfer equation" (RTE) [Chandrasekar 1960]

$$(\omega \cdot \nabla) L(\mathbf{x}, \omega) = (\omega \cdot \nabla) \Big(L_{\mathrm{e}}(\mathbf{x}, \omega) + L_{\mathrm{in}}(\mathbf{x}, \omega) + L_{\mathrm{out}}(\mathbf{x}, \omega) + L_{\mathrm{a}}(\mathbf{x}, \omega) \Big) \,, \tag{4.19}$$

where we use

$$(\omega \cdot \nabla) L(\mathbf{x}, \omega) = \frac{\mathrm{d} L(\mathbf{x} + \omega \cdot t, \omega)}{\mathrm{d}t} \tag{4.20}$$

around $t = 0$ for notational convenience. Notice, that due to this additional derivative, the expression $(\omega \cdot \nabla) L(\mathbf{x}, \omega)$ has units of $[\mathrm{W\,m^{-3}\,sr^{-1}}]$. We refer to it as "volumetric radiance" due to its volume (rather than surface area) density.

The RTE consists of four additive components: "emission" L_{e}, "in-scattering" L_{in}, "out-scattering" L_{out}, and "absorbtion" L_{a}. We explain all of these components in the following paragraphs and illustrate them in Figure 4.2.

Emission. Some participating media *emit* light, leading to the emission term $(\omega \cdot \nabla) L_{\mathrm{e}}(\mathbf{x}, \omega)$ in the RTE. Emission can be caused by various physical phenomena, such as combustion (e.g. fire), ionization (e.g. auroras), and chemiluminescence (i.e. chemical reactions). Due to the large number of possible emission sources we choose the most convenient path of leaving the term up to the virtual scene description, just like surface emission.

In-Scattering. The in-scattering term is conceptually similar to the scattering equation (4.15): it captures the aggregate amount of scattered radiance into direction ω, summed over all possible incident directions ω_{i}.

$$(\omega \cdot \nabla) L_{\mathrm{in}}(\mathbf{x}, \omega) = \sigma_{\mathrm{s}}(\mathbf{x}) \int_{\mathcal{S}^2} L(\mathbf{x}, -\omega_{\mathrm{i}}) \, f_{\mathrm{p}}(\mathbf{x}, \omega_{\mathrm{i}} \to \omega) \, \mathrm{d}\Omega(\omega_{\mathrm{i}}) \,. \tag{4.21}$$

Here, the "scattering coefficient" $\sigma_s(\mathbf{x})$ $[m^{-1}]$ is the 1-dimensional number density of scattering particles (i.e. number of particles per meter) along direction ω at $\mathbf{x}$; it determines how frequently a ray of light scatters while traversing a participating medium. The so-called "phase function" $f_p(\mathbf{x}, \omega_i \to \omega)$ is comparable to the BSDF as it relates incident radiance from one direction ω_i to scattered outgoing radiance towards another direction ω. As with the BSDF, we will formally define the phase function and present several physically plausible models later on in Section 4.7 and we will focus, for now, on expanding the measurement equation.

Out-Scattering. The out-scattering term is conceptually similar to the in-scattering term. Just like light can be scattered *into* ω from any other direction, light that is *already* traveling towards ω can also be scattered away. This effect is captured by the out-scattering term

$$(\omega \cdot \nabla) L_{\text{out}}(\mathbf{x}, \omega) = -\sigma_s(\mathbf{x}) L(\mathbf{x}, \omega)\,, \tag{4.22}$$

which—like in-scattering—must also proportional to the density of scatterers $\sigma_s(\mathbf{x})$.

Absorbtion. The last component of the RTE describes absorbed radiance:

$$(\omega \cdot \nabla) L_a(\mathbf{x}, \omega) = -\sigma_a(\mathbf{x}) L(\mathbf{x}, \omega)\,. \tag{4.23}$$

In this term, the "absorbtion coefficient" $\sigma_a(\mathbf{x})$ $[m^{-1}]$ has a similar role as the scattering coefficient $\sigma_s(\mathbf{x})$ has in out-scattering: it describes the 1-dimensional number-density of *absorbing* particles. Any light that hits an absorbing particle is lost, and therefore the change in radiance due to absorbtion $L_a(\mathbf{x}, \omega)$ is proportional to the density of absorbing particles $\sigma_a(\mathbf{x})$.

Combining all these terms into a single equation, yields the RTE

$$\begin{aligned}(\omega \cdot \nabla) L(\mathbf{x}, \omega) = {} & (\omega \cdot \nabla) L_e(\mathbf{x}, \omega) \\ & + \sigma_s(\mathbf{x}) \int_{\mathcal{S}^2} L(\mathbf{x}, -\omega_i)\, f_p(\mathbf{x}, \omega_i \to \omega)\, \mathrm{d}\Omega(\omega_i) \\ & - \sigma_s(\mathbf{x}) L(\mathbf{x}, \omega) \\ & - \sigma_a(\mathbf{x}) L(\mathbf{x}, \omega)\,. \end{aligned} \tag{4.24}$$

4.4.2 Extinction and Albedo

Due to the similar structure of absorbtion and out-scattering, the absorbtion coefficient $\sigma_a(\mathbf{x})$ and scattering coefficient $\sigma_s(\mathbf{x})$ are often combined into a

single "extinction coefficient" $\sigma_t(\mathbf{x}) = \sigma_a(\mathbf{x}) + \sigma_s(\mathbf{x})$ $[m^{-1}]$, that describes the 1-dimensional number-density of *all* light-particle interactions. Rewriting the RTE using $\sigma_t(\mathbf{x})$ results in

$$\begin{aligned}(\omega \cdot \nabla)L(\mathbf{x},\omega) = (\omega \cdot \nabla)L_e(\mathbf{x},\omega) \\ + \sigma_s(\mathbf{x}) \int_{\mathcal{S}^2} L(\mathbf{x},-\omega_i)\, f_p(\mathbf{x},\omega_i \to \omega)\, d\Omega(\omega_i) \\ - \sigma_t(\mathbf{x})L(\mathbf{x},\omega)\,. \end{aligned} \tag{4.25}$$

We will refer to the term $(\omega \cdot \nabla)L_{ext}(\mathbf{x},\omega) = \sigma_t(\mathbf{x})L(\mathbf{x},\omega)$ as "extinction term". Up until here, we described radiative transfer under the assumption that the participating medium consists of two types of infinitesimal particles: scatterers and absorbers with corresponding densities $\sigma_s(\mathbf{x})$ and $\sigma_a(\mathbf{x})$. However, the combination of scattering and absorbtion into the extinction coefficient $\sigma_t(\mathbf{x})$ gives rise to an alternative interpretation: a medium consisting of only one kind of particle with density $\sigma_t(\mathbf{x})$ that scatters a fraction $\alpha(\mathbf{x}) = \sigma_s(\mathbf{x})/\sigma_t(\mathbf{x})$ of radiance and absorbs the remaining fraction $1 - \alpha(\mathbf{x})$. In this context, α is referred to as the "albedo" of the participating medium. This interpretation is particularly interesting because it connects the phase function with the BSDF. The BSDF has its "albedo" (i.e. its integral over the projected solid sphere) baked in, while the phase function is normalized to integrate to 1 over the solid sphere. In this sense, the product of the albedo and the phase function—i.e. $\alpha(\mathbf{x})f_p(\mathbf{x},\omega_i \to \omega_o)$—corresponds more directly to the BSDF $f_s(\mathbf{x},\omega_i \to \omega_o)$ than the phase function alone does.

Both views, i.e. scattering and absorbtion vs. extinction and albedo, typically lead to equivalent light transport, but depending on the situation can each be mathematically simpler or harder to work with. In this dissertation we parameterize participating media in terms of albedo and extinction, referring to the tuple (α, σ_t, f_p) as "RTE parameters".

4.4.3 Integral Form of the Radiative Transfer Equation

Similar to the rendering equation, we would like to derive incident radiance $L_i(\mathbf{x},\omega_i)$ as a function of outgoing radiance $L_o(\mathbf{x}',\omega_o)$ at position $\mathbf{x}' = r(\mathbf{x},\omega)$ and towards direction $\omega_o = -\omega_i$. To this end, we integrate the RTE from $\mathbf{x}'$ to $\mathbf{x}$ along ω_o. Since any radiance introduced by the terms L_e and L_{in} on the way from $\mathbf{x}'$ to $\mathbf{x}$ is diminished by the extinction term L_{ext} along the remainder of the way towards $\mathbf{x}$, it is helpful to pre-integrate the fractional loss of radiance due to L_{ext}.

Setting all other terms of the RTE to zero, we obtain

$$(\omega \cdot \nabla) L(\mathbf{x}, \omega) = -(\omega \cdot \nabla) L_{\text{ext}}(\mathbf{x}, \omega) = -\sigma_{\text{t}}(\mathbf{x}) L(\mathbf{x}, \omega)\,, \tag{4.26}$$

which is an ordinary differential equation with solution

$$L(\mathbf{x}, \omega) = \tau(\mathbf{x} \leftrightarrow \mathbf{x}') L(\mathbf{x}', \omega)\,, \tag{4.27}$$

$$\tau(\mathbf{x} \leftrightarrow \mathbf{x}') = e^{-\int_0^d \sigma_{\text{t}}(\mathbf{x}_t)\,\mathrm{d}t}\,, \tag{4.28}$$

where $d = \|\mathbf{x} - \mathbf{x}'\|$ and $\mathbf{x}_t = \mathbf{x} + \omega \cdot t$. The factor $\tau(\mathbf{x} \leftrightarrow \mathbf{x}')$ which is called "transmittance", is the *fraction* of radiance originating from $\mathbf{x}$ towards ω that arrives at $\mathbf{x}'$. Using $\tau(\mathbf{x} \leftrightarrow \mathbf{x}')$, we obtain the *entire* RTE integral

$$L(\mathbf{x}, \omega) = \int_0^\infty \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t) \Big((\omega \cdot \nabla) L_{\text{e}}(\mathbf{x}_t, \omega) + (\omega \cdot \nabla) L_{\text{in}}(\mathbf{x}_t, \omega) \Big)\,\mathrm{d}t\,, \tag{4.29}$$

where we integrate towards infinity because arbitrarily distant positions may still contribute small amounts of radiance in the absence of occluding surfaces. Replacing $(\omega \cdot \nabla) L_{\text{in}}(\mathbf{x}_t, \omega)$ with its definition and back-substituting L with L_{i} and L_{o}, it becomes clear that the RTE—like the rendering equation—is self-recursive

$$\begin{aligned} L_{\text{i}}(\mathbf{x}, \omega_{\text{i}}) &\underset{4.18}{=} L_{\text{o}}(\mathbf{x}, \omega_{\text{o}}) \\ &\underset{4.29}{=} \int_0^\infty \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t) \Big((\omega_{\text{o}} \cdot \nabla) L_{\text{e}}(\mathbf{x}_t, \omega_{\text{o}}) \\ &\qquad + \sigma_{\text{s}}(\mathbf{x}_t) \int_{\mathcal{S}^2} L_{\text{i}}(\mathbf{x}_t, \omega_{\text{i}}') \, f_{\text{p}}(\mathbf{x}_t, \omega_{\text{i}}' \rightarrow \omega_{\text{o}}) \,\mathrm{d}\Omega(\omega_{\text{i}}') \Big)\,\mathrm{d}t\,. \end{aligned} \tag{4.30}$$

We would like to note, that the RTE as presented above is limited to infinitesimal scatterers that are spatially distributed in an *independent* manner, leading to transmittance that is *exponential* (Equation 4.28). As soon as scatterer positions are permitted to correlate, the solution in Equation 4.27 may be non-exponential. In this dissertation, we only consider exponential transmittance and we therefore refer the interested reader to recent research on non-exponential light transport [d'Eon 2018; Bitterli et al. 2018] for more information.

4.4.4 Combining the RTE with the Rendering Equation

To model a combination of volumetric and surface-based light transport, we combine the RTE with the rendering equation into the "volume rendering equation". This combinations requires surprisingly little effort. The key insight that is needed, is that within a scene that contains surfaces *and* participating media, the incident radiance $L_{\text{i}}(\mathbf{x}, \omega_{\text{i}})$ is the *sum* of volumetrically transported radiance and surface-transported radiance, both arriving at $\mathbf{x}$ from ω_{i}. We illustrate this in Figure 4.3.

When building this sum, we must account for surface occlusion within the RTE and for volume attenuation in the rendering equation. In this dissertation, we refer to the combination of the RTE and the rendering equation as the "volume rendering equation".

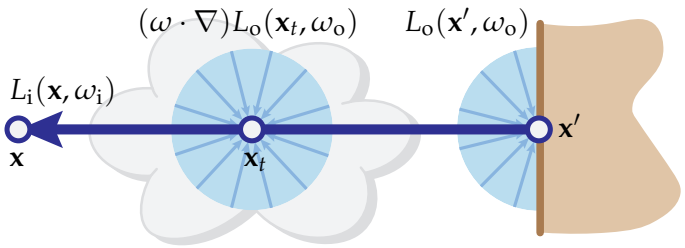


Figure 4.3: *Terms of the volume rendering equation.*

The volume-attenuation-aware rendering equation simply has an additional transmittance factor

$$L_i^{\partial\mathcal{V}}(\mathbf{x}, \omega_i) = \tau(\mathbf{x} \leftrightarrow \mathbf{x}') \Big(L_e(\mathbf{x}', \omega_o) + L_s(\mathbf{x}', \omega_o) \Big) , \tag{4.31}$$

and the surface-occlusion-aware RTE integrates volumetric transport only up to the nearest surface intersection rather than to infinity

$$L_i^{\mathcal{V}}(\mathbf{x}, \omega_i) = \int_0^{\hat{t}} \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t) \Big((\omega_o \cdot \nabla) L_e(\mathbf{x}_t, \omega_o) + (\omega_o \cdot \nabla) L_{in}(\mathbf{x}_t, \omega_o) \Big) \, \mathrm{d}t \, . \tag{4.32}$$

The sum $L_i(\mathbf{x}, \omega_i) = L_i^{\partial\mathcal{V}}(\mathbf{x}, \omega_i) + L_i^{\mathcal{V}}(\mathbf{x}, \omega_i)$ then leads to the combined equation

$$\boxed{\begin{aligned} L_i(\mathbf{x}, \omega_i) = {} & \int_0^{\hat{t}} \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t) \Big((\omega_o \cdot \nabla) L_e(\mathbf{x}_t, \omega_o) + (\omega_o \cdot \nabla) L_{in}(\mathbf{x}_t, \omega_o) \Big) \, \mathrm{d}t \\ + {} & \tau(\mathbf{x} \leftrightarrow \mathbf{x}') \Big(L_e(\mathbf{x}', \omega_o) + L_o(\mathbf{x}', \omega_o) \Big) . \end{aligned}} \tag{4.33}$$

Since both additive components of the volume rendering equation are self-recursive, the volume rendering equation itself is, too

$$\begin{aligned} L_i(\mathbf{x}, \omega_i) = {} & \int_0^{\hat{t}} \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t) \Big((\omega_o \cdot \nabla) L_e(\mathbf{x}_t, \omega_o) \\ & \underbrace{\qquad + \sigma_s(\mathbf{x}_t) \int_{\mathcal{S}^2} L_i(\mathbf{x}_t, \omega_i') \, f_p(\mathbf{x}_t, \omega_i' \to \omega_o) \, \mathrm{d}\Omega(\omega_i') \Big) \, \mathrm{d}t}_{\text{Volumetric light transport}} \\ + {} & \tau(\mathbf{x} \leftrightarrow \mathbf{x}') \Big(L_e(\mathbf{x}', \omega_o) \\ & \underbrace{\qquad + \int_{\mathcal{S}^2} L_i(\mathbf{x}', \omega_i') f_s(\mathbf{x}', \omega_i' \to \omega_o) \, |\cos \gamma_i'| \, \mathrm{d}\Omega(\omega_i') \Big)}_{\text{Surface light transport}} . \end{aligned} \tag{4.34}$$

Next, we will discuss the application of Monte Carlo integration so solve the volume rendering equation.

4.5 Path Tracing

To render an image, we estimate the measurement equation for every pixel I_{xy} via Monte Carlo (3.3)

$$\langle I_{xy} \rangle = \frac{L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, W_{xy}(\mathbf{x}, \omega_\mathrm{i})}{q(\mathbf{x}, \omega_\mathrm{i})}\,. \tag{4.35}$$

Since the incident radiance at the sensor $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ is given by the (volume) rendering equation, we recursively apply Monte Carlo to the nested integrals while preventing infinite recursion with russian roulette (Section 3.5.1).

4.5.1 Surface Path Tracing

Applying Monte Carlo (Equation 3.3) naïvely to the surface rendering equation (4.17) yields the estimator

$$\big\langle L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) \big\rangle = L_\mathrm{e}(\mathbf{x}', \omega_\mathrm{o}) + \frac{L_\mathrm{i}(\mathbf{x}', \omega_\mathrm{i}')\, f_\mathrm{s}(\mathbf{x}', \omega_\mathrm{i}' \to \omega_\mathrm{o})\, |\cos \gamma_\mathrm{i}'|}{q(\omega_\mathrm{i}' \,|\, \mathbf{x}', \omega_\mathrm{o})}\,. \tag{4.36}$$

Recursively applying Monte Carlo recursively to the nested L_i terms directly corresponds to the unidirectional path tracing algorithm that we briefly introduced in the beginning of this dissertation: each nested Monte Carlo estimator obtains the next path vertex $\mathbf{x}'$ via the ray tracing operator and subsequently samples a random direction ω_i' to continue the path in. Furthermore, each nested Monte Carlo estimator also evaluates $L_\mathrm{e}(\mathbf{x}', \omega_\mathrm{o})$ and records its value, weighted by the product of the weights $\frac{f_\mathrm{s}(\mathbf{x}', \omega_\mathrm{i}' \to \omega_\mathrm{o})\, |\cos \gamma_\mathrm{i}'|}{q(\omega_\mathrm{i}' \,|\, \mathbf{x}', \omega_\mathrm{o})}$ for all preceeding vertices.

4.5.2 Volume Path Tracing.

The same approach as above leads to a purely volumetric path tracer when applied to the RTE as opposed to the rendering equation

$$\big\langle L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) \big\rangle = \frac{\tau(\mathbf{x} \leftrightarrow \mathbf{x}_t)}{q(t|\mathbf{x}, \omega)} \left((\omega_\mathrm{o} \cdot \nabla) L_\mathrm{e}(\mathbf{x}_t, \omega_\mathrm{o}) + \frac{L_\mathrm{i}(\mathbf{x}_t, \omega_\mathrm{i}')\, \sigma_\mathrm{s}(\mathbf{x}_t) f_\mathrm{p}(\mathbf{x}_t, \omega_\mathrm{i}' \to \omega_\mathrm{o})}{q(\omega_\mathrm{i}' \,|\, t, \mathbf{x}, \omega)} \right). \tag{4.37}$$

In contrast to the surface path tracer from above, the next path vertex (in this case $\mathbf{x}_t$) is not obtained via ray tracing any surface geometry, but instead by randomly sampling a distance t. The path therefore traverses the scene

volume by interleaving distance sampling and directional sampling. Additionally, when computing the fraction of $(\omega_o \cdot \nabla) L_e(\mathbf{x}_t, \omega_o)$ that reaches the camera, the necessary weight now also includes a transmittance factor $\frac{\tau(\mathbf{x} \leftrightarrow \mathbf{x}_t)}{q(t|\mathbf{x},\omega)}$.

4.5.3 Unified Path Tracing

Naïvely applying Monte Carlo to the volume rendering equation results in

$$\left\langle L_i(\mathbf{x}, \omega_i) \right\rangle = \left\langle L_i^{\partial\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle + \left\langle L_i^{\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle, \tag{4.38}$$

$$\left\langle L_i^{\partial\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle = \tau(\mathbf{x} \leftrightarrow \mathbf{x}') \left(L_e(\mathbf{x}', \omega_o) + \frac{L_i(\mathbf{x}', \omega_i'')\, f_s(\mathbf{x}', \omega_i'' \to \omega_o)\, |\cos \gamma_i''|}{q(\omega_i'' \mid \mathbf{x}', \omega_o)} \right), \tag{4.39}$$

$$\left\langle L_i^{\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle = \frac{\tau(\mathbf{x} \leftrightarrow \mathbf{x}_t)}{q(t|\mathbf{x}, \omega)} \left((\omega_o \cdot \nabla) L_e(\mathbf{x}_t, \omega_o) + \frac{L_i(\mathbf{x}_t, \omega_i')\, \sigma_s(\mathbf{x}_t) f_p(\mathbf{x}_t, \omega_i' \to \omega_o)}{q(\omega_i' \mid t, \mathbf{x}, \omega)} \right), \tag{4.40}$$

where the same modifications were made as when unifying the surface rendering equation and the RTE: t now lies in the range $(0, \hat{t})$ as opposed to $(0, \infty)$, and the surface rendering equation is scaled by transmittance $\tau(\mathbf{x} \leftrightarrow \mathbf{x}')$.

Although this estimator is valid, it corresponds to a path tracer where at every scattering interaction, *two* samples are taken: one that estimates surface scattering and one that estimates volume scattering. This, unfortunately, leads to geometric cost in the path length due to a doubling of Monte Carlo estimators at every scattering interaction. In practice, this problem is avoided by randomly choosing between one of the two branches according to a probability $\tau \in [0, 1]$

$$\left\langle L_i(\mathbf{x}, \omega_i) \right\rangle = \begin{cases} \left\langle L_i^{\partial\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle / \tau & \text{if } \xi < \tau \\ \left\langle L_i^{\mathcal{V}}(\mathbf{x}, \omega_i) \right\rangle / (1 - \tau) & \text{otherwise.} \end{cases} \tag{4.41}$$

The probability $\tau \in [0, 1]$ effectively controls the ratio between volume scattering and surface scattering. It is common to use $\tau = \int_{\hat{t}}^{\infty} \tau(\mathbf{x} \leftrightarrow \mathbf{x}_t)\, \mathrm{d}t$ as it conveniently coincides with importance sampling $\tau(\mathbf{x} \leftrightarrow \mathbf{x}_t)$ with $t \in (0, \infty)$.

4.5.4 The Path Integral

The previous sections showed that surface and volume light transport is characterized by *infinitely many* recursively nested integrals over the *finite-dimensional* directional (and in the case of volumes, also the spatial) domain.

Each nested integral captures the scattering characteristics of radiance for a given incident direction and position in space. Even though the recursion is infinite, each nested integral contributes a certain finite amount of radiance to the measurement equation due to its L_e term. By separating light transport into the additive L_e terms in each of the nested integrals (i.e. into the contributions of k-vertex paths for each k) analogously to Section 3.5 on nested integrals, the rendering equation can be expressed as a *single* integral over the *infinite-dimensional* domain of light paths. This form of the rendering equation is referred to as the "path integral" [Veach 1997].

Definition. Let $\mathcal{P}_k = \mathcal{M}^k$, $\forall k \in \{2, 3, \dots \infty\}$ be the set of light paths with k vertices. A light path $\mathbf{x}_1 \dots \mathbf{x}_k = \overline{\mathbf{x}}_k \in \mathcal{P}_k$ consists of k vertex positions $\mathbf{x}_i \in \mathcal{M}$. The set of *all* light paths is then $\mathcal{P} = \bigcup_{k=2}^{\infty} \mathcal{P}_k$, where individual light paths with arbitrary numbers of vertices are denoted $\overline{\mathbf{x}} \in \mathcal{P}$.

In order to perform differential calculus on $\mathcal{P}$, we define the "product measure" $\Pi(\mathcal{P}')$, $\mathcal{P}' \subseteq \mathcal{P}$ in terms of its differential:

$$\mathcal{P}' = \bigcup_{k=2}^{\infty} \mathcal{P}'_k, \tag{4.42}$$

$$\Pi(\mathcal{P}') = \sum_{k=2}^{\infty} \Pi(\mathcal{P}'_k), \tag{4.43}$$

$$\mathrm{d}\Pi(\overline{\mathbf{x}}_k) = \prod_{i=1}^{k} \mathrm{d}M(\mathbf{x}_i). \tag{4.44}$$

The product measure is the product of differential area or volume $\mathrm{d}M(\mathbf{x}_i)$ (depending on whether $\mathbf{x}_i \in \partial\mathcal{V}$ or $\mathbf{x}_i \in \mathcal{V}$) around each vertex $\mathbf{x}_i$ of the path $\overline{\mathbf{x}}$

$$M(\mathcal{M}') = \begin{cases} A(\mathcal{M}') & \text{if } \mathcal{M}' \subseteq \partial\mathcal{V} \\ V(\mathcal{M}') & \text{otherwise} \end{cases} \iff \mathrm{d}M(\mathbf{x}) = \begin{cases} \mathrm{d}A(\mathbf{x}) & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ \mathrm{d}V(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4.45}$$

We further define familiar functions in terms of their corresponding path vertices, rather than position-direction pairs:

$$\omega_{ij} = \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} \tag{4.46}$$

$$W_{xy}(\mathbf{x}_i \to \mathbf{x}_j) = W_{xy}(\mathbf{x}_j, \omega_{ji}) \tag{4.47}$$

$$L(\mathbf{x}_i \to \mathbf{x}_j) = L(\mathbf{x}_i, \omega_{ij}) \tag{4.48}$$

$$f(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) = f(\mathbf{x}_j, \omega_{ji} \to \omega_{jk}), \tag{4.49}$$

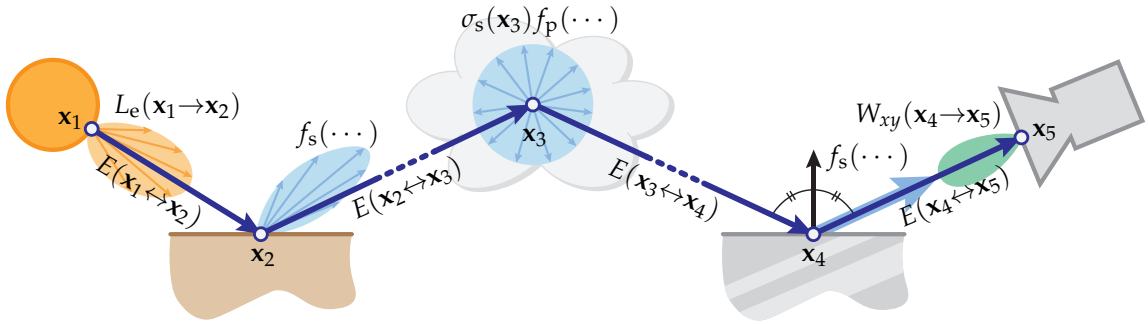


Figure 4.4: *Illustration of the terms of the path integral. The contribution of a k-vertex light path is the product of emitted light $L_e(\mathbf{x}_1 \to \mathbf{x}_2)$, the edge throughputs $E(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1})$, the scattering kernels $f(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1})$, and sensor importance $W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k)$.*

where the arrow denotes the direction of radiance propagation. Lastly, we define volume-surface-independent emission

$$L_e(\mathbf{x}_i \to \mathbf{x}_j) = \begin{cases} L_e(\mathbf{x}_i \to \mathbf{x}_j) & \text{if } \mathbf{x}_j \in \partial\mathcal{V} \\ (\omega_{ij} \cdot \nabla) L_e(\mathbf{x}_i \to \mathbf{x}_j) & \text{otherwise,} \end{cases} \tag{4.50}$$

scattering

$$f(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) = \begin{cases} f_s(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ \sigma_s(\mathbf{x}_k) f_p(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) & \text{otherwise,} \end{cases} . \tag{4.51}$$

and propagation

$$E(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) = \tau(\mathbf{x}_i \leftrightarrow \mathbf{x}_j)\, G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) \tag{4.52}$$

$$G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) = \frac{D(\mathbf{x}_i, \omega_{ij})\, D(\mathbf{x}_j, \omega_{ji})}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} , \tag{4.53}$$

$$D(\mathbf{x}, \omega) = \begin{cases} |N(\mathbf{x}) \cdot \omega| & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ 1 & \text{otherwise.} \end{cases} \tag{4.54}$$

We can then express the measurement equation as the "path integral" over $\mathcal{P}$

$$I_{xy} = \int_{\mathcal{P}} f_{xy}(\bar{\mathbf{x}})\, \mathrm{d}\Pi(\bar{\mathbf{x}}) \tag{4.55}$$

$$f_{xy}(\bar{\mathbf{x}}) = L_e(\mathbf{x}_1 \to \mathbf{x}_2)\, T(\bar{\mathbf{x}})\, W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \tag{4.56}$$

$$T(\bar{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \prod_{i=2}^{k-1} T(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) E(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) , \tag{4.57}$$

with the integrand $f_{xy}(\bar{\mathbf{x}})$ being the "path contribution function" and $T(\bar{\mathbf{x}})$ the "path throughput" of $\bar{\mathbf{x}}$. We illustrate the terms of the path integral in Figure 4.4 and we provide a full derivation of it in Section A.1.

Path Integral Monte Carlo. Due to its closed form, the path-integral formulation permits the application of certain general-purpose Monte Carlo integration techniques that would otherwise be difficult to reason about. More specifically, due to the integration over the path variable $\bar{\mathbf{x}}$ it becomes clear, that *any* kind of path sampling technique—not just unidirectional path tracing as derived before—can be used for solving the (volumetric) rendering equation with Monte Carlo.

This theoretical insight facilitated the development of several advanced rendering techniques, such as bidirectional path tracing [Lafortune and Willems 1993, 1996; Veach and Guibas 1994] and Metropolis light transport [Veach and Guibas 1997]. Although in this dissertation we do not make explicit use of the path integral formulation, it is useful for reasoning about the efficiency of traditional path tracing algorithms and for understanding the primary-sample-space formulation we will introduce next.

4.5.5 Primary-Sample Space

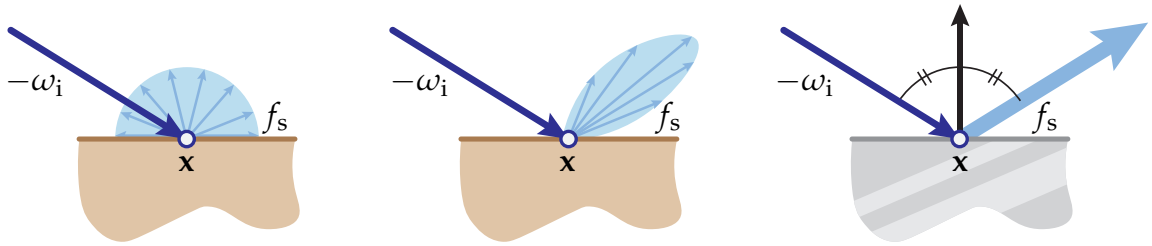
When implementing a Monte Carlo integration scheme on a computer, it is a fundamental requirement to computationally draw random samples from a variety of importance-sampling distributions. Such sampling routines are based on a number of techniques, including the inversion method (Section 2.6.1) and rejection sampling (Section 2.6.2). All these techniques are based in some way or another on a deterministic mapping from a fundamental source of randomness to the desired distribution. Usually, this source of randomness is implemented as the canonical uniform distribution $\mathcal{U}(0,1)$ directly on top of an underlying pseudo-random number generator; the Mersenne Twister [Matsumoto and Nishimura 1998] and, more lately, the PCG family [O'Neill 2014] are popular choices in renderers due to their high performance and good statistical qualities.

It follows then, that a randomly generated light path is the result of a deterministic mapping $\rho(z) : \mathbb{U} \to \mathcal{P}$, where $\mathbb{U}$ is the D-dimensional hypercube $\mathbb{U} = [0,1]^D$ with D being sufficiently big, and $z \sim \mathcal{U}(\mathbb{U})$

$$\bar{\mathbf{x}} = \rho(z)\,. \tag{4.58}$$

The elements of the vector z are called "primary samples" and $\mathbb{U}$ is called "primary sample space".

Notice, that this deterministic map is nothing more than the particular path-tracing algorithm that happens to be implemented. Any algorithm that



(a) *Diffuse* ***(b)*** *Glossy* ***(c)*** *Specular*

Figure 4.5: *The BSDF comes in many different shapes and forms. We illustrate three examples: diffuse materials* ***(a)*** *scatter light more-or-less uniformly into all direction (e.g. wall paint), glossy materials* ***(b)*** *tend to scatter light into a narrow forward-facing cone (e.g. rough metal), and specular materials* ***(c)*** *perfectly preserve the light's angle with the surface normal (Dirac-delta function, illustrated by the light-blue arrow; e.g. smooth metal).*

operates in primary sample space therefore abstracts away the implementation details of the underlying path-tracing algorithm as a black-box function. This is a tremendous advantage over more traditional algorithms in terms of implementation complexity and code reuse, because a primary-sample-space algorithm works *out of the box* with *any* existing path tracer.

The most popular approach that operates in primary sample space is primary-sample-space Metropolis light transport (PSSMLT) [Kelemen et al. 2002] which mutates the primary samples z, as opposed to path-space Metropolis light transport [Veach and Guibas 1997] which directly mutates the path $\bar{\mathbf{x}}$. PSSMLT is known for its ease of implementation, which made it the foundation of several advanced algorithms in the following years [Kitaoka et al. 2009; Hachisuka and Jensen 2011; Hachisuka et al. 2014; Li et al. 2015; Šik et al. 2016; Otsu et al. 2017; Bitterli et al. 2017].

Later, in Chapter 8, we will utilize deep neural networks to *learn* importance sampling in primary sample space. Concurrently with our work, Guo et al. [2018] and Zheng and Zwicker [2018] also investigated this avenue.

4.6 The Bidirectional Scattering Distribution Function

The "bidirectional scattering distribution function" (BSDF) describes how incident light is scattered by a surface material. Incident radiance from a single incident direction is typically scattered into multiple outgoing directions. For some materials such as glass and shiny metal the set of outgoing directions is small, whereas for other materials such as milk, wood, and stone, almost all outgoing directions are possible.

The BSDF is such a fundamental component of computer graphics that we will provide not only its definition, but also several analytic, semi-analytic, and data-driven models that correspond to various well known real-world materials.

4.6.1 Definition

The mathematical definition of the BSDF can be directly derived from the scattering equation (4.15):

$$\begin{aligned} L_\mathrm{s}(\mathbf{x},\omega_\mathrm{o}) &= \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o}) \,|\cos\gamma_\mathrm{i}|\,\mathrm{d}\Omega(\omega_\mathrm{i}) \\ \iff \quad \frac{\mathrm{d}L_\mathrm{s}(\mathbf{x},\omega_\mathrm{o})}{\mathrm{d}\Omega(\omega_\mathrm{i})} &= L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o}) \,|\cos\gamma_\mathrm{i}| \\ \iff \frac{\mathrm{d}L_\mathrm{s}(\mathbf{x},\omega_\mathrm{o})}{L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i})\,\mathrm{d}\Omega(\omega_\mathrm{i})\,|\cos\gamma_\mathrm{i}|} &= f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o})\,, \end{aligned} \tag{4.59}$$

where $\mathbf{x}$ is the scattering position, ω_i is the direction of incident radiance, and ω_o is the direction of outgoing radiance. Note, that the BSDF is a *directional density* of outgoing radiance, corresponding to the scattering of incident radiance towards more than just one outgoing direction.

The "bidirectional" in the BSDF's name stems from the curious fact that the BSDF is usually symmetric in ω_i and ω_o. This symmetry is a fundamental property of the underlying physics of light and is widely known as "Helmholtz reciprocity".

Due to energy conservation, no more than the total amount of incident radiance can be scattered away again[5]. Using Equation 4.15 and Helmholz symmetry, we can express this constraint mathematically:

$$\int_{\mathcal{S}^2} f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o})\;|\cos\gamma_\mathrm{i}|\;\mathrm{d}\Omega(\omega_\mathrm{i}) \;\leq 1,\;\forall\mathbf{x}\in\partial\mathcal{V},\forall\omega_\mathrm{o}\in\mathcal{S}^2\,, \tag{4.60}$$

$$\int_{\mathcal{S}^2} f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o})\,|\cos\gamma_\mathrm{o}|\,\mathrm{d}\Omega(\omega_\mathrm{o}) \leq 1,\;\forall\mathbf{x}\in\partial\mathcal{V},\forall\omega_\mathrm{i}\in\mathcal{S}^2\,. \tag{4.61}$$

Furthermore, although theoretically interesting, negative energies are typically not permitted, and therefore the BSDF is constrained to be positive:

$$f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\rightarrow\omega_\mathrm{o}) \geq 0,\;\forall\mathbf{x}\in\partial\mathcal{V},\,\forall\omega_\mathrm{i},\omega_\mathrm{o}\in\mathcal{S}^2\,. \tag{4.62}$$

[5]Note, that this is in contrast to neutron transport, where additional energy can be introduced into the system due to the splitting of atomic nuclei. Photons in the visible spectrum do not have enough energy to split nuclei.

BSDF versus BRDF and BTDF. When reading the above definition of the BSDF one might wonder why it is defined over the entire sphere of outgoing directions, rather than only the upper hemisphere. This is because the BSDF is not only used to model *reflection* (corresponding to the upper hemisphere), but also to model *transmission* (corresponding to the lower hemisphere). Historically, reflection and transmission were modeled by four separate functions, the B**R**DF (bidirectional reflectance distribution function) and the B**T**DF (bidirectional transmittance distribution function), each defined twice, once per side of the surface. However, mostly for convenience, it eventually became common practice to model both reflection and transmission using a single function.

In the following sections, we present (and in some cases derive) several commonly used BSDF models, including most of the ones shown in rendered images within the later chapters of this dissertation.

4.6.2 Diffuse BSDF

The diffuse (or Lambertian) BSDF is one of the most simple-to-define BSDFs. It describes a material without a transmissive component (i.e. a BRDF) that reflects light uniformly into all directions; see Figure 4.5(a). Mathematically, the diffuse BSDF is therefore a constant

$$f_s(\mathbf{x}, \omega_i \to \omega_o) = \frac{\alpha}{\pi}, \tag{4.63}$$

where $\alpha \in [0, 1]$ controls the fraction of incident light that is reflected rather than absorbed: an α of 0 describes a material that absorbs all incident light (black), whereas an α of 1 describes a material that reflects all incident light (white). The factor $1/\pi$ ensures that the energy conservation requirement of Equation 4.61 is kept. This can easily be verified by calculating the fraction of reflected radiance by integration:

$$\int_{\mathcal{S}^2} f_s(\mathbf{x}, \omega_i \to \omega_o) \, |\cos \gamma_o| \, \mathrm{d}\Omega(\omega_o) = \frac{\alpha}{\pi} \int_{\mathcal{S}^2} |\cos \gamma_o| \, \mathrm{d}\Omega(\omega_o) = \alpha \,. \tag{4.64}$$

The diffuse BSDF is a reasonable model for objects with a "flat" appearance, such as painted walls, paper, and wood. It is worth noting, however, that the diffuse model suboptimally captures grazing-angle behavior of many realistic materials, because the model inadequately captures the effect of micro-scale structure. We refer the interested reader to the more accurate Oren-Nayar model [Oren and Nayar 1994], which is a widely-used generalization of Lambert's model.

4.6.3 Mirror BSDF

Another very common class of materials exhibit mirror-like reflections. In physics, mirror reflections (see Figure 4.5(c)) are often defined implicitly by the constraint that the incident and outgoing directions of radiance form the same angle with the surface normal and lie within the same plane. This constraint is expressed mathematically, by enforcing that the mean vector of ω_o and ω_i coincides with the surface normal

$$N(\mathbf{x}) = \frac{\omega_\mathrm{i} + \omega_\mathrm{o}}{2\cos\gamma_\mathrm{i}}\,. \tag{4.65}$$

The above constraints removes *all* degrees of freedom, permitting only a single valid ω_o for any given ω_i. Therefore, all reflected radiance is concentrated into a single direction:

$$f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o}) = \frac{\delta(\omega_\mathrm{o} - \omega_\mathrm{r})}{\cos\gamma_\mathrm{i}}\,, \tag{4.66}$$

$$\omega_\mathrm{r} = 2\,N(\mathbf{x})\cos\gamma_\mathrm{i} - \omega_\mathrm{i}\,, \tag{4.67}$$

where ω_r is the reflected direction corresponding to ω_i and $\delta(x)$ is the Dirac-delta which, conceptually, is zero everywhere but at $x = 0$ and integrates to 1[6].

It is easy to show that the above mirror BSDF reflects all energy:

$$\int_{\mathcal{S}^2} f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o})\,|\cos\gamma_\mathrm{o}|\,\mathrm{d}\Omega(\omega_\mathrm{o}) = \int_{\mathcal{S}^2} \frac{\delta(\omega_\mathrm{o} - \omega_\mathrm{r})}{\cos\gamma_\mathrm{i}}|\cos\gamma_\mathrm{o}|\,\mathrm{d}\Omega(\omega_\mathrm{o}) = 1\,. \tag{4.68}$$

In computer graphics literature, we refer to all mirror-like BSDFs (usually those that involve Dirac-delta functions) as "specular" BSDFs. Although specular BSDFs are responsible for many visually interesting phenomena, they also often complicate rendering algorithms due to the inherent difficulty of deterministically and efficiently satisfying multiple constraints such as Equation 4.65. This difficulty is exacerbated if the to-be-rendered virtual scene exhibits light transport with many specular interreflections.

4.6.4 Smooth Dielectric BSDF

Figure 4.6 shows a smooth (that is, the material surface is microscopically flat) dielectric BSDF, an example of a BSDF that has both a reflective f_r and

[6] Formally, the Dirac-delta function is a *generalized* function that is defined purely by its integral and always is associated with a specific measure. Integrated over this measure, it is 1 if the integration domain contains the zero-element, and 0 otherwise.

a transmissive f_t component, the sum of which yields f_s. Many translucent materials encountered in the real world are dielectric such as water, glass, diamond, sugar, and even air.

Snell's Law. Both the reflective and the transmissive component are—like the mirror BSDF—delta functions. In fact, the reflective component *is* a mirror reflection. The transmissive component, on the other hand, has a curious property: it deflects light as it enters and leaves the material, referred to as "refraction", where the angle of deflection depends on the "indices of refraction" of the source material η_i and the destination material η_t. The index of refraction is a scalar $\eta \geq 1$, with $\eta = 1$ only being attained by vacuum. The exact relationship of the angles γ_i and γ_t on the indices of refraction is given by Snell's law

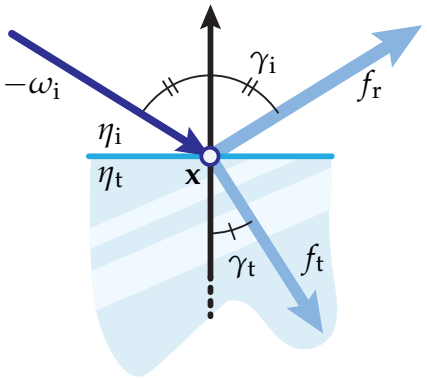


Figure 4.6: *Dielectric BSDF.*

$$\frac{\eta_i}{\eta_t} = \frac{\sin\gamma_t}{\sin\gamma_i} = \frac{c_t}{c_i}, \tag{4.69}$$

where c_i and c_t are the speed of the light in the respective material, which is inversely proportional to the material's refractive index. This relationship hints at the cause behind Snell's law: it actually is a direct consequence (not shown) of *Fermat's principle*, which dictates that light always follows the path of *least time*.

The index of refraction of a material not only depends on the material itself, but also on the wavelength of light, meaning that different colors of light are refracted into different directions. This is why prisms made out of dielectric materials are able to decompose white light into its individual colors.

The Fresnel Equations. With the direction of refraction being known, we require the ratio between reflected and refracted radiance such that we can mathematically formulate the dielectric BSDF. This ratio can be directly derived from the wave nature of light[7]. Augustin-Jean Fresnel derived the "Fresnel equations" that describe the fraction of reflected radiance for light

[7] Although we concern ourselves with geometric optics at a macro scale, in some situations—like this one—more accurate physical models can be "baked" into localized functions such as the BSDF.

that is polarized perpendicular to the surface (i.e. its electromagnetic wave is aligned perpendicular to the surface)

$$R^{\top}(\omega_{\mathrm{i}}) = \left| \frac{\eta_{\mathrm{t}} \cos\gamma_{\mathrm{i}} - \eta_{\mathrm{i}} \cos\gamma_{\mathrm{t}}}{\eta_{\mathrm{t}} \cos\gamma_{\mathrm{i}} + \eta_{\mathrm{i}} \cos\gamma_{\mathrm{t}}} \right|^2 \tag{4.70}$$

and parallel to the surface

$$R^{\parallel}(\omega_{\mathrm{i}}) = \left| \frac{\eta_{\mathrm{t}} \cos\gamma_{\mathrm{t}} - \eta_{\mathrm{i}} \cos\gamma_{\mathrm{i}}}{\eta_{\mathrm{t}} \cos\gamma_{\mathrm{t}} + \eta_{\mathrm{i}} \cos\gamma_{\mathrm{i}}} \right|^2 . \tag{4.71}$$

The modeling of polarized light is out of the scope of this dissertation, so we assume photons are polarized in either direction with equal probability, resulting in

$$R(\omega_{\mathrm{i}}) = \frac{1}{2}\Big(R^{\top}(\omega_{\mathrm{i}}) + R^{\parallel}(\omega_{\mathrm{i}}) \Big) , \tag{4.72}$$

with $T(\omega_{\mathrm{i}}) = 1 - R(\omega_{\mathrm{i}})$ being the corresponding transmitted portion of incident radiance.

Combining Equations (4.69) and (4.72), we finally obtain the dielectric BSDF

$$f_{\mathrm{s}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) = f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) + f_{\mathrm{t}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) , \tag{4.73}$$

$$f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) = R(\omega_{\mathrm{i}}) \frac{\delta(\omega_{\mathrm{o}} - \omega_{\mathrm{r}})}{\cos\gamma_{\mathrm{i}}} , \tag{4.74}$$

$$f_{\mathrm{t}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) = T(\omega_{\mathrm{i}}) \frac{\delta(\omega_{\mathrm{o}} - \omega_{\mathrm{t}})}{\cos\gamma_{\mathrm{i}}} \frac{\eta_{\mathrm{t}}^2}{\eta_{\mathrm{i}}^2} , \tag{4.75}$$

where an additional factor of $\eta_{\mathrm{t}}^2/\eta_{\mathrm{i}}^2$ sneaked in. The next paragraph explains this factor.

Non-Symmetry due to Refraction. Recall, that we claimed, that—according to Helmholtz reciprocity—BSDFs are *usually* symmetric. Dielectric transmission is an exception to this symmetry. A direct consequence of the directionally dependent deflection is that (solid-)angles are compacted or stretched as the index of refraction increases or decreases, respectively. Radiance, being a density w.r.t. solid-angle, consequently changes non-symetrically as its differential solid-angle is altered. Veach [1997] derived a generalized symmetry relationship that incorporates this behavior:

$$\frac{f_{\mathrm{s}}(\mathbf{x}, \omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}})}{\eta_{\mathrm{o}}^2} = \frac{f_{\mathrm{s}}(\mathbf{x}, \omega_{\mathrm{i}} \leftarrow \omega_{\mathrm{o}})}{\eta_{\mathrm{i}}^2} . \tag{4.76}$$

It is worth noting, that in the case of *spectral* radiance, due to additional compaction and stretching along wavelength, the relationship becomes

$$\frac{f_s(\mathbf{x}, \omega_i \to \omega_o)}{\eta_o^3} = \frac{f_s(\mathbf{x}, \omega_i \leftarrow \omega_o)}{\eta_i^3}. \tag{4.77}$$

4.6.5 Smooth Conductor BSDF

Smooth conductors (i.e. conductive materials with a microscopically smooth surface) also behave mirror-like. Like dielectrics, conductors have an index of refraction, which is, however, complex-valued. Conductors—unlike dielectrics—do not transmit light, making them effectively a colored mirror, the color of which is a function of their index of refraction (via the Fresnel equations)

$$f_s(\mathbf{x}, \omega_i \to \omega_o) = R(\omega_i) \frac{\delta(\omega_o - \omega_r)}{\cos \gamma_i}. \tag{4.78}$$

The Conductor BSDF is primarily useful for modeling the appearance of metals.

4.6.6 Rough Surfaces

The dielectric and conductor BSDFs from above were described as referring to "smooth" surfaces. At first thought, it may seem that such a smooth-surface model would be perfectly sufficient, because roughness could be explicitly captured by the surface geometry, but—even though there exist techniques that to some degree work in such a way [Jakob et al. 2014b; Yan et al. 2014]—it is usually more much convenient to model small-scale roughness statistically inside of the BSDF. Such a statistical model *averages* a BSDF over possible microgeometry orientations. Averaging over the Dirac-delta BSDFs such as the dielectric or the conductor recovers a finite non-Dirac BSDF as illustrated in Figure 4.5(b). In this dissertation, we refer to BSDFs with this general shape as "glossy". In the following sections, we will describe a set of historically popular glossy BSDF models.

4.6.7 Blinn-Phong BSDF

Phong [1975] was the first to phenomenologically model glossy reflection. He discovered, that reflected radiance of specular surfaces with micro-roughness

tends to cluster around the reflected direction ω_r prescribed by the macro normal, which he modeled by a cosine raised to some controllable power

$$f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) = \alpha \, c_\mathrm{phong} \left(\omega_\mathrm{r} \cdot \omega_\mathrm{o}\right)^\beta , \tag{4.79}$$

$$c_\mathrm{phong} = \frac{1}{\int_{\mathcal{S}^2} \left(\omega_\mathrm{r} \cdot \omega_\mathrm{o}\right)^\beta \cos\gamma_\mathrm{o} \, \mathrm{d}\Omega(\omega_\mathrm{o})} , \tag{4.80}$$

where the power β controls the concentration of light around ω_r (larger β means a higher concentration, i.e. a smoother surface), c_phong ensures energy conservation, and $\alpha \in [0, 1]$ is the familiar surface albedo from the diffuse BSDF.

Blinn [1977] later proposed a refined version of Phong's BSDF that is defined in terms of the half vector ω_h between ω_i and ω_o

$$f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) = \alpha \, c_\mathrm{blinn} \left(N(x) \cdot \omega_\mathrm{h}\right)^\beta , \tag{4.81}$$

$$c_\mathrm{blinn} = \frac{1}{\int_{\mathcal{S}^2} \left(N(x) \cdot \omega_\mathrm{h}\right)^\beta \cos\gamma_\mathrm{o} \, \mathrm{d}\Omega(\omega_\mathrm{o})} , \tag{4.82}$$

$$\omega_\mathrm{h} = \frac{\omega_\mathrm{i} + \omega_\mathrm{o}}{\|\omega_\mathrm{i} + \omega_\mathrm{o}\|} , \tag{4.83}$$

where c_blinn again ensures energy conservation. This model is based on Torrance and Sparrow's microfacet model [Torrance and Sparrow 1967] and in Blinn's work was used in computer graphics for the first time. Although Blinn suggested the additional usage of a Fresnel term for improved accuracy, many practitioners omit it when implementing the Blinn-Phong model.

4.6.8 Torrance-Sparrow Microfacet BSDFs

Rather than relying on approximations such as the Blinn-Phong model, it is sometimes possible to model rough surfaces directly using the Torrance-Sparrow model [Torrance and Sparrow 1967]. After being popularized by Walter et al. [2007], the use of this model in computer graphics has undergone a significant advancement [Heitz 2014; Heitz et al. 2015, 2016b; Jakob et al. 2014b; Belcour and Barla 2017; Dupuy et al. 2016]. In the following paragraphs we briefly outline the basic theory behind the Torrance-Sparrow model; we refer the interested reader to the above references for a more in-depth treatment.

The Microfacet Distribution. The Torrance-Sparrow model defines the micro-scale surface as a collection of randomly oriented differential facets

with normal $N_\mu \in \mathcal{S}^{2^+}(\mathbf{x})$. The set $\mathcal{S}^{2^+}(\mathbf{x})$ denotes the upper hemisphere at $\mathbf{x} \in \partial\mathcal{V}$, i.e. the set of all directions satisfying $N(\mathbf{x}) \cdot N_\mu \geq 0$. This means, that we only permit upward-facing (relative to the macro surface) micro normals. We characterize a collection of microfacets by the distribution of their normal vectors $D(\mathbf{x}, N_\mu)$. The above cases of smooth surfaces therefore corresponds to

$$D(\mathbf{x}, N_\mu) = \delta\big(N(\mathbf{x}) - N_\mu\big)\,. \tag{4.84}$$

To develop a precise definition of the surface consisting of microfacets, we postulate that all microfacets have the same differential surface area $\mathrm{d}A(\mathbf{x})$. Our goal is to define $D(\mathbf{x}, N_\mu)$ as the orientation of a microfacet at a uniformly random position on the *macro*surface. We assume that the microgeometry is continuous and non-overlapping, yielding the differential macro surface area covered by a microfacet

$$\mathrm{d}A^\top_\omega(\mathbf{x}) = \mathrm{d}A(\mathbf{x})\big(N(\mathbf{x}) \cdot N_\mu\big)\,. \tag{4.85}$$

The distribution of microfacet normals must satify two constraints: it must be a valid probability density (i.e. integrate to 1), and the microfacets must cover exactly one differential piece of macro surface area to make the following derivations easier. Using Equation 4.85, we can express the latter constraint as

$$\begin{aligned} &\int_{\mathcal{S}^2} D(\mathbf{x}, \omega_\mathrm{h})\,\mathrm{d}A^\top_\omega(\mathbf{x})\,\mathrm{d}\Omega(\omega_\mathrm{h}) = \mathrm{d}A(\mathbf{x}) \\ \underset{4.85}{\Longleftrightarrow} &\int_{\mathcal{S}^2} D(\mathbf{x}, \omega_\mathrm{h})\big(N(\mathbf{x}) \cdot N_\mu\big)\,\mathrm{d}\Omega(\omega_\mathrm{h}) = 1\,. \end{aligned} \tag{4.86}$$

This constraint is convenient, because if we define $D(\mathbf{x}, \omega_\mathrm{h})$ as a density per *projected* solid-angle, it *also* ensures that $D(\mathbf{x}, \omega_\mathrm{h})$ is a valid probability density. Rewriting the above integral, we obtain the definition of $D(\mathbf{x}, \omega_\mathrm{h})$:

$$D(\mathbf{x}, \omega_\mathrm{h}) = \frac{\mathrm{d}A(\mathbf{x})}{\mathrm{d}\Omega(\omega_\mathrm{h})\,\mathrm{d}A^\top_\omega(\mathbf{x})} = \frac{1}{\mathrm{d}\Omega(\omega_\mathrm{h})\big(N(\mathbf{x}) \cdot N_\mu\big)}\,. \tag{4.87}$$

Using this definition, we can formulate the Torrance-Sparrow microfacet BSDF as the expected reflected radiance given N_μ sampled from $D(\mathbf{x}, N_\mu)$

$$f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) = \int_{\mathcal{S}^2} G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, J_i\, f_\mu(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o})\, J_o\, D(\mathbf{x}, N_\mu)\big(N(\mathbf{x}) \cdot N_\mu\big)\,\mathrm{d}\Omega(N_\mu)\,, \tag{4.88}$$

where $f_\mu(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o})$ is the BSDF of the microfacets (also called micro BSDF), J_i and J_o are the respective absolute Jacobian determinants of transforming

incident radiance from the macro scale to the micro scale and back (respective ratios of foreshortening terms)

$$J_i = \left| \frac{N_\mu \cdot \omega_\mathrm{i}}{N(\mathbf{x}) \cdot \omega_\mathrm{i}} \right|, \quad J_o = \left| \frac{N_\mu \cdot \omega_\mathrm{o}}{N(\mathbf{x}) \cdot \omega_\mathrm{o}} \right|, \tag{4.89}$$

and $G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})$, called "geometry-term" or "shadowing-masking" term, the values of which lie in $[0, 1]$, captures self-occlusion of the micro geometry.

Rough Conductor. It is possible to obtain a closed-form BSDF for rough conductor surfaces by plugging the smooth conductor BSDF (4.78) into Equation 4.88:

$$f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \underset{4.88}{\underset{4.78}{=}} \int_{\mathcal{S}^2} G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, J_i\, R(\omega_\mathrm{i}) \frac{\delta(\omega_\mathrm{o} - \omega_\mathrm{r})}{N_\mu \cdot \omega_\mathrm{i}}\, J_o\, D(\mathbf{x}, N_\mu) \big(N(\mathbf{x}) \cdot N_\mu\big)\, \mathrm{d}\Omega(N_\mu)\,. \tag{4.90}$$

To resolve the Dirac-delta function, we need to express the integral with respect to $\mathrm{d}\Omega(\omega_\mathrm{r})$. Using

$$\mathrm{d}\Omega(N_\mu) = \frac{N(\mathbf{x}) \cdot N_\mu}{4|\omega_\mathrm{r} \cdot N_\mu|}\, \mathrm{d}\Omega(\omega_\mathrm{r}) \tag{4.91}$$

and

$$\omega_\mathrm{o} = \omega_\mathrm{r} \iff \omega_\mathrm{h} = N_\mu\,, \tag{4.92}$$

we can simplify to obtain

$$\begin{aligned} f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) &\underset{4.91}{=} \int_{\mathcal{S}^2} G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, J_i\, R(\omega_\mathrm{i}) \frac{\delta(\omega_\mathrm{o} - \omega_\mathrm{r})}{|N_\mu \cdot \omega_\mathrm{i}|}\, J_o\, D(\mathbf{x}, N_\mu) \frac{N(\mathbf{x}) \cdot N_\mu}{4|\omega_\mathrm{r} \cdot N_\mu|}\, \mathrm{d}\Omega(\omega_\mathrm{r}) \\ &\underset{4.92}{=} \frac{G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, J_i\, R(\omega_\mathrm{i})\, J_o\, D(\mathbf{x}, \omega_\mathrm{h}) \left(N(\mathbf{x}) \cdot \omega_\mathrm{h}\right)}{4|\omega_\mathrm{h} \cdot \omega_\mathrm{i}|\, |\omega_\mathrm{o} \cdot \omega_\mathrm{h}|} \\ &= \frac{G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, R(\omega_\mathrm{i})\, D(\mathbf{x}, \omega_\mathrm{h})}{4\,|\cos\gamma_\mathrm{i}\, \cos\gamma_\mathrm{o}|}\,. \end{aligned} \tag{4.93}$$

Rough Dielectric. The exact same derivations producing Equation 4.93 can be applied to derive the macro-scale BSDF for dielectric reflection. The reflection component is derived using the exact same formulas, whereas the transmission component requires using Equation 4.75 in place of Equation 4.78 within Equation 4.88. Omitting the details, this produces

$$f_\mathrm{t}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \underset{4.88}{\underset{4.75}{=}} \frac{|\omega_\mathrm{i} \cdot \omega_\mathrm{h}|\, |\omega_\mathrm{o} \cdot \omega_\mathrm{h}|}{|\omega_\mathrm{i} \cdot N(\mathbf{x})|\, |\omega_\mathrm{o} \cdot N(\mathbf{x})|} \frac{\eta_\mathrm{o}^2\, G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, T(\omega_\mathrm{i})\, D(\mathbf{x}, \omega_\mathrm{h})}{\big(\eta_\mathrm{i}(\omega_\mathrm{i} \cdot \omega_\mathrm{h}) + \eta_\mathrm{o}(\omega_\mathrm{o} \cdot \omega_\mathrm{h})\big)^2}\,. \tag{4.94}$$

The interested reader can find the full derivation in the work of Walter et al. [2007].

Choices for D and G. The physical plausibility and therefore the quality of rendered images hinges on using a realistic microfacet distribution $D(\mathbf{x}, \omega_\mathrm{h})$ and an accurate corresponding shadowing-masking term $G(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})$. Deriving appropriate formulas for the microfacet distribution and the shadowing-masking term is unfortunately out of the scope of this dissertation. We refer the reader to the work of Walter et al. [2007] for several such derivations and a more detailed overview of the theory.

Energy Loss. The Torrance-Sparrow model treats self-occlusion of the micro geometry as absorbtion of energy, but in reality light may scatter instead of being absorbed. This amounts to modeling only the *singly* scattered light due to microfacets but ignoring any higher order scattering. Kelemen and Szirmay-Kalos [2001] were the first to propose re-introducing an approximation of the lost energy in the form of an additive Lambertian (uniform) component. They argued, that multiply scattered light approaches uniformity as the number of scattering interactions approaches infinity. Jakob et al. [2014a] extended the aforementioned approach to correctly handle reciprocity.

More recently, Heitz et al. [2016b] proposed to instead model high-order microfacet scattering with a random walk, the characteristics of which could directly be derived from previous advances in microflake models [Jakob et al. 2010; Heitz 2014; Heitz et al. 2015]. The random walk is much more accurate than the uniform approximation, but on the flipside introduces additional noise. It is also difficult to integrate with bi-directional path-tracing techniques.

4.6.9 Data-Driven BSDFs.

Although the volume of research on analytic and semi-analytic BSDF models is vast, there also exist purely data-driven approaches. Research on data-driven BSDFs ranges from measurement devices [Nicodemus et al. 1992; White et al. 1998; Ghosh et al. 2007; Marschner et al. 1999, 2000; Mattison et al. 1998; Ngan et al. 2005; Ren and Zhao 2010; Ward 1992] over parameterizations [Löw et al. 2012; Ngan et al. 2005; Holzschuch and Pacanowski 2017; Bagher et al. 2012, 2016; Dupuy and Jakob 2018] to publicly available databases [Marschner et al. 1999; Matusik et al. 2003; Ngan et al. 2005; Löw et al. 2012; Apian-Bennewitz 2013; Filip et al. 2014; Filip and Vávra 2014; Dupuy and Jakob 2018].

Despite of the vast body of research on the topic, the unfortunate reality in computer graphics is that only the MERL database [Matusik et al. 2003],

the UTIA databases [Filip et al. 2014; Filip and Vávra 2014], and the newer (steadily growing) RGL Material Database [Dupuy and Jakob 2018] have sufficient directional resolution for the BSDFs to be useful in a rendering system. Due to their particular measurement setups, the MERL database does not contain anisotropic materials, the UTIA does not contain high-frequency reflectances, and both of them are limited to RGB (non-spectral) data. The RGL database overcomes these limitations at the expense of requiring considerably longer time for each measured material, and, at the time of writing this dissertation, contains a smaller amount of materials than the MERL and the UTIA databases. None of the aforementioned databases contains measurements of transmissive materials, i.e. all of them are limited to capturing only the BRDF component.

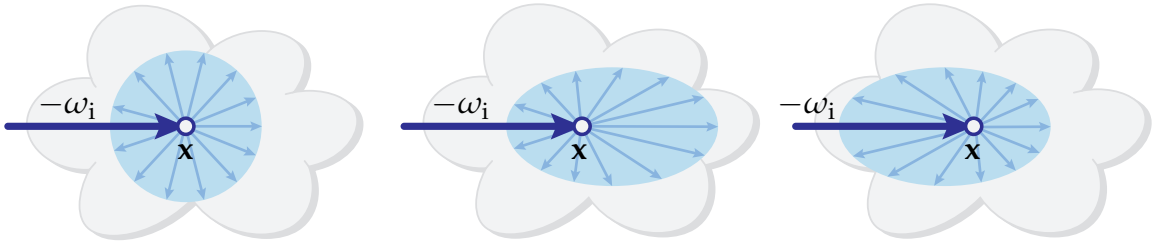
Regardless of these limitations, the above databases of physical material properties are not only immensely useful for verifying theoretical models (such as the microfacet model), but are also invaluable in synthesizing images with known physical correctness (up to measurement and quantization error). In rendering systems it is often possible to blend between measured materials to augment the range of tabulated data further.

4.7 The Phase Function

In this section, we mathematically define the "phase function" which captures light-particle interactions within participating media. Similarly to how the definition of the BSDF can be derived from the scattering equation (4.59), the phase function's definition can be derived from the in-scattering term (4.21):

$$
\begin{aligned}
(\omega \cdot \nabla) L_{\text{in}}(\mathbf{x}, \omega_{\text{o}}) &= \sigma_{\text{s}}(\mathbf{x}) \int_{\mathcal{S}^2} f_{\text{p}}(\mathbf{x}, \omega_{\text{i}} \to \omega_{\text{o}}) L(\mathbf{x}, \omega_{\text{i}}) \, \mathrm{d}\Omega(\omega_{\text{i}}) \\
\frac{(\omega \cdot \nabla)}{\sigma_{\text{s}}(\mathbf{x})} L_{\text{in}}(\mathbf{x}, \omega_{\text{o}}) &= \int_{\mathcal{S}^2} f_{\text{p}}(\mathbf{x}, \omega_{\text{i}} \to \omega_{\text{o}}) L(\mathbf{x}, \omega_{\text{i}}) \, \mathrm{d}\Omega(\omega_{\text{i}}) \\
\iff \quad \frac{(\omega \cdot \nabla)}{\sigma_{\text{s}}(\mathbf{x})} \frac{\mathrm{d} L_{\text{in}}(\mathbf{x}, \omega_{\text{o}})}{\mathrm{d}\Omega(\omega_{\text{i}})} &= f_{\text{p}}(\mathbf{x}, \omega_{\text{i}} \to \omega_{\text{o}}) L(\mathbf{x}, \omega_{\text{i}}) \\
\iff \frac{(\omega \cdot \nabla)}{\sigma_{\text{s}}(\mathbf{x})} \frac{\mathrm{d} L_{\text{in}}(\mathbf{x}, \omega_{\text{o}})}{L(\mathbf{x}, \omega_{\text{i}}) \, \mathrm{d}\Omega(\omega_{\text{i}})} &= f_{\text{p}}(\mathbf{x}, \omega_{\text{i}} \to \omega_{\text{o}}) \, .
\end{aligned}
\tag{4.95}
$$

To understand the phase function, it helps to first consider the ratio $\frac{(\omega \cdot \nabla)}{\sigma_{\text{s}}(\mathbf{x})} L_{\text{in}}(\mathbf{x}, \omega_{\text{o}})$. Recall, that $(\omega \cdot \nabla) L_{\text{in}}(\mathbf{x}, \omega_{\text{o}})$ describes a 1-dimensional radiance density $[\mathrm{W\,m^{-3}\,sr^{-1}}]$ and $\sigma_{\text{s}}(\mathbf{x})$ a 1-dimensional number density $[\mathrm{m^{-1}}]$ of particles. The aforementioned ratio therefore has units of $[\mathrm{W\,m^{-2}\,sr^{-1}}]$ and refers to radiance *per particle*. It follows, that the phase function describes



(a) *Isotropic* $g = 0$ **(b)** *Forward scattering* $g \in (0, 1]$ **(c)** *Backward scattering* $g \in [-1, 0)$

Figure 4.7: *Phase functions that are always oriented along ω_i can either predominantly scatter towards the front* ***(b)****, characterized by a positive mean cosine, or towards the back* ***(c)*** *(negative mean cosine). The special case of uniform scattering in all directions* ***(a)*** *(being a sufficient but not necessary condition for having a mean cosine of zero) is referred to as "isotropic"; all other cases are "anisotropic".*

the *directional scattering profile* of *individual* light-particle interactions; see Figure 4.7 for an illustration.

Since we do not permit negative energies, we constrain the phase function to be positive

$$f_p(\mathbf{x}, \omega_i \to \omega_o) \geq 0, \ \forall \mathbf{x} \in \mathcal{V}, \forall \omega_i, \omega_o \in \mathcal{S}^2 . \tag{4.96}$$

Again, like with the BSDF, we typically assume Helmholz reciprocity, i.e. f_p has the same value when swapping ω_i and ω_o. Unlike the BSDF, however, where energy-dissipation is part of the distribution, the phase function must always integrate to 1 (i.e. conserve energy)

$$\int_{\mathcal{S}^2} f_p(\mathbf{x}, \omega_i \to \omega_o) \, \mathrm{d}\Omega(\omega_i) = 1, \ \forall \mathbf{x} \in \mathcal{V}, \forall \omega_o \in \mathcal{S}^2 , \tag{4.97}$$

$$\int_{\mathcal{S}^2} f_p(\mathbf{x}, \omega_i \to \omega_o) \, \mathrm{d}\Omega(\omega_o) = 1, \ \forall \mathbf{x} \in \mathcal{V}, \forall \omega_i \in \mathcal{S}^2 , \tag{4.98}$$

with fractional absorbtion being explicitly modeled by the albedo $\alpha \in [0, 1]$ as part of the RTE, as previously seen in Section 4.4.1.

4.7.1 Anisotropy of Phase Functions

Most phase function models that are used in practice are rotationally symmetric around ω_i and independent from both ω_i and $\mathbf{x}$. Mathematically, this amounts to the phase function being a function of only the angle between ω_i and ω_o

$$f_p(\mathbf{x}, \omega_i \to \omega_o) = f_p(-\omega_i \cdot \omega_o) = f_p(\cos \gamma) , \tag{4.99}$$

where the angle is by convention parameterized by its cosine $\cos\gamma = -\omega_\mathrm{i} \cdot \omega_\mathrm{o}$. These kinds of phase functions are a suitable model when the infinitesimal particles constituting the participating medium are either rotationally symmetric themselves, or are oriented uniformly at random.

Such phase functions can be broadly categorized by their tendency to scatter light forward, backward, or in no particular direction at all. This tendency is quantified easily by the phase-function-weighted cosine

$$g = 2\pi \int_{-1}^{1} \cos\gamma \, f_\mathrm{p}(\cos\gamma) \, \mathrm{d}\cos\gamma \,, \tag{4.100}$$

which can also be interpreted as the expected cosine upon randomly scattering proportional to f_p. Figure 4.7 illustrates the conceptual difference between different values of g.

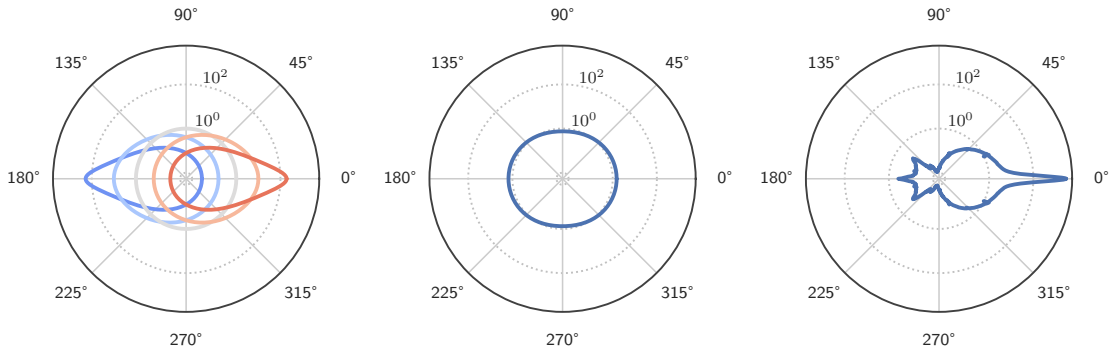
4.7.2 The Henyey-Greenstein Phase Function

The Henyey-Greenstein (HG) phase function [Henyey and Greenstein 1941] was originally introduced as a model of light scattering by interstellar dust. Since then, the HG phase function became widely adopted in multiple fields as an analytic model for all kinds of volumetric scattering [Pegoraro 2016; Novák et al. 2018].

$$f_\mathrm{p}(\cos\gamma, g) = \frac{1}{4\pi} \frac{1 - g^2}{\left(1 + g^2 - 2g\cos\gamma\right)^{3/2}} \,, \tag{4.101}$$

The HG phase function is actually a *family* of phase functions that is parameterized by the aforementioned mean cosine g, which makes it an especially suitable choice when one wants to prescribe anisotropy explicitly. It is easy to verify that

$$\begin{aligned}
g &\underset{4.99}{=} 2\pi \int_{-1}^{1} \cos\gamma \, f_\mathrm{p}(\cos\gamma, \hat{g}) \, \mathrm{d}\cos\gamma \\
&= 2\pi \int_{-1}^{1} \cos\gamma \, \frac{1}{4\pi} \frac{1 - \hat{g}^2}{\left(1 + \hat{g}^2 - 2\hat{g}\cos\gamma\right)^{3/2}} \, \mathrm{d}\cos\gamma \\
&= \frac{1 - \hat{g}^2}{2} \int_{-1}^{1} \cos\gamma \left(1 + \hat{g}^2 - 2\hat{g}\cos\gamma\right)^{-3/2} \mathrm{d}\cos\gamma \\
&= \frac{1 - \hat{g}^2}{2} \, \frac{\hat{g}^2 - \hat{g}\cos\gamma + 1}{\hat{g}^2 \sqrt{\hat{g}^2 - 2\hat{g}\cos\gamma + 1}} \Big|_{-1}^{1} \\
&= \frac{1}{2}\left(\frac{\hat{g}^3 + 1}{\hat{g}^2} + \frac{\hat{g}^3 - 1}{\hat{g}^2}\right) = \frac{\hat{g}^3}{\hat{g}^2} = \hat{g} \,.
\end{aligned} \tag{4.102}$$



(a) *Henyey-Greenstein* ***(b)*** *Rayleigh* ***(c)*** *Lorenz-Mie*

Figure 4.8: *Henyey-Greenstein* ***(a)****, Rayleigh* ***(b)****, and Lorenz-Mie* ***(c)*** *phase functions, plotted as a function of γ with logarithmic scale. The shown instances of the HG phase function are parameterized with $g \in \{-0.9, -0.6, 0, 0.6, 0.9\}$ (from blue to red). The shown Lorenz-Mie phase function was numerically computed using MiePlot [Laven 2017] for spherical scatterers with gamma-distributed radii (in µm) with shape $k = 2$ and scale $\theta = 2$, and averaged across three different color channels: red (650 nm), green (530 nm), and blue (450 nm).*

It is also worth noting that for $g = 0$, the HG phase function becomes the *isotropic* phase function, which scatters light uniformly in all directions

$$f_p(\cos\gamma, 0) = \frac{1}{4\pi} \frac{1-0}{(1+0-0\cos\gamma)^{3/2}} = \frac{1}{4\pi}. \tag{4.103}$$

This special phase function is in concept similar to the Lambertian (diffuse) BSDF (see Section 4.6), which also scatters light uniformly in all directions. Figure 4.8(a) illustrates the shape of the HG phase function for varying values of g, where the values are, from blue to red: $(-0.9, -0.6, 0, 0.6, 0.9)$.

4.7.3 Rayleigh Scattering

Rayleigh scattering [Rayleigh 1899] approximates Maxwell's equations for light interactions with particles that are much smaller than the wavelength of light (below $\lambda/10$). Rayleigh's phase function for such particles is simple:

$$f_p(\cos\gamma) = \frac{1}{4\pi}\frac{3}{4}(1+\cos^2\gamma). \tag{4.104}$$

It consists of two additive components (the squared cosine and the 1), where the cosine corresponds to forward and backward scattering (without any scattering at a right angle) of horizontally polarized light and 1 describes isotropic scattering of vertically polarized light. Since we express the phase function in a rotationally independent manner, all rotations must be integrated out,

resulting in the additive combination of horizontal and vertical polarization effects. Since $f_{\mathrm{p}}(\cos\gamma) = f_{\mathrm{p}}(\cos\gamma + \pi)$, the Rayleigh phase function has $g = 0$. We show the Rayleigh phase function in Figure 4.8(b), comparing it to the HG and the Lorenz-Mie phase functions which typically have much more pronounced anisotropic behavior.

In practice, however, light is often not polarized uniformly in all directions. This is part of the reason that the sky often has a different hue of blue at the horizon than towards the zenith: microscopic aerosol in the atmosphere that scatter light in a polarization-dependent manner. In such cases, if the Rayleigh phase function is used as a scattering model, it requires polarization-dependent weighting of its additive terms.

Rayleigh not only derived a phase function for microscopic particles, but also their scattering coefficient, which, due to the assumption of small particle scales, must consider quantum mechanical effects to be accurate. From these effects, Rayleigh derived

$$\sigma_{\mathrm{s}}(\lambda, \eta, d, \rho) = \rho \, \frac{2\pi^5}{3} \, \frac{d^6}{\lambda^4} \left(\frac{\eta^2 - 1}{\eta^2 + 2} \right)^2, \tag{4.105}$$

where λ is the wavelength of light, η is the index of refraction of the particles, d is their diameter, and ρ is their 3-dimensional number density[8]. It is precisely the wavelength dependence of the above scattering coefficient that gives our atmosphere its characteristic blue color.

4.7.4 The Lorenz-Mie Phase Function

The Lorenz-Mie phase function (developed independently by Mie [1908] and Lorenz [1890]) is the solution to Maxwell's equations for planar waves of light that are scattered by a dielectric sphere with a size comparable to the wavelength of light. Unlike more approximate solutions such as the Henyey-Greenstein (Section 4.7.2) or the Rayleigh (Section 4.7.3) phase functions, the Lorenz-Mie phase function has a very intricate shape, consisting of a strong forward-facing peak, a wide sideways-facing lobe, and three backward-facing peaks. The Lorenz-Mie phase function is especially useful to model the appearance of atmospheric clouds, consisting of tiny water droplets or ice crystals that are too big to be well approximated by Rayleigh scattering, but small enough to be affected by wave optics (around 1 to 100 microns).

[8] The factor ρ is required to transform scattering cross section $[\mathrm{m}^2]$ (which Rayleigh originally derived) to our desired 1-dimensional number density $[\mathrm{m}^{-1}]$.

In Figure 4.8(c), we show an example Lorenz-Mie phase function that was numerically computed using MiePlot [Laven 2017] for an aerosol with gamma-distributed radii (in µm) with shape $k = 2$ and scale $\theta = 2$. To obtain the final shape, we averaged phase-function profiles for three visible colors, red (650 nm), green (530 nm), and blue (450 nm), across 50 randomly sampled radii from the gamma distribution.

CHAPTER 5

Machine Learning in this Dissertation

The goal of machine learning is the automatic data-driven generation of programs. Its applications are manifold, including language translation, voice and image recognition, sound generation, robotics, self-driving cars, medicine, and many more. Machine learning nowadays truly is ubiquitous, which is a testimony of its ability to solve many problems far more effectively than any existing human-designed solution does.

5.1 Overview

The field of machine learning is often subdivided into "supervised learning", "unsupervised learning", and "reinforcement learning". The following is a brief overview over each of these sub-fields.

Supervised Learning. In supervised learning, the goal is to learn values (sometimes also referred to as labels) that correspond to a given input. This includes problems such as classification (e.g. does image x contain a cat or a dog?), regression (e.g. meaningful dense approximation of sparse data), but also certain portions of statistical modeling (e.g. what is the probability that patient x has cancer, given MRI scan y?). In general, whenever one is able to supply a machine-learning algorithm with training data that consists of desired input-output pairs, one operates in the domain of supervised learning.

Unsupervised Learning. In contrast to supervised learning, *un*supervised learning deals with automatically extracting hidden properties of unlabeled data. The field of unsupervised learning includes clustering (e.g. the genomes of which lifeforms are closely related?), signal compression (e.g. encode any given video at the highest possible quality using only x bits), and, once again, portions of statistical modeling such as density estimation (e.g. which probability density underlies natural images?). The line between unsupervised and supervised learning is often unclear: the task of *weighted* density estimation (which we introduce in detail later) has, for instance, qualities of supervised learning (the weights can be interpreted as labels) *and* of unsupervised learning (the underlying probability density is only correlated with the weights and still must be learned). Such cases are also often referred to as *semi*-supervised learning.

Reinforcement Learning. Reinforcement learning is the study of automatically navigating an agent through a (partially) unknown environment to maximize a pre-defined reward. A prime example for a reinforcement-learning problem are video games: the goal is to learn the actions of a character that maximize the score upon completion of the game. However, the applications of reinforcement learning are not limited to just games; being able to do well in games translates very well to succeeding in various real-world tasks, such as navigating self-driving cars, controlling robots, and more abstract settings such as interacting with the stock market. At first, it may seem like reinforcement learning should be classified under semi-supervised learning (the reward function can be seen as providing training labels while the agent is tasked with discovering the underlying dynamics of its world on its own), but the vastness of specialized applications, research, and algorithms arguably warrants its own category. That said, ongoing research reveals closer connections between reinforcement learning and other fields: Dahm and Keller [2018] showed, for example, that reinforcement learning can be mapped to the learning of Fredholm integral equations of the second kind, which suggests a close connection to existing (un)supervised function-approximation techniques such as regression and density estimation.

5.2 Function Approximation

In this dissertation, we are not interested in the general field of machine learning, but rather in one particular facet of it: the study of data-driven function approximation. All our contributions throughout the Chapters 6–8 use machine learning to *approximate* parts of the path integral (Chapters 6)

and/or to *importance sample* parts of the path integral (Chapters 6–8). For this purpose, we resort to weighted density estimation / regression on the class of functions $f : \mathcal{D} \to \mathbb{R}$ that map an arbitrary domain $\mathcal{D}$ to the real number line.

5.2.1 Objective

In function approximation, our goal is to match a ground-truth function $f(x) : \mathcal{D} \to \mathbb{R}$ as well as possible with a "trainable model" $h(x;\theta) : \mathcal{D} \times \mathbb{R}^M \to \mathbb{R}$. The model $h(x;\theta)$ depends on a set of "model parameters" $\theta \in \mathbb{R}^M$, the optimization of which to obtain the best possible fit to $f(x)$ is referred to as "training the model". The quality of the fit is judged with the help of a loss function $\mathcal{L}(f \,\|\, h;\theta)$ that quantifies the distance between f and h. The objective can then be framed mathematically as finding the parameters θ that minimize the loss

$$\widehat{\theta} \in \arg\min_{\theta} \mathcal{L}(f \,\|\, h;\theta)\,. \tag{5.1}$$

Although the loss $\mathcal{L}$ could in principle have any form, in this dissertation we are interested in losses that have the form of an integral over a per-data-point loss function ℓ

$$\mathcal{L}(f \,\|\, h;\theta) = \int_{\mathcal{D}} \ell\big(f(x), h(x;\theta)\big)\,\mathrm{d}\mu(x)\,. \tag{5.2}$$

It is generally impossible to evaluate the loss integral exactly, let alone optimize θ in a way that depends on it. Therefore, what is most commonly done, is to estimate the loss integral in a Monte Carlo fashion with a finite set of training data points. Assuming the training data is distributed approximately uniformly (according to μ) over the data domain allows us to formulate an approximately ideal choice of θ as those parameters that minimize a Monte Carlo estimator of the loss integral

$$\langle\widehat{\theta}\rangle \in \arg\min_{\theta} \langle\mathcal{L}(f \,\|\, h;\theta)\rangle = \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \ell\big(f(X_i), h(X_i;\theta)\big)\,. \tag{5.3}$$

Note, that if the training data is not distributed uniformly according to μ, then we must either adjust the Monte Carlo estimate

$$\langle\widehat{\theta}\rangle \in \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \frac{\ell\big(f(X_i), h(X_i;\theta)\big)}{q(X_i)} \tag{5.4}$$

or accept the additional approximation error incurred by not taking $q(X_i)$ into account.

5.2.2 Overfitting

"Overfitting" is a term that is frequently used in machine learning. It refers to the situation, where the training data set is so small (or, conversely, $h(x;\theta)$ is so expressive) that the optimal parameter choice for minimizing the Monte Carlo loss estimate $\langle\widehat{\theta}\rangle$ is much different from the parameter choice that minimizes the loss integral $\widehat{\theta}$. Detecting when overfitting occurs is relatively simple: it is common practice to hold back a subset of the available data as "validation data", which is not used to train the model. Overfitting is then revealed by checking whether the model generalizes to the validation data. There exist two main approaches to avoid overfitting.

Regularization. The first, "regularization", is to artificially limit the model capacity just enough so that overfitting no longer occurs. Typically, regularization is implemented by penalizing excessively large parameter values. For example, a so-called L^n regularizer produces a modified loss

$$\hat{\mathcal{L}}(f \,\|\, h;\theta) = \mathcal{L}(f \,\|\, h;\theta) + \lambda\|\theta\|_n\,, \tag{5.5}$$

that penalizes large-magnitude θ according to the L^n norm in addition to penalizing a bad fit, where the degree of regularization is controlled by a tunable "hyperparameter" λ that is not learned. There exist several other regularization techniques, many of which are tailored towards the specific model that is being trained (e.g. the use of prior distributions in bayesian inference, or "dropout" in neural networks) which are out of the scope of this dissertation.

More Training Data. The second approach to avoid overfitting—which may sound trivial—is to simply use a larger set of training data. In the limit, where the amount of training data approaches infinity, the optimal fit to the training data converges to the actual optimal fit, after all. In practice, though, it is often very difficult and/or expensive to obtain large-enough quantities of training data. Think of training a model to diagnose a specific type of brain cancer: there may simply not exist many people with that specific type of brain cancer to collect data from (and that is a good thing). For such cases, there exist techniques that *artificially* increase the amount of data, for example by "augmenting" the training data via

- exploiting symmetry equivalences (e.g. horizontally flipping images, where left and right should be interchangeable),
- adding random noise,

- using random crops (subsets of dimensions) of data points, and
- generating additional data using a computer simulation that was bootstrapped with the original data.

Furthermore, many optimization techniques assume independent data samples, but still work reasonably well when being fed the same data multiple times in a row in random order. One pass through the training data is referred to as an "epoch". In practice, when training data is sparse, it is not uncommon to optimize models with very large numbers of epochs.

Overfitting in this Dissertation. In this dissertation, we are in the fortunate position that our training data is relatively cheap to generate: it consists of Monte Carlo samples from (parts of) the path integral. This allows us to avoid overfitting without the need for any regularization, data augmentation, or using more than one epoch, simply by generating as much training data as is required on the fly. We will therefore disregard the pitfalls of overfitting in the remainder of this chapter on machine learning, and we will instead focus on models that are interesting to us.

5.2.3 Regression

Regression is the setting, where training data of the form of noisy input-output observations of $f(x)$ is available. Formally, a training data set with N data points is a set of tuples

$$\mathbb{D}_N = \Big\{ \big(x_1, \langle f(x_1)\rangle\big), \dots, \big(x_N, \langle f(x_N)\rangle\big) \Big\},\ x_i \in \mathcal{D}\,, \tag{5.6}$$

where $\langle f(x)\rangle$ is a noisy estimate of $f(x)$.

In regression, there are usually a number of model-dependent assumptions that make optimization feasible. A common set of assumptions would be that x_i are distributed uniformly and independently, and that the noise in the loss function induced by the noise of $f(x)$ has approximately zero-mean, i.e.

$$\begin{aligned}\mathbb{E}\Big[\ell\big(\langle f(x_i)\rangle, h(x_i;\theta)\big)\Big] &\approx \mathbb{E}\Big[\big\langle \ell\big(f(x_i), h(x_i;\theta)\big)\big\rangle\Big] \\ &= \ell\big(f(x_i), h(x_i;\theta)\big)\,.\end{aligned} \tag{5.7}$$

These assumptions make it possible to directly evaluate the previously introduced Monte Carlo estimate of the loss

$$\langle\widehat{\theta}\rangle \in \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \ell\big(\langle f(x_i)\rangle, h(x_i;\theta)\big)\,, \tag{5.8}$$

which in turn allows for a number of optimization techniques to be used. For example, if ℓ and h are differentiable w.r.t. θ, then setting the gradient of $\mathcal{L}$ to zero might be a viable option to find its minima.

5.2.4 Density Estimation

Density estimation addresses the same objective as regression, only with a different kind of training data. Instead of consisting of noisy input-output pairs, the training data consists only of the inputs x_i, but is distributed according to $p(x) = f(x)/F$:

$$\mathbb{D}_N = \left\{x_1, \ldots, x_N\right\},\ x_i \sim p\,. \tag{5.9}$$

From this setting, it is only possible to learn the probability density function (PDF) that is proportional to $f(x)$, leading to the name "density estimation". We refer to the learned density as $q(x;\theta) = h(x;\theta)/F$ and we express the loss in terms of PDFs for convenience

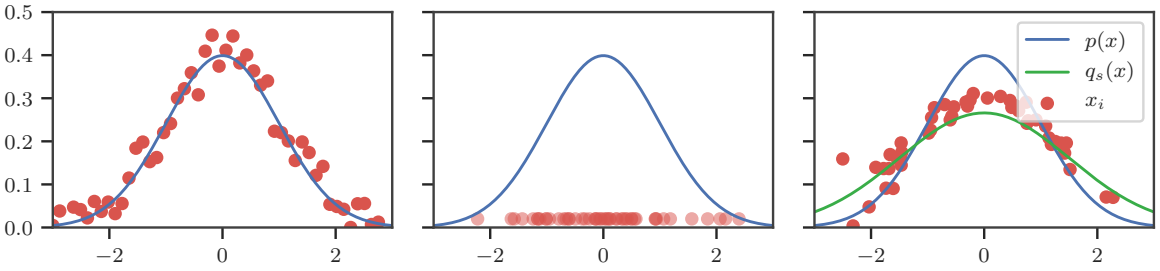
$$D(p \,\|\, q;\theta) = \mathcal{L}(F \cdot p \,\|\, F \cdot q;\theta) = \mathcal{L}(f \,\|\, h;\theta)\,. \tag{5.10}$$

Popular choices for $D(p \,\|\, q;\theta)$ include divergences such as the ones presented in Section 2.3; e.g. the KL-divergence D_{KL}.

Minimizing the loss in a density-estimation setting is often much more difficult than in the regression setting. After all, the loss function involves the ground-truth PDF p which is impossible to evaluate (otherwise, one could obtain regression training data by evaluating $p(x)$ for each data point). For certain loss functions, however, the fact that the training data is distributed according to p enables optimization. For example, the KL-divergence. Then

$$\begin{aligned}
\langle\widehat{\theta}\rangle &\in \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \frac{p(x_i)\log\frac{p(x_i)}{q(x_i;\theta)}}{p(x_i)} \\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \log\frac{p(x_i)}{q(x_i;\theta)} \\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \Big(\log p(x_i) - \log q(x_i;\theta)\Big) \\
&= \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log q(x_i;\theta)\,,
\end{aligned} \tag{5.11}$$

where the $\log p(x_i)$-term was dropped in the last step because it is independent of θ. This shows, that in order to minimize the KL-divergence in a density-estimation setting, one must maximize the average learned log-PDF of p-distributed training data.



(a) *Regression* ***(b)*** *Density estimation* ***(c)*** *w/ weights*

Figure 5.1: *Illustration of the training data available when performing regression* ***(a)****, density estimation* ***(b)****, and weighted density estimation / regression* ***(c)****.*

5.2.5 Weighted Density Estimation / Regression

As our goal in this dissertation is to learn an approximation and/or importance sampling of the path integral using Monte Carlo estimates of the same, the training data available to us has the form

$$\mathbb{D}_N = \left\{ \left(x_1, \langle f(x_1) \rangle\right), \ldots, \left(x_N, \langle f(x_N) \rangle\right) \right\}, \; x_i \sim q_\mathrm{s}\,, \tag{5.12}$$

where $q_\mathrm{s}(x)$ is known and may or may not be trained, and $\langle f(x) \rangle$ is an unbiased Monte Carlo estimate of $f(x)$. The resulting optimization objective

$$\langle \widehat{\theta} \rangle \in \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \frac{\left\langle \ell\big(f(X_i), h(X_i;\theta)\big) \right\rangle}{q_\mathrm{s}(X_i)} \tag{5.13}$$

lies somewhere between density estimation and regression: if $q_\mathrm{s}(x) = p(x)$, then the objective becomes that of density estimation, whereas if $q_\mathrm{s}(x) = 1/\mu(\mathcal{D})$, then the objective is that of regression with uniformly distributed data points. Figure 5.1 illustrates the different kinds of possible training data.

This continuum between density estimation and regression can either be classified as "weighted density estimation" or "weighted regression". Depending on the classification, different interpretations of the objective function are possible. In the density-estimation setting the weight is $w_\mathrm{d} = p(x)/q_\mathrm{s}(x)$, leading to the objective function

$$\langle \widehat{\theta} \rangle \in \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} w_{\mathrm{d},i} \frac{\left\langle d\big(p(X_i), q(X_i;\theta)\big) \right\rangle}{p(X_i)}\,, \tag{5.14}$$

whereas in the regression setting, the weight is $w_\mathrm{r} = 1/q_\mathrm{s}(x)$ with corresponding objective function

$$\langle \widehat{\theta} \rangle \in \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} w_{\mathrm{r},i} \left\langle \ell\big(f(X_i), h(X_i;\theta)\big) \right\rangle. \tag{5.15}$$

Regardless of how one prefers to write the objective, the optimal parameters $\langle\widehat{\theta}\rangle$ are the same. In the following, we describe suitable models for $h(x;\theta)$ and/or $q(x;\theta)$ and corresponding optimization techniques that we employ throughout this dissertation in the pursuit of learning the intricacies of light transport.

5.3 Piecewise-Constant Functions

When the to-be-learned function can be parameterized over a low-enough-dimensional space, e.g. $f(x) : \mathbb{R}^D \to \mathbb{R}$ with $D \leq 5$, then it is often feasible to approximate it at reasonable resolution using a piecewise-constant function. We take this approach in Chapters 6 and 7, where we learn piecewise-constant approximations of certain low-dimensional subspaces of the path integral that can later be reused to accelerate rendering.

5.3.1 Definition

A piecewise-constant function is constant within each element of a set of "bins" $B(\mathcal{D}) = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ that partition the function's domain

$$\mathcal{D} = \bigcup_{i=1}^{K} \mathcal{D}_i \,, \tag{5.16}$$

$$\forall i, j \in \{1, \ldots, K\} : i \neq j \implies \mathcal{D}_i \cap \mathcal{D}_j = \emptyset \,. \tag{5.17}$$

Let $\Theta(b) : B(\mathcal{D}) \to \mathbb{R}$ return the constant value of a piecewise-constant function within a given bin b, then the corresponding piecewise-constant function is defined as

$$h(x;\theta) = \sum_{j=1}^{K} \Theta(\mathcal{D}_i)\, \mathbb{1}_{\mathcal{D}_j}(x) \,, \tag{5.18}$$

where $\mathbb{1}_{\mathcal{D}}(x)$ is the indicator function that denotes set membership

$$\mathbb{1}_{\mathcal{D}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{D} \\ 0 & \text{otherwise.} \end{cases} \tag{5.19}$$

The piecewise-constant function $h(x;\theta)$ is therefore fully defined by the choice of bins $B(\mathcal{D})$ and the bin-to-value mapping $\Theta(b)$. Although any choice of $B(\mathcal{D})$ is theoretically allowed, in most practical scenarios where the domain spans the real hyperspace $\mathcal{D} = \mathbb{R}^D$, the bins $B(\mathbb{R}^D)$ are chosen to correspond to the subdivision of either a regular or an irregular grid (e.g. an axis-aligned tree data structure).

Tabulation. In the case where the bins are arranged in a regular grid, the process of approximating a function in a piecewise-constant manner is often referred to as "tabulation" without its close connection to machine learning being made clear. We argue, that viewing tabulation as a machine-learning model that requires sound optimization procedures helps avoid common subtle errors when training data is only available in the form of Monte Carlo estimates of the to-be-approximated function.

5.3.2 Optimization

We consider the scenario where the bins $B(\mathcal{D})$ are predetermined and our goal is to optimize the bin-to-value mapping $\Theta(b)$. First, we must realize that the bin-to-value mapping is nothing more than a list of values $\theta = (\theta_1, \ldots, \theta_K)$ with cardinality K being the number of bins and the j-th value being $\theta_j = \Theta(\mathcal{D}_j)$. The goal of the optimization is thus to find the bin values θ that minimize the loss integral. For piecewise-constant functions, the loss integral can be expressed as the sum of per-bin loss integrals

$$\mathcal{L}(f \parallel h; \theta) = \int_{\mathcal{D}} \ell\big(f(x), h(x;\theta)\big)\, \mathrm{d}\mu(x) \tag{5.20}$$

$$= \sum_{j=1}^{K} \int_{\mathcal{D}_j} \ell\big(f(x), \theta_j\big)\, \mathrm{d}\mu(x)\,, \tag{5.21}$$

which allows us to optimize $\theta_1, \ldots, \theta_K$ independently from each other using only the loss of those data points that lie within each respective bin

$$\langle\hat{\theta}_j\rangle \in \arg\min_{\theta_j} \frac{1}{N} \sum_{i=1}^{N} \frac{\big\langle \ell\big(f(x_i), \theta_j\big)\big\rangle}{q_\mathrm{s}(x_i)}\, \mathbb{1}_{\mathcal{D}_j}(x_i)\,. \tag{5.22}$$

We proceed to find the minimum of the loss by setting its gradient to zero. We use the square loss $\ell\big(f(x_i), \theta_j\big) = \big(f(x_i) - \theta_j\big)^2$, because it produces a simple closed form of the loss gradient

$$\nabla_{\theta_j} \frac{1}{N} \sum_{i=1}^{N} \frac{\Big\langle \big(f(x_i) - \theta_j\big)^2 \Big\rangle}{q_\mathrm{s}(x_i)}\, \mathbb{1}_{\mathcal{D}_j}(x_i) = -\frac{2}{N} \sum_{i=1}^{N} \frac{\big\langle f(x_i)\big\rangle - \theta_j}{q_\mathrm{s}(x_i)}\, \mathbb{1}_{\mathcal{D}_j}(x_i) = 0\,. \tag{5.23}$$

Solving for θ_j results in the sample mean

$$\langle\hat{\theta}_j\rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{\big\langle f(x_i)\big\rangle}{q_\mathrm{s}(x_i)}\, \mathbb{1}_{\mathcal{D}_j}(x_i)\,. \tag{5.24}$$

In conclusion, fitting a piecewise-constant function with respect to the squared loss requires simply assigning to each bin the mean value of all Monte Carlo estimates of the true function that fall within the bin.

Other Loss Functions. There exist many other loss functions, many of which perhaps being more reasonable to use than the square loss. Most of these are, however, generally more difficult to optimize for. The work of Pantaleoni and Heitz [2017] is a notable exception: they derive a closed-form optimization of piecewise-constant functions with respect to the χ^2-divergence, which is a very attractive quantity to optimize for if the fitted piecewise-constant function will be used to importance sample another Monte Carlo estimator, because the χ^2-divergence directly corresponds to the variance of this Monte Carlo estimator, as previously shown in Section 3.3.3. Unfortunately, in our experiments, their optimization was sometimes numerically unstable and prone to overfitting, even if a large amount of training data is available, which is why we ultimately use the square loss within this dissertation.

5.3.3 Usage

Fitting. Following the optimization procedure from the preceeding section, it is simple to fit a piecewise-constant function to a target function, even if only unnormalized stochastic estimates of the target function are available. One must obtain a reasonably large set of estimates of the target function within in each bin and then assign to each bin the mean of all estimates that fall within it. According to Section 3.2, the standard error of an estimated bin value $\langle\hat{\theta}_j\rangle$ compared to the optimally-fitted bin value $\hat{\theta}_j$ is $\mathcal{O}(1/\sqrt{N})$, where N is the number of samples that fall within bin j. This allows for a robust estimation of the required number of samples to obtain a fit within a given desired error tolerance.

Evaluation. The evaluation of a fitted piecewise-constant function $h(x;\theta)$ at a given input x is as simple as (i) determining the index j of the bin that contains x and (ii) looking up the corresponding fitted value θ_j. For example, when $x \in [0, d]$, K bins are arranged in a 1-dimensional regular grid, and θ is stored in an array, then the evaluation of $h(x;\theta)$ amounts to looking up the j-th value of the array, where

$$j = \left\lfloor \frac{x \cdot K}{d} \right\rfloor . \tag{5.25}$$

Sampling. It is also possible to draw samples according to $X \sim q(x;\theta)$, where $q(x;\theta) \propto h(x;\theta)$. To see how, it is important to realize that $q(x;\theta)$ is a mixture of uniform distributions with each uniform distribution having a

probability measure that is proportional to the value of its corresponding bin. Drawing a sample from $q(x;\theta)$ then amounts to (i) randomly selecting a bin with probability proportional to its value, and (ii) picking a uniformly random position within the chosen bin. The first step can be achieved regardless of the underlying domain $\mathcal{D}$ by subdividing the interval $[0,1]$ into K segments that have length proportional to $\Theta(\mathcal{D}_j)$, drawing an auxiliary random number $\xi \sim \mathcal{U}(0,1)$, and selecting the bin index j from the bin that corresponds to the segment that ξ lies within. The second step is difficult to implement for arbitrary bins $\mathcal{D}_j$, but is simple when the bins are arranged in a regular D-dimensional grid: in the latter case, one can simply draw a sample from the D-dimensional uniform distribution that covers the selected bin following Section 2.6.1.

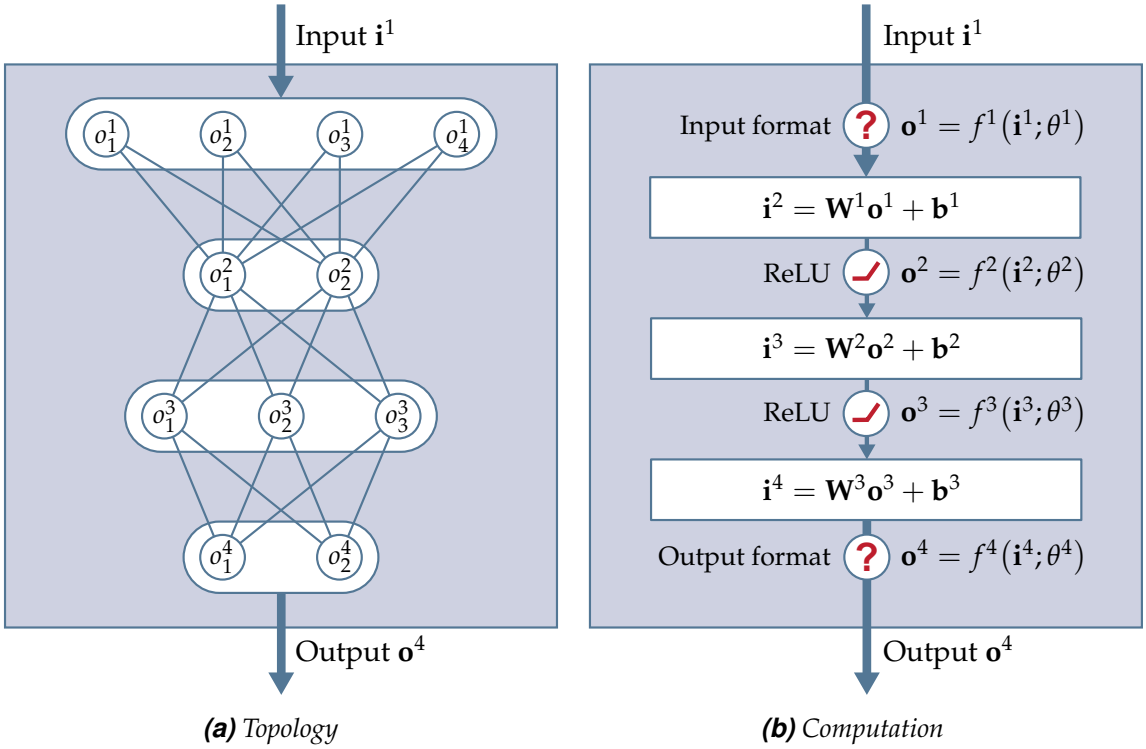
5.4 Neural Networks

Deep artificial neural networks (see Bengio et al. [2013]; LeCun et al. [2015] for a comprehensive review) are able to efficiently model complex relationships between input and output variables in a highly non-linear manner. They have emerged as the dominant model that achieves state-of-the-art results in a wide variety of challenging problems, e.g. image recognition [Simonyan and Zisserman 2014; He et al. 2016], machine translation [Wu et al. 2016], or generative modeling of raw audio and natural images [van den Oord et al. 2016a,b].

5.4.1 Definition

In principle, any system consisting of connected units ("neurons") where computation is performed along the connections is considered a "neural network". This includes not only "artificial" neural networks (e.g. as simulated on computers), but also biological neural networks as found in living organisms.

Artificial Feed-Forward Nets. In this dissertation, however, we are interested in only one particular kind of neural network: artificial feed-forward neural networks that are simulated on a computer. Feed-forward neural networks form no cycles, i.e. computation follows the edges of an acyclic directed graph, the nodes of which are the neurons. In the following, we will refer to artificial feed-forward neural networks simply as "neural networks". The computations within a neural network are divided into two categories:



(a) *Topology* ***(b)*** *Computation*

Figure 5.2: *Feed-forward neural-network topology* ***(a)*** *and corresponding computation graph* ***(b)*** *for an example network with 4 layers that have 4, 2, 3, and 2 neurons, respectively.*

inter-neuron computations along the connections between neurons and *intra*-neuron computations, also called neuron "activations", within each neuron. The inter-neuron computations are affine transformations, which, due to their simplicity, allow for a large number of connections to be computationally tractable. The neuron activations, on the other hand, are non-linear (otherwise, the repeated affine inter-neuron transformation would collapse to a single affine transformation) and typically performed on a per-neuron basis.

Multi-Layer Perceptrons. We will focus on those neural networks that are classified as "multi-layer perceptrons" (MLPs). The neurons of multi-layer perceptrons are arranged in $L \in \mathbb{N}; L \geq 2$ layers, where the inputs to the neurons of the l-th layer are represented by vector $\mathbf{i}^l$ and the outputs are another vector

$$\mathbf{o}^l = f^l\left(\mathbf{i}^l;\theta^l\right), \tag{5.26}$$

where $f^l\left(\mathbf{i}^l;\theta^l\right)$ is the layer's activation function parameterized by trainable parameters θ^l. The affine inter-layer computations are simple matrix multiplications with trainable real-valued "weight matrices" that capture individual

neuron connectivity, and trainable real-valued bias vectors

$$\mathbf{i}^{l+1} = a^l(\mathbf{o}^l) = \mathbf{W}^l\mathbf{o}^l + \mathbf{b}^l\,. \tag{5.27}$$

Figure 5.2 illustrates the topology and the computation graph of an example MLP with 4 layers that have 4, 2, 3, and 2 neurons, respectively.

The first layer $l = 1$ is the "input layer", the intermediate layers $l \in \{2, \ldots, L-1\}$ are "hidden layers" (because their values are intermediate results of the computation that are usually not observed externally), and the last layer $l = L$ is the "output layer". Therefore, the function learned by an MLP is

$$h(x;\theta) = f^1 \circ a^1 \circ \cdots \circ f^{L-1} \circ a^{L-1} \circ f^L\,, \tag{5.28}$$

where the set of trainable parameters is $\theta = (\theta^1, \mathbf{W}^1, \mathbf{b}^1, \ldots, \mathbf{W}^{L-1}, \mathbf{b}^{L-1}, \theta^L)$.

Activation Functions. We distinguish between the *input* activation function f^1, the *hidden* activation functions $f^2, \ldots, f^{L-1}$, and the *output* activation function f^L. In our work, the role of the input and output activation functions f^1 and f^L is to transform data to and from neural-network-suitable representations. For example, if the inputs span a large range of magnitudes, then normalizing them (e.g. such that they have zero-mean and unit variance) often drastically improves training performance. Input normalization is, in fact, a common strategy across a wide variety of machine-learning techniques. Some MLP architectures even normalize the distribution of intermediate results of hidden layers as part of the hidden activation functions, a process that is called batch normalization [Ioffe and Szegedy 2015]. We observed no benefit when using batch normalization in our experiments and we therefore only normalize inputs.

In similar spirit, when the desired output of a neural network should meet certain constraints, then it is worthwhile to enforce these constraints via a suitable output activation functions. For example, when the output should be a vector of probabilities that sum up to 1, then the "softmax" function $\Sigma(x)$ is a reasonable choice for transforming the arbitrary real-valued output of the last affine transformation to such a vector of probabilities

$$f^L(\mathbf{i}^L) = \Sigma(\mathbf{i}^L) = \frac{\exp(\mathbf{i}^L)}{\sum_{k=1}^{D} \exp(i_k^L)}\,. \tag{5.29}$$

In Chapter 8 we will present concrete problem-specific instances of input and output activation functions.

Lastly, due to a history of empirically and theoretically demonstrated robust performance, we choose to always use so-called "rectified linear units" (ReLUs) as hidden activation functions.

$$f_{\text{ReLU}}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.30}$$

In the following paragraph, we elaborate on the reasoning behind this choice.

Why ReLU-Based MLPs? Our choice of focusing on feed-forward MLPs with ReLU activations is motivated by various empirical and theoretical results. In terms of empiciral results, the widespread adoption of ReLUs in state-of-the-art neural-network architectures speaks for itself [Ramachandran et al. 2017]. Nevertheless, we experimented with alternative activation functions, including variations of the ReLU such as "parametric ReLUs" [He et al. 2015] and "concatenated ReLUs" [Shang et al. 2016], with and without looks-linear initialization as proposed by Balduzzi et al. [2017]. In all our experiments we were unable to outperform vanilla ReLUs, which is why they remained our preferred choice.

As for theoretical arguments in favor of MLPs *in general*, it has been shown that MLPs with a fixed number $L \geq 2$ of layers are universal function approximators under certain conditions on the activation function that the ReLU meets [Cybenko 1989; Hornik 1991]. Mathematically, what we mean by "universal function approximators" is that any bounded continuous function can be approximated within arbitrarily small error by some finitely wide, fixed-depth $L \geq 2$ layer MLP. Similar proofs are possible when, instead, the layer width is fixed but the number of layers may be arbitrarily big: Lu et al. [2017] showed that any Lebesgue-integrable function $f : \mathbb{R}^D \to \mathbb{R}$ (i.e. an even bigger class of function than the one considered above) can be approximated up to arbitrarily small error by a $(D + 4)$-wide, finitely deep MLP. Ongoing research is making progress towards compelling theoretical evidence for a greater approximation quality of deep ReLU networks rather than wide ones [Lu et al. 2017; Hanin 2017], which further motivates our choice of ReLU activations.

5.4.2 Optimization

It is generally impossible to minimize $\mathcal{L}(f \,\|\, h; \theta)$ in closed form when $h(x; \theta)$ is a neural network. Practitioners therefore resort to numerical optimization techniques: by far the most popular and successful approaches to optimizing deep neural networks are based on gradient descent.

(Stochastic) Gradient Descent. "Gradient descent" is an iterative function-minimization algorithm that utilizes the first-order derivative of the to-be-minimized function. It works as follows: given a function to minimize $\mathcal{L}(f \parallel h; \theta)$ and fixed parameters θ, a new set of parameters θ' is obtained by traversing the direction of steepest descent (i.e. the negative gradient of $\mathcal{L}(f \parallel h; \theta)$ at θ) for a fixed distance λ:

$$\theta' = \theta - \lambda \nabla_\theta \mathcal{L}(f \parallel h; \theta)\,. \tag{5.31}$$

If λ—usually called "learning rate"—is sufficiently small and $\mathcal{L}(f \parallel h; \theta)$ is locally Lipschitz w.r.t. θ, then $\mathcal{L}(f \parallel h; \theta')$ is smaller than $\mathcal{L}(f \parallel h; \theta)$. It can be proven that for $\lambda \to 0$, iterating the above update rule makes θ converge to a local optimum.

Since it is generally impossible or too expensive to compute the exact gradient $\nabla_\theta \mathcal{L}(f \parallel h; \theta)$, practitioners instead use unbiased estimates of it. This is called "stochastic gradient descent". Conveniently, even when only unbiased estimates of the gradient are used, the convergence of stochastic gradient descent to local minima can still be proven [Kiwiel 2001].

In practice, more advanced update rules than the one in Equation 5.31 are used. Popular methods make use of the first and second-order moments of unbiased loss gradient estimates in order to adaptively scale the learning rate λ. Exact descriptions of such advanced methods are out of the scope of this dissertation; in all our optimizations we use a method called "Adam" [Kingma and Ba 2014], which was—like stochastic gradient descent—proven to converge to local minima. Even though convergence is only guaranteed up to local minima, recent research indicates that the loss function of neural-network outputs may be well behaved enough such that local minima are a reasonable proxy of global minima [Li et al. 2017].

Obtaining Unbiased Loss Gradients. In order to use stochastic-gradient-descent-based optimization methods, one must be able to estimate $\nabla_\theta \mathcal{L}(f \parallel h; \theta)$. What makes this possible in the case of neural networks is the differentiability of $\ell\big(f(x), h(x;\theta)\big)$ via the chain rule[1]. For example, the gradient of the loss function w.r.t. the parameters of the l-th layer activation function θ^l is

$$\begin{aligned}\nabla_{\theta^l} \ell\big(f(x), h(x;\theta)\big) &= \frac{\partial \ell}{\partial \theta^l}\big(f(x), h(x;\theta)\big) = \frac{\partial \ell}{\partial h}\frac{\partial h}{\partial \theta^l}\big(f(x), h(x;\theta)\big) \\ &= \frac{\partial \ell}{\partial f^1}\frac{\partial f^1}{\partial a^1}\cdots\frac{\partial a^{l-1}}{\partial f^l}\frac{\partial f^l}{\partial \theta^l}\big(f(x), h(x;\theta)\big)\,.\end{aligned} \tag{5.32}$$

[1] The gradient computation hinges on the assumption that the loss function as well as the activation functions of the neural network are differentiable (almost) everywhere.

Due to the linearity of the gradient operator ∇_θ, one can express the gradient of the loss integral in terms of the gradient of the loss integrand[2]

$$\nabla_\theta \mathcal{L}(f \parallel h; \theta) = \int_{\mathcal{D}} \nabla_\theta \ell\big(f(x), h(x;\theta)\big) \, \mathrm{d}\mu(x) \,, \tag{5.33}$$

permitting the stochastic estimation of the gradient using a finitely sized set of data

$$\big\langle \nabla_\theta \mathcal{L}(f \parallel h; \theta) \big\rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{\big\langle \nabla_\theta \ell\big(f(x_i), h(x_i;\theta)\big) \big\rangle}{q(x_i;\theta)} \,. \tag{5.34}$$

As previously discussed, in the application to Monte Carlo light-transport simulation that we are interested in, we only have access to Monte Carlo *estimates* of $f(x)$, which means that we can not evaluate the above expression for arbitrary loss functions. In Chapter 8, we show that the above expression *can* be estimated without bias when optimizing the KL and χ^2-divergence loss functions, even when only $\langle f(x) \rangle$ is available. An alternative but more expensive approach is to estimate $f(x)$ with sufficiently many samples such that the remaining error can be neglected. We went this route in one of our co-authored publications that is not discussed in further detail within this dissertation [Kallweit et al. 2017].

5.4.3 Usage

Fitting. As previously discussed, we train a neural network to approximate a given target function using Adam optimization [Kingma and Ba 2014]. We compute the necessary stochastic gradient of the loss function w.r.t. the network's parameters using the automatic differentiation and backpropagation capabilities of the TensorFlow framework [Abadi et al. 2015].

Evaluation. Evaluating a trained neural networks for a given input x is done in closed form via Equation 5.28. Since the affine transforms and many common activation functions lend themselves to parallel execution via vector processor instructions, it is common to use optimized implementations that are available in existing frameworks. In this dissertation, we use TensorFlow [Abadi et al. 2015].

[2] This identity only holds under certain continuity constraints on ℓ. In Chapter 8, we actually run into a situation where the identity is invalid.

Sampling. Unfortunately, it is generally not possible to draw samples proportional to a function that was learned by a MLP. MLPs are therefore not directly applicable to importance sampling of Monte Carlo estimators. In Chapter 8, we show, however, how MLPs (and other gradient-descent-optimized function approximators) can be integrated into a computation graph that *does* permit learning of a PDF that can be sampled from and evaluated.

5.5 Prior Work using Machine Learning in Rendering

Even though sophisticated machine-learning techniques that are rooted in statistical modeling were only recently adopted in rendering literature [Vorba et al. 2014; Vévoda et al. 2018], simpler techniques such as tabulation have seen usage for multiple decades. In this section, we provide a brief sub-sampling (a complete list would be too expansive for the scope of this dissertation) of various usages of machine learning that exist in the field of photorealistic rendering.

5.5.1 Tabulation and Piecewise-Polynomial Fits

Tabulation has a long history of successful usage in rendering and the field of Computer Graphics in general. Perhaps the most obvious application of tabulation is the standard representation of images: as a regular lattice of pixels. Although pixels are typically defined as point samples of the incident radiance field on the image plane, these point samples are often approximated via a filtering kernel to combat aliasing. Using a box-filter (also known as "nearest-neighbor" filter) amounts precisely to approximating the image plane as a piecewise-constant function according to the definition and optimization procedure we gave in Section 5.3. In practice, it is common to use smoother filter kernels than a box (e.g. tent or Gaussian shaped), which we will also consider as "tabulation" in the remainder of this dissertation. Other similarly trivial examples of the usage of tabulation to approximate 2-dimensional functions are textures (including their corresponding mipmaps) and environment maps.

Tabulation is also frequently used to represent reflectance functions, such as the data-driven BSDFs we described in Section 4.6.9 and data-driven phase functions (e.g. the Lorenz-Mie phase function from Section 4.7.4). Researchers developed specialized reflectance functions for other kinds of scatterers than differential surface patches and differential volume regions, for instance for hair segments [Marschner et al. 2003], water droplets [Sadeghi et al.

2012], grains [Müller 2016], and large volume regions [Kajiya and Kay 1989; Jensen et al. 2001; Moon et al. 2007; Lee and O'Sullivan 2007]. Many of these reflectance functions are difficult or impossible to accurately model in closed form; instead, their advantage lies in parameterizing reflectance using a small number of dimensions that are feasible to tabulate within the practical memory and computational limits of today's hardware.

We take a similar approach in Chapter 6, where we present a novel parameterization of large-scale volumetric light transport that allows us, after an initial tabulation step, to drastically accelerate the computation of volumetric light transport in continuous and granular participating media with little loss of accuracy. In Chapter 7 we tabulate the 5-dimensional incident radiance field at a signal-adaptive resolution such that it can be used to importance sample a path tracer much better than many traditional heuristics can.

5.5.2 Parametrix Mixture Models

It is often convenient to approximate a function using a (weighted) sum of parametric basis functions. Such a mixture of parametric basis functions is generally referred to as a "parametric mixture model". The basis functions can take any form; prior work on reflectance functions and light-field representation used, for example, spherical harmonics [Cabral et al. 1987; Westin et al. 1992], spherical wavelets [Schröder and Sweldens 1995], Zernike polynomials [Koenderink et al. 1996], cosine lobes [Heitz et al. 2016a,b], and Gaussians [Wang et al. 2009; Xu et al. 2013; Vorba et al. 2014; Herholz et al. 2016]. Some works even generalize beyond the exact choice of underlying basis function [Herholz et al. 2018].

Notice, that the piecewise-constant approximation as defined in Section 5.3 also falls under the umbrella of parametric mixtures: the additive component of each bin (Equation 5.18) is a parametric (its constant value being its single parameter) basis function. The piecewise-constant approximation corresponds to a special case of parametric mixtures, though, because its basis functions are non-overlapping, i.e. at any given point in the function's domain, only at most a single basis function has a non-zero value. This makes many mathematical derivations (including the optimization that we derived) much simpler than in the general case where basis functions may overlap. In this dissertation, we do not make any use of parametric mixture models beyond the piecewise-constant approximation, which is why we will not describe their applications and optimization procedures in further detail.

5.5.3 Probabilistic Models

In this chapter, we introduced machine learning through the point of view of general function approximation. When the to-be-learned function is a probability distribution, then it is often possible to craft more effective optimization procedures than would be possible in the general case. Probabilistic modeling has seen little usage in rendering literature for a long time, but was recently picked up in the form of Bayesian learning of posterior distributions [Vorba et al. 2014; Vévoda et al. 2018]. Since we only take the probabilistic view in a single one of our chapters (Chapter 8) we defer a more detailed overview over probabilistic modeling and a corresponding survey of previous work to that chapter.

5.5.4 Neural Networks and Deep Learning

Although deep learning (and neural networks in general) have been successfully applied to problems in computer graphics, such as denoising [Bako et al. 2017; Chaitanya et al. 2017; Vogels et al. 2018] and fluid simulation [Chu and Thuerey 2017], they have seen relatively little usage in photorealistic rendering. Three notable exceptions are the work of Nalbach et al. [2017] who use convolutional neural networks (CNNs) to synthesize ambient occlusion, illumination, and other effects in screen space. Their approach operates in 2D, using color and feature images as inputs to the network. More closely related to our work in the context of path tracing is that of Ren et al. [2013], who train multiple spatially distributed 2-hidden-layer neural networks to predict radiance in a specific region of a scene.

More recently, in one of our co-authored works [Kallweit et al. 2017] we also predict radiance, but operate globally rather than locally, using a much deeper network architecture, and utilizing several recent advances in the field of deep neural networks such as residual connections [He et al. 2016] and Adam optimization [Kingma and Ba 2014].

In contrast, our method in Chapter 8 not only learns a predictive approximation of the radiance field, but can also draw samples from it, which is enabled by recent advances in the field of normalizing flows [Dinh et al. 2014, 2016].

Part II

Technical Contributions

CHAPTER 6

Multiple Scattering in Translucent Materials

This chapter is based on the following publication by Müller et al. [2016]:

Efficient Rendering of Heterogeneous Polydisperse Granular Media
Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, Jan Novák
ACM Trans. Graph. (Proc. SIGGRAPH Asia), vol. 35, no. 6, pp. 168:1–168:14, Nov 2016

6.1 Overview and Prior Work

In this chapter we aim at accelerating the rendering of bright translucent materials, the appearance of which is dominated by light paths that scatter hundreds to thousands of times beneath the surface. We are particularly interested in a method that can handle *dynamic, heterogeneous* translucent materials (i.e. materials with spatially and temporally varying appearance parameters), as those are most difficult to render with traditional techniques.

Our effort is focused on idealized continuous participating media that obey the radiative transfer equation (RTE). We made this choice not only because such media are a frequently used approximation in movie production, but also because their accelerated rendering constitutes the missing piece that is required to handle dynamic, heterogeneous assemblies of grains in existing techniques that approximate part of the appearance of *granular* media [Meng et al. 2015; Müller 2016] using the RTE. Granular assemblies are desirable to

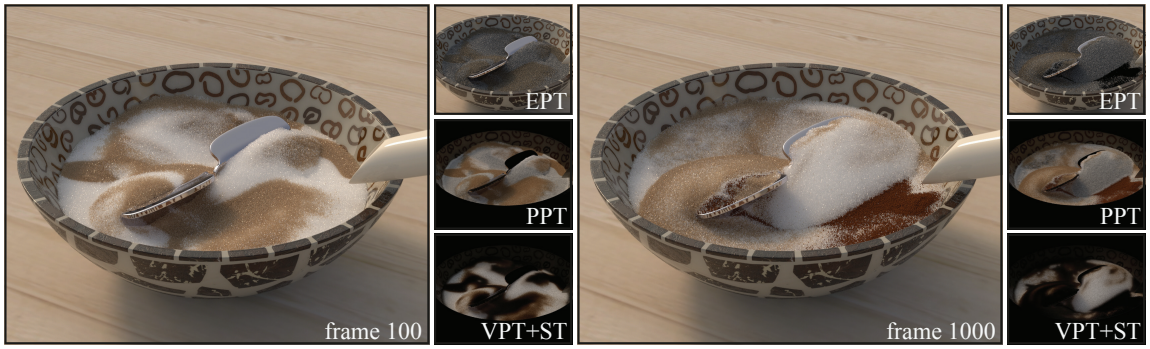


Figure 6.1: *Our method allows efficient rendering of dynamic, heterogeneous mixtures of grains, as demonstrated in two frames of the* BOWL *animation. The small images on the right of each frame show the individual components of our method, which add up to the full light transport. Explicit Path Tracing (EPT) simulates most of the high-frequency detail, such as glints. Proxy Path Tracing (PPT) accelerates the simulation of light scattering within each individual grain. Shell Tracing (ST) advances light through the medium in large steps. Across the entire simulation, we achieve a* 2.5–5× *speed improvement over the explicitly path traced reference.*

render, because they are ubiquitous in nature; examples of their occurrence include snow, sand, spices, and bubbles.

6.1.1 Multiple Scattering in Continuous Volumes

Although there already exist several approaches for accelerating the simulation of long "high-order" paths in RTE-based continuous volumes, they all either suffer from visible approximation error [Stam 1995; Jensen et al. 2001; Li et al. 2005] or require scene-dependent precomputation [Lee and O'Sullivan 2007; Moon et al. 2007; Ramamoorthi 2009; Zhao et al. 2013]. Our goal is to find a technique that *at most* needs a single scene-*independent* precomputation while still being more accurate than existing diffusion-based approaches (Figure 6.2(e)).

The work of Lee and O'Sullivan [2007] and Moon et al. [2007] is of particular interest to us in this regard. They propose to side-step the expensive simulation of long light paths by learning the propagation of radiance through large spherical shells filled with the to-be-rendered medium. The authors then invoke their learned "shell transport functions" (STFs) to traverse the granular medium in large steps (the radius of their shells) during rendering, referred to as "shell tracing" (Figure 6.2(d)). Although the STFs require a per-medium precomputation, shell tracing is otherwise independent of the scene geometry and introduces no bias other than that caused by an imperfect fit. This is in stark contrast with the popular diffusion approximation that

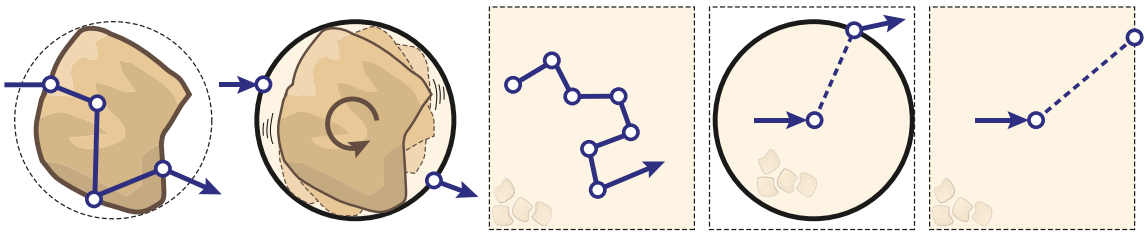
inherently makes simplifying assumptions about the scene geometry and the anisotropy of the underlying medium.

These desirable aspects of shell tracing and its high performance make it attractive to build on. We leverage an isotropy assumption found by Müller [2016] to reduce the dimensionality of STFs from 4-D to 2-D, allowing us to densely fit an STF database over the *full* RTE appearance space (α, g), where g parameterizes the Henyey-Greenstein phase function. To this end, we import a technique called "white Monte Carlo" [Alerstam et al. 2008] from biomedical optics that enables the computation of STFs for all possible values of α *at once*. The resulting precomputation time is just a few hours, running on a single workstation. Our STF database permits the efficient usage of the correct STF in arbitrary continuous media on the fly during rendering.

6.1.2 Granular Materials

The appearance of granular materials can be classified into two components. First, there is high-frequency detail that is governed by the fine-scale structure and arrangement of individual grains. This detail is caused by short "low-order" light paths that either directly reflect incoming light towards the camera, producing glints and the granular texture, or few-bounce indirect reflections that exhibit perceivable glimmer. Secondly, in addition to the high-frequency detail, there exists a smooth large-scale appearance that is shared with other translucent materials and which leads to the characteristic saturated and soft color that most granular materials have. This part of the appearance is caused by similarly long high-order light paths as those found in continuous participating media.

Although both the low-order and the high-order components are a result of the same underlying light transport that is governed by the full granular detail and therefore can be obtained by explicit path tracing (Figure 6.2(a)), each component exhibits fundamentally different characteristics that warrant a specifically tailored acceleration technique. Low-order paths are usually cheap to simulate because they are short, but they may exhibit large variance due to their radiance profile being sharp and high-frequency. This is especially noticeable when the grains are shiny and reflective, producing direct reflections of light, such as glints, that are difficult to importance sample. High-order paths, on the other hand, by consisting of large numbers of vertices, are exceedingly expensive to simulate. Their variance is typically lower than that of low-order paths, but not by enough to counteract their greater expense.



(a) *Explicit PT* ***(b)*** *Proxy PT* ***(c)*** *Volum. PT* ***(d)*** *Shell tracing* ***(e)*** *Diffusion*

Figure 6.2: *Overview of different techniques that range from accurate and slow (left) to approximate and fast (right). Meng et al. [Meng et al. 2015] combine* **(a)**, **(c)**, *and* **(e)**, *Moon et al. [2007] combine* **(a)** *and* **(d)**, *and Müller [2016] combines* **(a, b, c, and d)**. *We extend Müller's technique by extending* ***(d)*** *and automatically selecting between* ***(a)*** *and* ***(b)***.

Due to the inherent difficulty in efficiently simulating either of the two components, there exists previous work tailored to high-order scattering [Moon et al. 2007; Meng et al. 2015; Müller 2016] and to low-order scattering [Müller 2016].

Meng et al. [2015] propose to approximate the granular medium by a corresponding continuous participating medium, the RTE parameters of which are obtained in a learning step prior to rendering from the individual grains and their arrangement. Meng et al. are then able to leverage volumetric path tracing [Lafortune and Willems 1996] (Figure 6.2(c)) in conjunction with existing high-order acceleration techniques for the RTE; they use a variant of the diffusion approximation [Stam 1995; Jensen et al. 2001; Li et al. 2005] (Figure 6.2(e)). Müller [2016] extends Meng et al.'s method to be able to extract RTE parameters from heterogeneous, polydisperse arrangements of grains and proposes to leverage a simplified form of STFs instead of the diffusion approximation to reduce error. They are, however, unable to leverage STFs for accelerating the rendering of *general* heterogeneous continuous media—this is where our contributions fit in.

Unfortunately, these high-order solutions are quite approximate: not only do granular media not follow the underlying assumption of the RTE (i.e. infinitesimal, independently distributed scatterers), but many existing techniques for efficiently solving the RTE introduce additional approximation error. Therefore, the high-order solutions *must* be prefixed by explicit path tracing through several grains in order to retain visual accuracy. The bottleneck of the simulation is thereby shifted towards low-order scattering. Müller [2016] addresses part of this bottleneck by introducing a complementary low-order acceleration technique, named "proxy path tracing" (Figure 6.2(b)), which replaces the detailed geometry of each grain with a statistical aggregate, referred to as "grain scattering distribution function" (GSDF). Proxy

path tracing is accurate enough to be used immediately on the first interaction from the camera for distant views, and typically requires handling only the very first grain using explicit path tracing for close-up views—significantly reducing the current path tracing bottleneck in low-order scattering. We embrace proxy path tracing as a component of rendering granular materials and we propose a novel heuristic that automatically determines which light paths require explicit path tracing of the first grain interaction based on an error analysis of the GSDF.

6.1.3 Contribution Summary

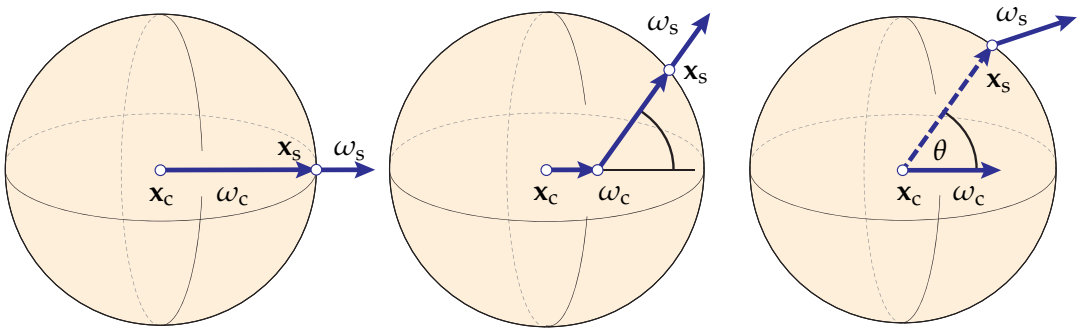
Our contributions in this chapter are two-fold: (i) we extend shell tracing to handle dynamic, heterogeneous continuous participating media using only a single scene-independent precomputation, and (ii) we propose an automated heuristic that selects between Müller's proxy path tracing and explicit path tracing at the first grain interaction along each simulated light path. The end result is the first accelerated method that can handle heterogeneous, dynamically changing mixtures of grains like that shown in Figure 6.1 without requiring manual tuning or expensive precomputation.

6.2 Multiple Scattering in Continuous Volumes

Our goal in this section is to aggregate many light-particle interactions into one step to accelerate the evaluation of high-order transport in high-albedo RTE-based continuous volumes. We first briefly review the 4-D STFs by Lee and O'Sullivan [2007] and then explain the modified 2-D STFs of Müller [2016]. Lastly, we describe our novel approach to computing Müller's STFs densely over the RTE parameter space (σ_t, α, f_p), allowing their use in any scene without dedicated precomputation.

6.2.1 Definition of Shell Transport Functions

A shell transport function [Lee and O'Sullivan 2007; Moon et al. 2007] aggregates volumetric light transport that exits the surface of a large spherical region. More precisely, the STF captures the outgoing light field on the shell surface due to an infinitesimal pencil beam emitter at the shell center. The STF is ten-dimensional, being parameterized by the location $\mathbf{x}_c$ and direction ω_c of the emitter, and by the position $\mathbf{x}_s$ and direction ω_s on the surface of the shell (see Figure 6.3 for an illustration). Given a photon originating from this



(a) *Uncollided radiance* **(b)** *Single scattering* **(c)** *Multiple scattering*

Figure 6.3: *The parameterization of the shell transport function of Müller [2016]. The function is split into non-scattering paths* **(a)***, single scattering paths* **(b)** *and multiple scattering paths* **(c)***. The dashed arrow from* $\mathbf{x}_c$ *to* $\mathbf{x}_s$ *summarizes all scattering happening within the shell.*

pencil beam emitter, the STF quantifies the PDF of the photon first exiting the shell at $\mathbf{x}_s$ in direction ω_s:

$$f_{stf}(\mathbf{x}_c, \omega_c \rightarrow \mathbf{x}_s, \omega_s) = \alpha_s p_s(\mathbf{x}_s, \omega_s | \mathbf{x}_c, \omega_c)\,, \tag{6.1}$$

where α_s and $p_s(\mathbf{x}_s, \omega_s | \mathbf{x}_c, \omega_c)$ are the fraction and spatio-angular distribution of light that reached the surface of the shell, respectively. We will refer to α_s as the shell albedo.

By assuming a homogeneous participating medium (i.e. optical properties invariant with respect to position or direction), Lee and O'Sullivan [2007] make the STF agnostic to the position and orientation of the emitter, and rotationally symmetric about ω_c. The function thus reduces to four dimensions and can be conveniently parameterized by the radius $r = \|\mathbf{x}_s - \mathbf{x}_c\|$, elevation angle of the surface point $\theta = \cos^{-1}\left(\omega_c \cdot N(\mathbf{x}_s)\right)$, and outgoing direction ω_s; see Figure 6.3 for an illustration. To precompute the STF, Lee and O'Sullivan [2007] trace a large number of randomly seeded photons through an infinite homogeneous participating medium and fit a piecewise-constant function to the directional radiance profile of photons that reach a distance r from the origin.

6.2.2 Appearance Parameterization

In order to efficiently support spatially varying appearance, we strive for precomputing a *general* function, fitted only *once*, that can be used for arbitrary RTE-based participating media parameterized by (σ_t, α, f_p). We model this function as piecewise constant and follow the fitting procedure described in Section 5.3.

Since the STF, as proposed by Lee and O'Sullivan [2007], is already 4-D, adding the RTE parameters naively as extra dimensions would make precomputation and storage intractable. Fortunately, the extinction coefficient of the medium can be implicitly accounted for by expressing the shell's radius r in units of mean free paths $1/\sigma_\mathrm{t}$. Furthermore, we can reduce the phase function f_p to a single parameter, the mean cosine g of scattered light in the Henyey-Greenstein phase function [Henyey and Greenstein 1941]. The appearance of any homogeneous participating medium can thus be classified using (α, g). Adding these to the STF results in a total of only six dimensions, which, while still being intractable to store densely, are close to a feasible number of dimensions. Furtunately, Müller [2016] tackled this remaining issue by studying the directional distribution of radiance exiting the shell and proposing a decomposition that allows reducing the STF to only four remaining dimensions. Under normal circumstances, fitting such a 4-D piecewise-constant function at high resolution would still be expensive, but in this particular situation we are able to leverage white Monte Carlo [Alerstam et al. 2008] to fit the function for all possible values of α *at once*, effectively reducing the number of to-be-fitted dimensions to only three.

6.2.3 Separation of Uncollided and Once-Scattered Radiance

Incorporating all modification described above, we define our STF as a sum of uncollided, single-scattered, and multi-scattered components:

$$\begin{aligned} f_\mathrm{stf}(\alpha, g, \mathbf{x}_\mathrm{c}, \omega_\mathrm{c} \to \mathbf{x}_\mathrm{s}, \omega_\mathrm{s}) = {} & \alpha_\mathrm{s}^0(r)\, \delta(\omega_\mathrm{c} - \omega_\mathrm{s}) \\ & + \alpha_\mathrm{s}^1(\alpha, g, r)\, p_s^1(\mathbf{x}_\mathrm{s}, \omega_\mathrm{s} | g, r) \\ & + \alpha_\mathrm{s}^\mathrm{m}(\alpha, g, r)\, p_s^\mathrm{m}(\mathbf{x}_\mathrm{s}, \omega_\mathrm{s} | \alpha, g, r)\,, \end{aligned} \tag{6.2}$$

where the superscript "0, 1, m" mark quantities of the respective components, $p_s^1(\mathbf{x}_\mathrm{s}, \omega_\mathrm{s}, r)$ is the distribution of single-scattered photons. The joint PDF $p_s^\mathrm{m}(\mathbf{x}_\mathrm{s}, \omega_\mathrm{s} | \alpha, g, r)$ is defined as the product of positional and directional components:

$$p_s^\mathrm{m}(\mathbf{x}_\mathrm{s}, \omega_\mathrm{s} | \alpha, g, r) = p_s^{\mathrm{m},\mathbf{x}}(\theta | \alpha, g, r)\; p_s^{\mathrm{m},\omega}(\omega_\mathrm{s} | \alpha, g, r, \theta)\,. \tag{6.3}$$

Separating out the uncollided and single-scattered radiance, and integrating them on the fly using Monte Carlo estimators, results in a significantly smoother function to be fitted. We also make use of the observation that the multi-scattering lobe $p_s^{\mathrm{m},\omega}$ can be approximated well by a cosine-weighted hemispherical distribution oriented about normal $N(\mathbf{x}_\mathrm{s})$ [Müller 2016]; Figure 6.5 demonstrates the approximation error. This reduces the p_s^m to four dimensions (α, g, r, θ).

Table 6.1: *STF precomputation discretization spacing, range, and resolution of the four tabulated dimensions. We only precompute a given radius r if the corresponding shell albedo (α_s) is larger than 0.5. Furthermore, we add an additional bin for $\alpha = 1$.*

Dim.	Domain	Spacing	Resolution
g	$\in [-1, 1]$	linear in g	200 bins
α	$\in [0, 0.999]$	linear in $\log \frac{1}{1-\alpha}$	1000 bins
r	$\in [1, 256]$ $(\alpha_s > 0.5)$	linear in $\log r$	≤ 9 bins
θ	$\in [0, \pi]$	linear in $\cos\theta$	45 bins

6.2.4 Precomputation of the STF Database

To construct the STF database p_s^m, we discretize its individual components according to Table 6.1 and populate it via white Monte Carlo [Alerstam et al. 2008] in the following way. For each value of the mean cosine g_j, we create an infinite homogeneous volume with the corresponding HG phase function and $\alpha = 1$ (hence the "white" in "white Monte Carlo"). We trace N photons and first fit the shell with the smallest radius. The radius for the k-th shell is computed as $r_k = 2^{k-1}$ mean free paths; we start with $k = 1$. For each spatial bin θ_l, we construct a histogram H_l, where the b-th bin *counts* photons exiting the shell after exactly b bounces; see Figure 6.4. This allows computing the multi-scattered radiance for *any* value of the discretized albedo α_i as

$$p_s^{m,\mathbf{x}}(\theta_l|\alpha_i, g_j, r_k) = \frac{1}{N}\sum_{b=2}^{|H_l|} \alpha_i^b H_l[b]\,. \qquad (6.4)$$

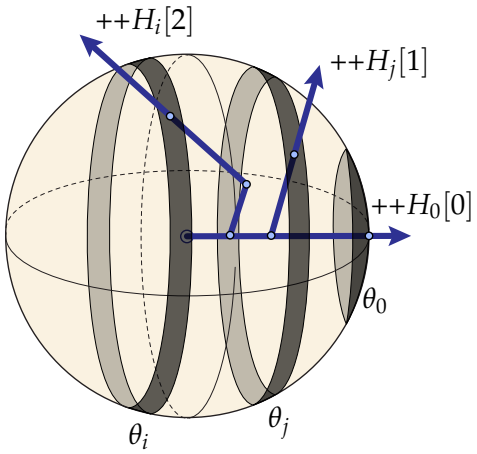


Figure 6.4: *Shell histograms.*

We also use the histogram to compute the albedos of the individual components as

$$\alpha_s^0(r_k) = \exp(-r_k)\,, \qquad (6.5)$$

$$\alpha_s^1(\alpha_i, g_j, r_k) = \frac{1}{N}\sum_l \alpha_i H_l[1]\,, \qquad (6.6)$$

$$\alpha_s^m(\alpha_i, g_j, r_k) = \frac{1}{N}\sum_l \sum_{b=2}^{|H_l|} \alpha_i^b H_l[b]\,. \qquad (6.7)$$

Equations 6.4–6.7 express all terms for a shell of radius r_k mean free paths. We use the smaller shells to accelerate tracing of paths for larger shells; this

significantly reduces the number of traced path segments, cutting down the precomputation of the entire STF database to around 10 hours on our workstation running two Intel Xeon E5-2680v3 CPUs (24 cores in total). With our discretization defined in Table 6.1 stored as an array of 4 byte floating-point numbers, the entire STF database occupies roughly 400 MB of memory in uncompressed form. We did not attempt to reduce the database's memory footprint, and we chose its resolution conservatively.

6.2.5 Shell Tracing with the STF Database

Homogeneous Media. Suppose a light path within a RTE-based participating medium is to be continued from vertex $\mathbf{x}_c$ in direction ω_c. We first look up the STF with the best-matching g and α parameters from our database. Then, we use the largest radial component of this STF that fits inside the mesh to advance the light path. Next, we randomly pick one of the components proportional to their respective α_s. If the uncollided component is chosen, we simply continue the path forward in r mean free path units, i.e. $\mathbf{x}_s = \mathbf{x}_c + r\omega_c$, $\omega_s = \omega_c$. For single scattering, we first sample the free-flight distance to the *one* scattering location, $\mathbf{x}$, then generate direction ω_s by phase function importance sampling, and set $\mathbf{x}_s$ to the shell surface point seen from $\mathbf{x}$ in ω_s. To sample multiple scattering, we sample $\mathbf{x}_s$ on the surface of the shell according to the fitted piecewise-constant distribution $p_s^{m,\mathbf{x}}$ and choose ω_s by cosine-weighted sampling of the hemisphere at $\mathbf{x}_s$.

Figure 6.5 shows the Lucy statue filled with a forward-scattering high-albedo continuous volume. The renderings use various instances of the STFs to compute high-order transport and demonstrate the error of the multiple approximations we use. In order to evaluate our method on varying medium densities, we show the statue at four different sizes: 10 cm, 1 m, 10 m, and 100 m large. We compare the VPT ground truth to:[1]

- STFs with a cosine hemisphere as their directional component,
- STFs with our additional assumption that the volume has a HG phase function,
- STFs additionally interpolated logarithmically in size such that they tightly fit into the volume boundary,
- the diffusion approximation used by Meng et al. [2015].

[1] The full 4-D STFs as described by Lee and O'Sullivan [2007] are an exact solution up to a quantization error, and are thus not listed in Figure 6.5.

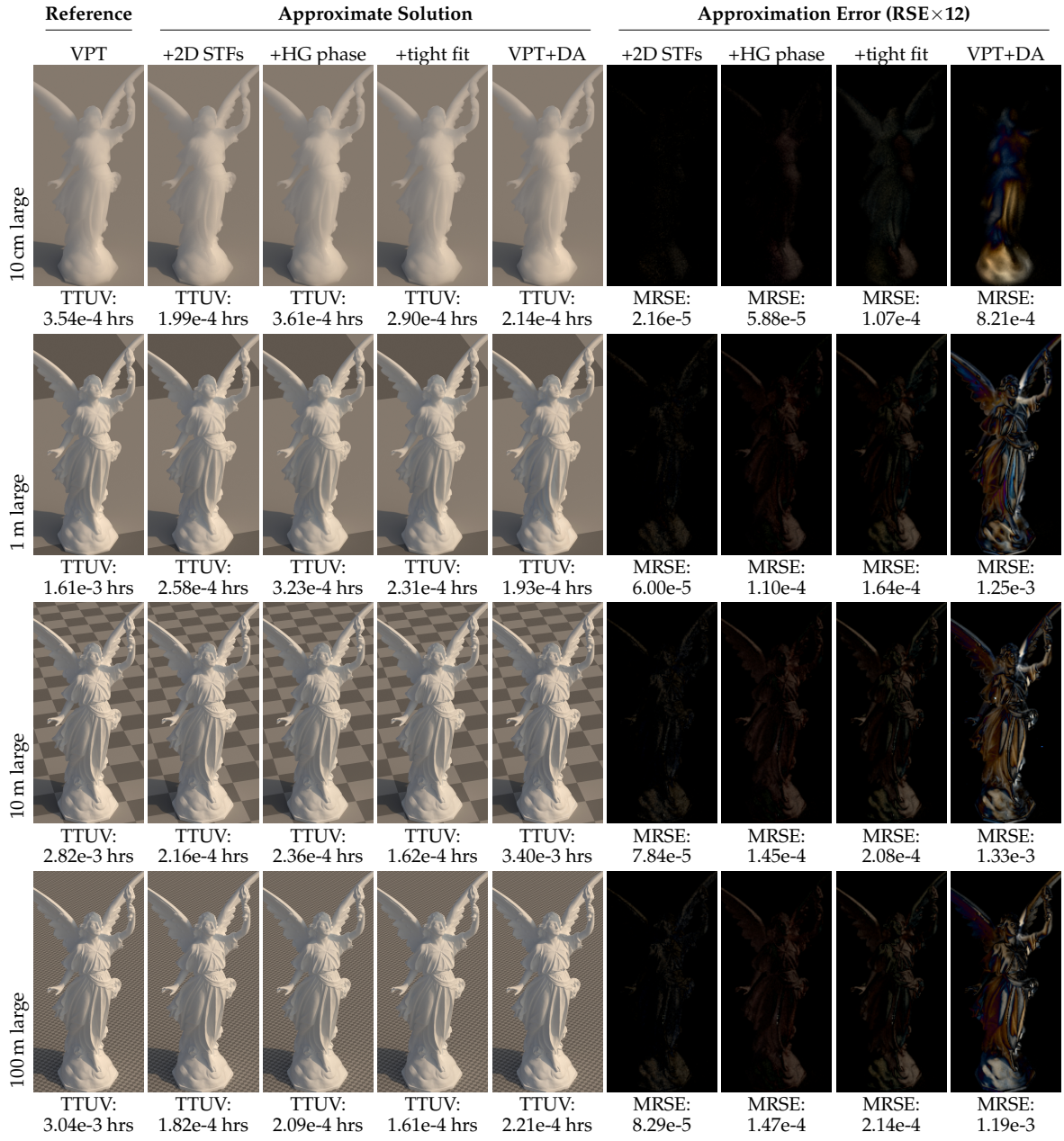


Figure 6.5: *Error introduced by the STF approximations. The* LUCY *statue is filled with a forward-scattering homogeneous continuous medium. The first column shows volumetrically path traced (VPT) reference images, which can be matched perfectly by 4-D STF. The following three columns demonstrate the visual impact of each approximation: 2-D STFs approximate $p_s^{\mathrm{m},\omega}$ as a cosine-weighted hemispherical distribution [Müller 2016], using the HG phase function reduces the dimensionality of continuous-volume-appearance space, and interpolation between tabulated STFs allows fitting STFs tightly into the mesh. The last column shows the diffusion approximation (DA) employed by Meng et al. [2015] to illustrate the increased accuracy of our proposed solution.*

The results suggest that the cosine hemisphere approximation introduces minimum error compared to diffusion while maintaining a similar rendering speed. The next step—introducing the HG phase assumption—again adds little error, but permits the precomputation of STFs over a dense appearance space, which can be re-used for all possible sets of RTE parameters. However, the lookups of the correct STF from this database during path tracing end up increasing render times slightly. With our last approximation—interpolation of STFs in order to achieve tight fits in the bounding mesh—we introduce further error, but gain speed again.

Heterogeneous Media. In heterogeneous media, we choose an STF the same way as in homogeneous media, except that we use the RTE parameters at the current path vertex. The difference is in choosing the size (radius) of the STF. We tested two techniques that are inspired by Müller [2016]: *greedy shell selection*, which ignores the heterogeneity and selects the shell respecting only the boundary of the assembly (like in homogeneous media), and *conservative shell selection*, limiting the shell size to a locally homogeneous region. Müller's conservative shell selection is only designed for binary heterogeneous media where a clear boundary between piecewise-homogeneous regions exists. In the following, we present a more general approach that is able to identify locally homogeneous regions in arbitrary heterogeneous media.

Conservative STF Selection. When conservatively selecting STF sizes, we limit the amount of variation of σ_t and α within the volume of the selected STF. More specifically, we choose an upper bound on the STF radius r by enforcing the following constraints on all locations $\mathbf{x}$ inside the STF:

$$\left|\log_2\left(\frac{\sigma_\mathrm{t}(\mathbf{x})}{\sigma_\mathrm{t}(\mathbf{x}_\mathrm{c})}\right)\right| < 0.5\,, \tag{6.8}$$

$$\left|\log_2\left(\frac{1-\alpha(\mathbf{x})+\epsilon}{1-\alpha(\mathbf{x}_\mathrm{c})+\epsilon}\right)\right| < 0.5\,. \tag{6.9}$$

Constraint (6.8) is motivated by the exponential nature of transmittance $\tau(\sigma_\mathrm{t}, t) = \sigma_\mathrm{t} e^{-\sigma_\mathrm{t} t}$. Constraint (6.9) is based on the observation that the diffuse reflectance of a continuous volume is roughly proportional to $\log(\frac{1}{1-\alpha})$ up to values very close to 1, at which point the relation becomes sublinear. We set $\epsilon = 0.001$ in order to avoid the singularity at $\alpha = 1$, and to capture the sublinear aspect.

Figure 6.6 shows a comparison of the greedy and conservative shell selection on a heterogeneous wedge with a variety of optical configurations. The

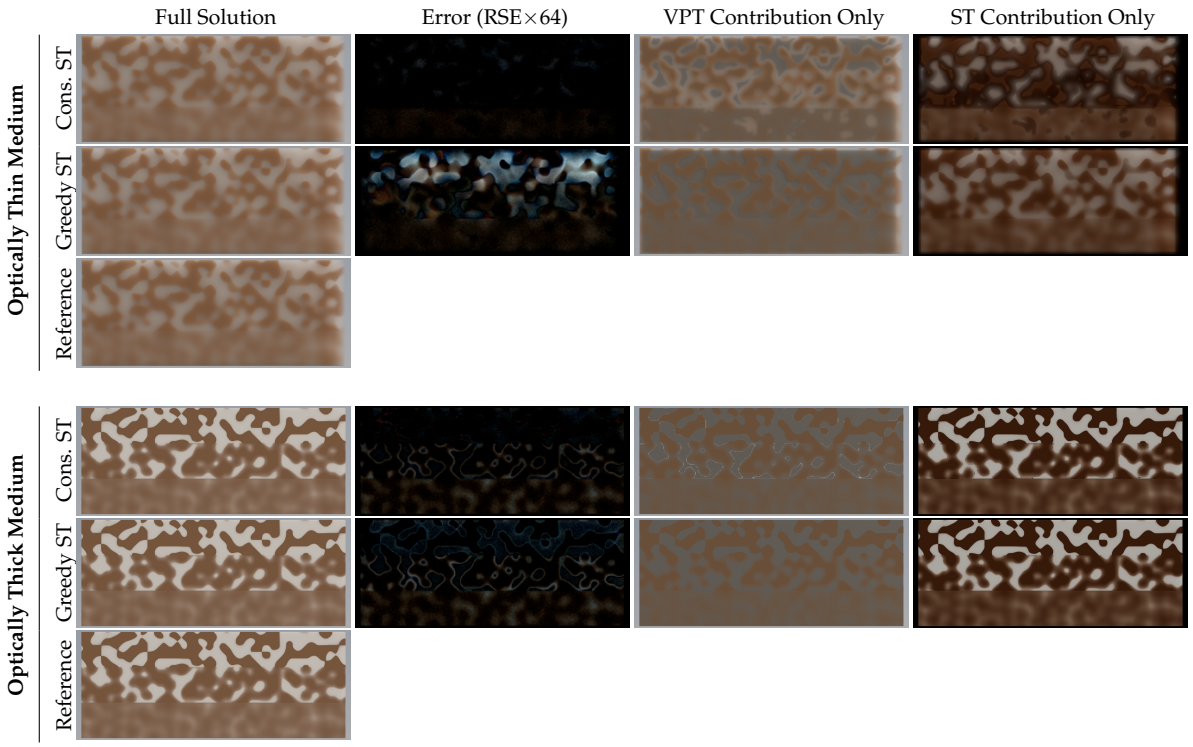


Figure 6.6: *Conservative (first row) and greedy (second row) usage of STFs in a wedge with decreasing thickness (from left to right). Heterogeneity is modeled using a Perlin noise with a hard (top third), soft (middle third), and smooth (bottom third) transition. Relative squared error (middle left column) shows that using ST greedily, not respecting the heterogeneity, leads to larger error than a conservative algorithm in optically thin media. On the other hand, it provides greater speedup as larger portions of paths are constructed using STFs; see the right column visualizing the energy computed using STFs.*

difference images illustrate, how greedy shell tracing introduces additional error when compared to conservative shell tracing, i.e. respecting the heterogeneity. This effect is especially pronounced in optically *thin* media. In optically *thick* media, however, there is relatively small difference between greedy and conservative shell tracing. Since the primary benefit of shell tracing is the acceleration of high-order scattering in optically thick media, we found little merit in using conservative shells and instead select shells greedily in all other renderings in this chapter. Nevertheless, we expect that in heterogeneous media with higher spatial density variation as found in our scenes (e.g. an avalanche) the bias from greedy shell selection may increase, in which case conservative selection could offer a superior trade-off between performance and error.

6.3 Granular Materials

To accelerate the simulation of low-order scattering in granular materials, Müller [2016] propose to approximate the individual grains using tight spherical proxies with largely identical internal transport; up to the accuracy of the so-called "grain scattering distribution function" (GSDF), denoted f_g, which models the grain's appearance. The GSDF is effectively a BSSRDF [Nicodemus et al. 1992] that relates the incident radiance to differential outgoing radiance on the surface of the proxy sphere:

$$f_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} \to \mathbf{x}_\mathrm{o}, \omega_\mathrm{o}) = \frac{\mathrm{d}L_\mathrm{o}(\mathbf{x}_\mathrm{o}, \omega_\mathrm{o})}{L_\mathrm{i}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i})\,\mathrm{d}\Omega(\mathbf{x}_\mathrm{i})\,\mathrm{d}\Omega(\omega_\mathrm{i})}\,. \tag{6.10}$$

The differential of $\mathbf{x}_\mathrm{i}$ is taken with respect to the solid-angle measure because the GSDF is defined in units of the grain size, causing $\mathbf{x}_\mathrm{i}$ to always reside on the unit sphere $\mathcal{S}^2$. We will now briefly introduce Müller's GSDF, followed by our automatic switching criterion between explicit path tracing and proxy path tracing.

6.3.1 Definition of the GSDF

The GSDF is defined as the sum of two terms, one for uncollided-flux (i.e. light that hits the grain's bounding sphere, but misses the grain itself—a delta component; superscript 0) and one for scattered transport (superscript $+$):

$$\begin{aligned} f_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} \to \mathbf{x}_\mathrm{o}, \omega_\mathrm{o}) = {} & \alpha_\mathrm{g}^0(\mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\,\delta(\omega_\mathrm{i} + \omega_\mathrm{o}) \\ & + \alpha_\mathrm{g}^+(\mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\,p_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} | \mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\,. \end{aligned} \tag{6.11}$$

The PDF $p_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} | \mathbf{x}_\mathrm{o}, \omega_\mathrm{o})$ describes the distribution of spatio-angular locations of incident radiance with respect to a given location of the outgoing radiance. In other words, p_g represents the shape of the incident light field contributing to a given coordinate $(\mathbf{x}_\mathrm{o}, \omega_\mathrm{o})$ in the outgoing light field. Unfortunately, p_g is 8-D in its full form, making tabulation of the GSDF for arbitrary grains impractical. To this end, Müller introduced a number of assumptions that reduce the 8-D space to two independent 3-D subspaces.

Random-Orientation Assumption. First, Müller [2016] assumes that each light path interacting with a proxy experiences the grain under a different random orientation. The blurry visual effect of this is best understood by looking at the middle row in Figure 6.7.

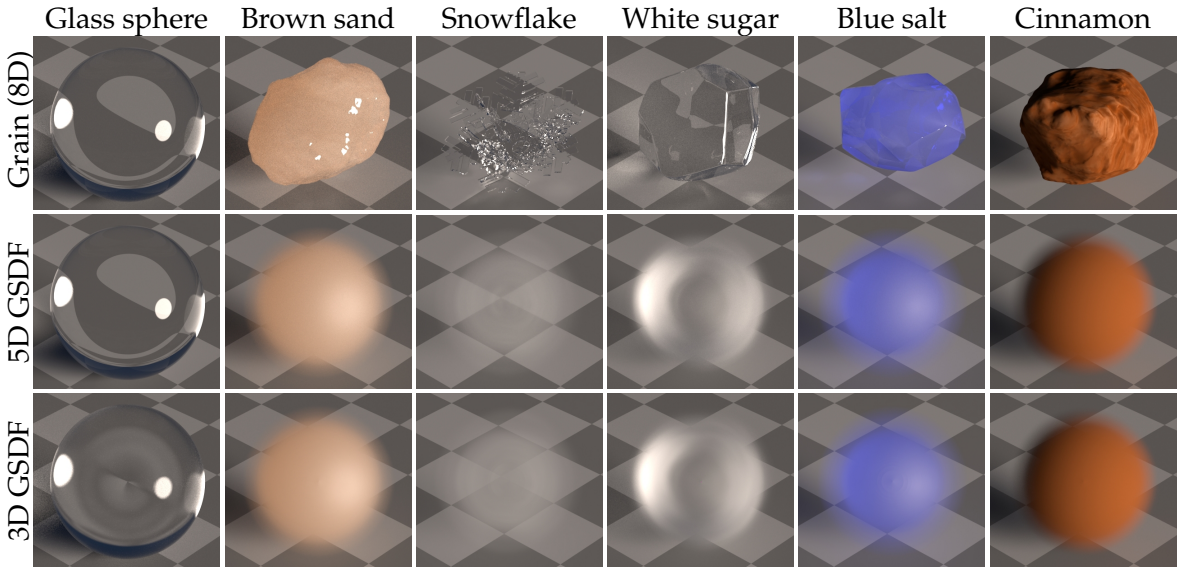


Figure 6.7: *Appearance of various grains (top row) and their corresponding GSDFs (bottom row). The middle row shows the impact of the random-orientation assumption without assuming separability of the directional and spatial components of the GSDF.*

Mathematically, the approximation averages the grain appearance over all sphere surface points $\mathbf{x}_o$ and rotations about their respective normal $N(\mathbf{x}_o)$. Thus, the GSDF no longer depends on $\mathbf{x}_o$ and the azimuthal component γ_o of ω_o; see Figure 6.8 for an illustration. In terms of the *outgoing* light field, the GSDF varies only in the inclination angle $\beta_o = \cos^{-1}\left(\omega_o \cdot N(\mathbf{x}_o)\right)$.

$$f_g(\mathbf{x}_i, \omega_i \to \mathbf{x}_o, \omega_o) \approx \alpha_g^0(\beta_o)\,\delta(\omega_i + \omega_o) + \alpha_g^+(\beta_o)\,p_g(\mathbf{x}_i, \omega_i|\beta_o)\,. \tag{6.12}$$

The distribution of scattered radiance $p_g(\mathbf{x}_i, \omega_i|\beta_o)$ is thereby reduced to 5-dimensions, which is still too much. Müller [2016] further reduces the dimensionality by decoupling the spatial and angular dimensions of the *incident* light field and tabulating them independently.

Separability Assumption. In order to decouple the spatial and angular components, Müller [2016] assumes that the distribution of scattered radiance is separable, that is:

$$p_g(\mathbf{x}_i, \omega_i|\beta_o) \approx p_g^{\mathbf{x}}(\mathbf{x}_i|\beta_o)\,p_g^{\omega}(\omega_i|\beta_o)\,. \tag{6.13}$$

Introducing the separability is motivated by the fact that grains are relatively small and most incident illumination can often be assumed as purely directional, i.e. $L_i(\mathbf{x}_i, \omega_i) = L_i^{\omega}(\omega_i)$. Under directional illumination, the reflection

equation simplifies, allowing pre-integration over the spatial domain:

$$\begin{aligned} L_\mathrm{o}(\mathbf{x}_\mathrm{o}, \omega_\mathrm{o}) &= \int_{\mathcal{S}^2} \int_{\mathcal{S}^2} f_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} \to \mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\, L_\mathrm{i}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i})\, \mathrm{d}\Omega(\mathbf{x}_\mathrm{i})\, \mathrm{d}\Omega(\omega_\mathrm{i}) \\ &= \int_{\mathcal{S}^2} L_\mathrm{i}^\omega(\omega_\mathrm{i}) \int_{\mathcal{S}^2} f_\mathrm{g}(\mathbf{x}_\mathrm{i}, \omega_\mathrm{i} \to \mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\, \mathrm{d}\Omega(\mathbf{x}_\mathrm{i})\, \mathrm{d}\Omega(\omega_\mathrm{i}) \\ &= \int_{\mathcal{S}^2} L_\mathrm{i}^\omega(\omega_\mathrm{i})\, \hat{f}_\mathrm{g}(\omega_\mathrm{i} \to \mathbf{x}_\mathrm{o}, \omega_\mathrm{o})\, \mathrm{d}\Omega(\omega_\mathrm{i})\,, \end{aligned} \tag{6.14}$$

where the distribution of scattered radiance from Equation (6.12) can be represented, without changing the reflection integral, by a product of the directional component $p_\mathrm{g}^\omega(\omega_\mathrm{i}|\beta_\mathrm{o})$ and a constant. Evaluating the reflection equation thus requires only a 3-dimensional precomputation.

In reality, the incident illumination is not perfectly directional, e.g. light arriving from nearby grains. Müller [2016] thus does not assume perfectly directional lighting, but only employs the separability assumption, i.e. Müller replaces the aforementioned constant by a 3-dimensional spatial distribution $p_\mathrm{g}^\mathbf{x}(\mathbf{x}_\mathrm{i}|\beta_\mathrm{o})$, as shown in Equation (6.13).

Figure 6.7 shows the type of blurring that the random-orientation and separability assumptions induce. While the first is responsible for the loss of geometric detail, the second assumption may result in further blurring of details seen through the grain (see the bottom image of the glass sphere in Figure 6.7). Despite the introduced approximations, the reduced-dimensional GSDF still preserves most of the important directional effects, such as local brightening and darkening due to reflection and refraction that are important to preserve characteristic visual features, such as sheen.

Minimal Approximation Impact. To ensure that the reduced-dimensional GSDF does not impair visual quality, Müller uses proxies only after a hand-tuned number of full-detail grain interactions. We propose to replace this hand-tuned parameter with an automatic heuristic.

6.3.2 Directional Approximation Error of the GSDF

Our goal is to *automatically* decide in which situations the GSDF provides a sufficiently good approximation, and when to employ explicit path tracing, e.g. to resolve glints. For this purpose, we estimate the GSDF's approximation error in the directional domain (i.e. when averaging grain appearances over all incident and outgoing path locations $\mathbf{x}_\mathrm{i}$ and $\mathbf{x}_\mathrm{o}$) during its precomputation as:

$$e_{f_\mathrm{g}}(\omega_\mathrm{i}, \omega_\mathrm{o}) = \left| f_\mathrm{g}(\omega_\mathrm{i} \to \omega_\mathrm{o}) - f_\mathrm{g}(\beta_\mathrm{i}) \right|. \tag{6.15}$$

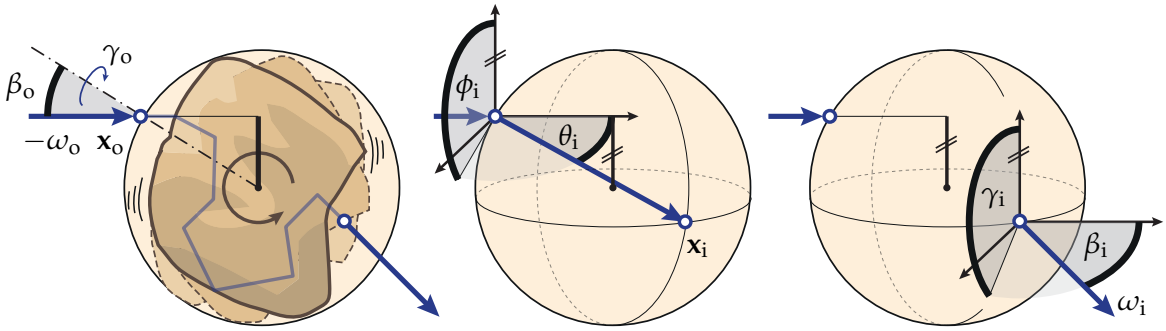


Figure 6.8: *The reduced-dimensional GSDF of Müller [2016] preserves the inclination angle β_o of light paths (the remaining dimensions of the outgoing light field are neglected), and decomposes the spatial $\mathbf{x}_i$ and directional ω_i components of the incident light field.*

For a given pair of incident and outgoing directions, this formula captures the absolute difference between the spatially averaged ground truth 8-D GSDF $f_g(\omega_i \to \omega_o)$, and the spatially averaged approximate 3-D GSDF $f_g(\beta_i)$. Note, that spatial averaging collapses the 3-D GSDF into a rotationally symmetric phase function, depending only on a single one-dimensional parameter $\beta_i = \cos^{-1}(-\omega_i \cdot \omega_o)$.

To illustrate the meaning behind the 4-D error function $e_{f_g}(\omega_i, \omega_o)$, we visualize it in Figure 6.9 for the "white sugar" grain of Figure 6.7. The figure shows polar slices through the error function at 25 fixed outgoing directions ω_o, parameterized by ω_i. Inspecting the figure shows, that it is primarily high-frequency detail (i.e. the glints that we want to detect) which is lost by the 3-D GSDF approximation. To confirm this observation, we show additional visualizations of the errors of the "glass sphere", "brown sand", "snowflake", and "cinnamon" grains in Section B.1.

In order to robustly limit the error during rendering, we therefore use the *maximum* approximation error to decide when to switch to proxy path tracing

$$E_{f_g} = \max_{\omega_i, \omega_o} e_{f_g}(\omega_i, \omega_o)\,. \tag{6.16}$$

This error directly corresponds to the worst-case appearance difference between a grain and its corresponding 3-D GSDF. The next section details how this error is used to choose between explicit and proxy path tracing.

6.3.3 Choice Between Explicit and Proxy Path Tracing

When a ray intersects the first grain bounding sphere, we decide whether to instantiate the full geometry (EPT) or its corresponding proxy (PPT) depending on how close the grain is to the camera and how good of an approximation

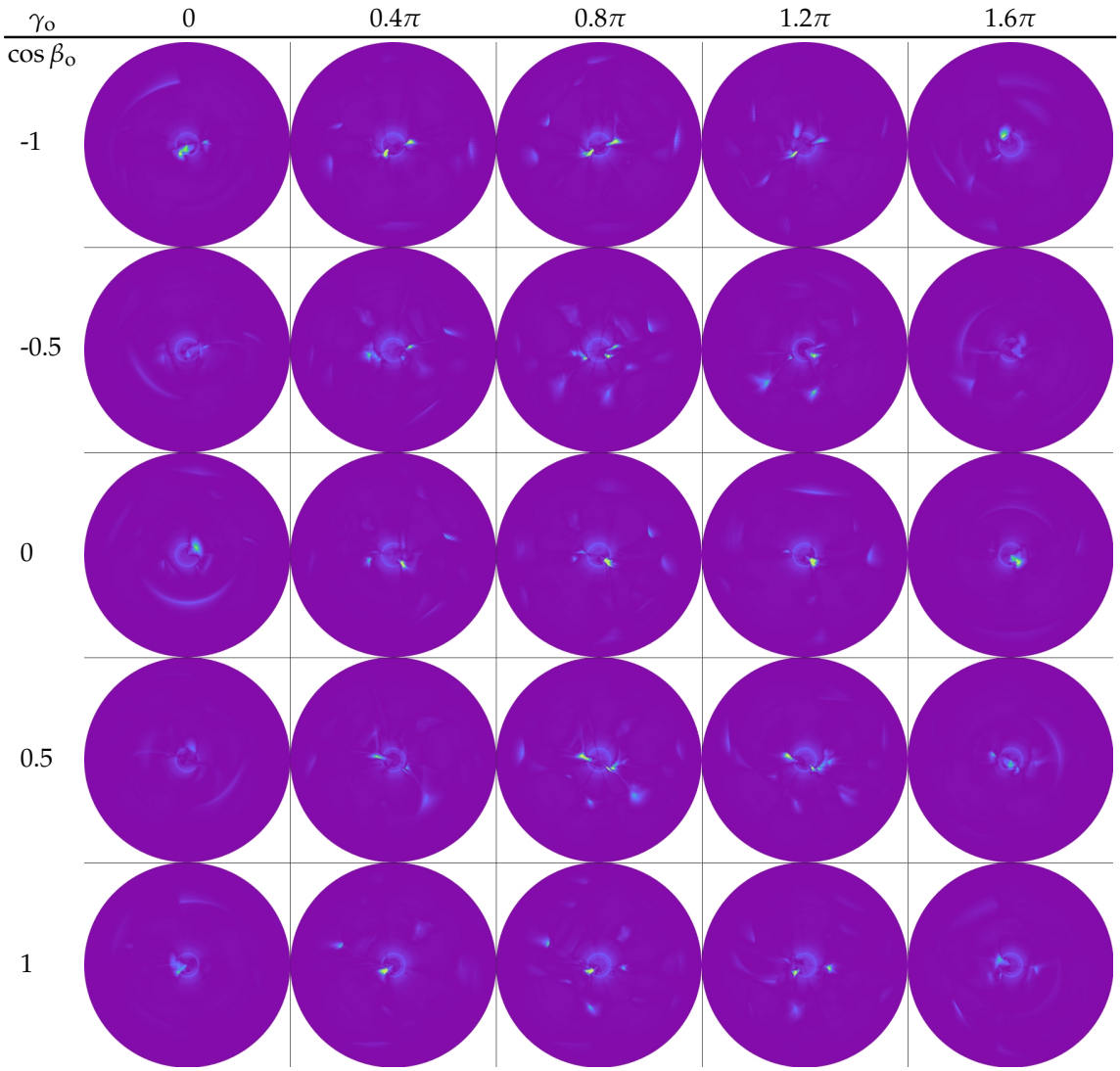


Figure 6.9: *Front-facing directional GSDF approximation error $e_{f_g}(\omega_i, \omega_o)$ on sugar grains. We visualize the 4-dimensional error function by plotting slices through $\cos\beta_o$ (vertical) and γ_o (horizontal). The individual circular heatmaps encode $\sin\beta_i$ as the distance from their center and γ_i as their rotational component.*

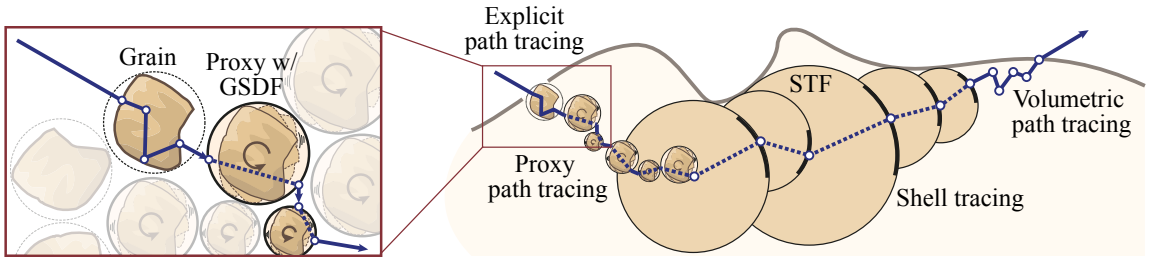


Figure 6.10: *We start building paths by (optionally) explicitly path tracing (EPT) to account for fine detail of grains and resolve high-frequency effects, e.g. glints. We then switch to the proxy representation by Müller [2016] which leverages GSDFs to accelerate grain-internal transport. To quickly simulate long paths, we employ shell tracing using our extension to STFs. Whenever the smallest shell does not fit inside the boundary of the assembly, we employ volumetric path tracing.*

our GSDF is for that particular grain's type:

$$\text{instantiate} \begin{cases} \text{explicit grain} & \text{if } \Omega_g > 4\Omega_{px} \text{ or } E_{f_g} > 0.1\,, \\ \text{proxy} & \text{otherwise}\,, \end{cases}$$

where Ω_g is the solid angle occupied by the grain's bounding sphere when viewed from the camera, and Ω_{px} is approximately the solid angle of the corresponding pixel. This allows our method to switch to PPT immediately for distant grains that are well approximated by their proxies. Furthermore, this criterion is only used for the first grain interaction of each light path. After interacting with a grain or its proxy once, further interactions will no longer instantiate explicit grains.

6.3.4 Full Grain Rendering Algorithm

Our rendering algorithm is a straightforward extension of the algorithm by Müller [2016]. It is based on standard unidirectional path-tracing. Rays originating from the camera traverse through the scene and query for the nearest grain bounding sphere intersection. When a ray intersects the first grain bounding sphere, we decide whether to instantiate the full geometry (EPT) or its corresponding proxy (PPT) based on our automated criterion. The following bounding sphere interactions are all resolved via proxy path tracing until the path reaches deeper than 1 mean-free-path (i.e. $1/\sigma_t$) into the continuous approximation of the granular medium [Meng et al. 2015; Müller 2016], at which point we switch to our extended shell tracing algorithm. After having begun with shell tracing, we must resort to volumetric path tracing near the boundary, where the smallest fitted shell no longer fits. This algorithm is illustrated in Figure 6.10.

6.4 Implementation

We implemented our method as a forward path-tracer in the open source Mitsuba renderer [Jakob 2010]. Like Müller [2016] we allow grains to be loaded explicitly—e.g. from a simulation—in addition to procedural instantiation. Our only two constraints on the grains are that they have to be oriented uniformly at random and that their bounding spheres must not overlap. Furthermore, we use the optimized procedural grain instantiation as described by Müller [2016] in order to accelerate ray-grain intersection queries. When comparing our method to the method of Meng et al., we use this optimization in both cases, such that our method does not have an unfair advantage that stems from implementation details.

Multi-channel rendering. STFs introduce complications in multi-channel rendering. Since we compute STFs at discrete radii, which are multiples of the mean-free-path of the medium, the sizes of shells in each channel do not match up if the extinction of the medium varies across channels. This means that we cannot trivially evaluate STFs across channels. We solve this problem by approximating STFs of arbitrary radii by linearly interpolating precomputed STFs with neighboring radii in logarithmic space. For example, if $f_{\text{stf}}(r_1)$ and $f_{\text{stf}}(r_2)$ were precomputed, and $r_1 \leq r < r_2$, then

$$f_{\text{stf}}(r) \approx (1 - w_r) f_{\text{stf}}(r_1) + w_r f_{\text{stf}}(r_2)\,, \tag{6.17}$$

$$w_r = \frac{\log(r) - \log(r_1)}{\log(r_2) - \log(r_1)}\,. \tag{6.18}$$

The usage of the logarithmic space is motivated by the exponential nature of continuous media, and by empirical experiments, e.g. as shown in Figure 6.5.

6.5 Results

In this section we compare our grainular-media-rendering method to reference images rendered with EPT and the algorithm of Meng et al. [2015]. To best illustrate the bias each method introduces we present results that are as converged as possible. Unfortunately, some reference images converge extremely slowly (e.g. the SNOWMAN reference in Figure 6.12 took 13 core-years to compute) and thus still exhibit residual noise. Due to the varying degrees of convergence across images, we do not report absolute rendering times, but instead use *time to unit variance* (TTUV). Additionally, we report *mean relative squared error* (MRSE). A comprehensive summary of the aforementioned metrics on the scenes presented in this chapter can be found in Table 6.2.

Table 6.2: *Time to unit variance (TTUV) and mean relative squared error (MRSE) for each scene presented in this chapter. We provide numbers for a reference solution, the method by Meng et al. [2015], and our method.*

Scene	Resolution	Method	TTUV (speedup)	MRSE
DUNES	2.07 MP	Reference (EPT)	8.74e-1 hrs	—
		Ours (EPT+PPT+ST)	**6.66e-2 hrs** (13.1×)	**7.25e-4**
		Meng et al. (EPT+VPT+DA)	3.28e-1 hrs (2.53×)	7.82e-3
SNOWMAN	1.04 MP	Reference (EPT)	1.67e-0 hrs	—
		Ours (EPT+PPT+ST)	**2.93e-4 hrs** (5699×)	**1.60e-4**
		Meng et al. (EPT+VPT+DA)	1.10e-2 hrs (152.×)	2.93e-3
TWO PILES	1.15 MP	Reference (EPT)	2.97e-1 hrs	—
		Ours (EPT+PPT+ST)	**7.17e-2 hrs** (4.14×)	**7.86e-4**
		Meng et al. (EPT+VPT)	8.30e-2 hrs (3.58×)	2.60e-3
BOWL (frame 100)	1.31 MP	Reference (EPT)	4.47e-3 hrs	—
		Ours (EPT+PPT+ST)	**7.88e-4 hrs** (5.67×)	**1.11e-4**
		Meng et al. (EPT+VPT)	9.36e-4 hrs (4.78×)	5.01e-4
BOWL (frame 1000)	1.31 MP	Reference (EPT)	2.17e-3 hrs	—
		Ours (EPT+PPT+ST)	7.79e-4 hrs (2.78×)	**1.19e-4**
		Meng et al. (EPT+VPT)	**7.27e-4 hrs** (2.98×)	5.53e-4
LUCY (1m large)	0.52 MP	Reference (VPT)	1.61e-3 hrs	—
		Ours (VPT+ST)	2.31e-4 hrs (6.96×)	**1.64e-4**
		Meng et al. (VPT+DA)	**1.93e-4 hrs** (8.35×)	1.25e-3
LUCY (100m large)	0.52 MP	Reference (VPT)	3.04e-3 hrs	—
		Ours (VPT+ST)	**1.61e-4 hrs** (18.9×)	**2.14e-4**
		Meng et al. (VPT+DA)	2.21e-4 hrs (13.8×)	1.19e-3

Time to Unit Variance. We define TTUV as the product of the time T it takes to render an image on a single CPU core and its mean pixel variance. With the TTUV it becomes trivial to compute the rendering time required for achieving a desired variance (dividing it by the desired variance) and vice versa, the variance achieved after rendering for a desired time (dividing it by the desired time).

Mean Relative Squared Error. The MRSE is defined as the mean $\frac{1}{N}\sum_{i=1}^{N}(v_i - \hat{v}_i)^2/(\hat{v}_i^2 + \epsilon)$ where p_i and $\hat{p}_i$ are the values of the i-th approximate and reference pixels, respectively, and ϵ is set to 0.01 in order to not over-emphasize very dark pixels and to prevent division by 0. We downscale images by a factor of 4 in both width and height via bilinear filtering before computing MRSE to reduce the influence of residual Monte Carlo noise in the reference image.

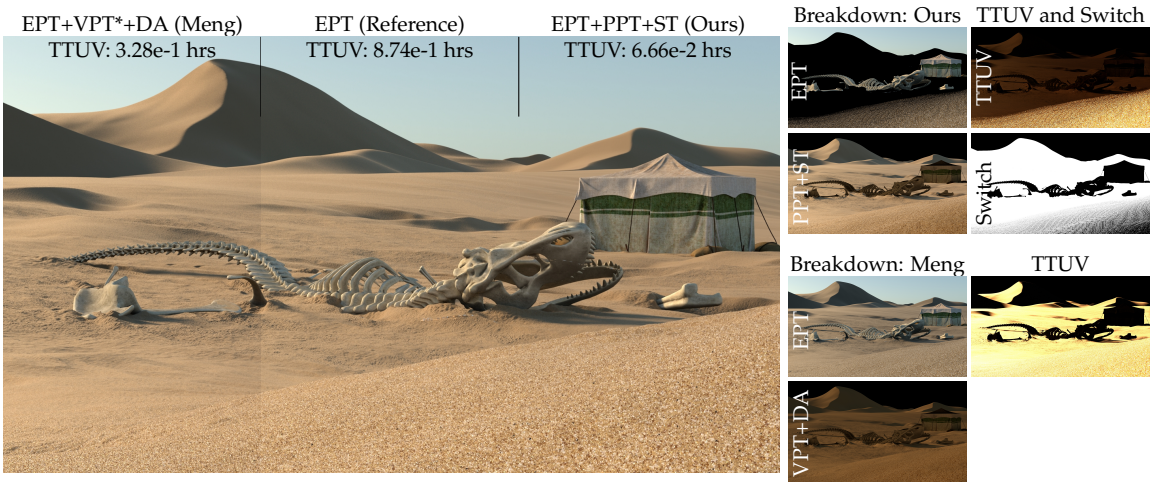


Figure 6.11: *The* DUNES *scene is a static desert landscape consisting of quadrillions of procedurally instantiated polydisperse sand grains with average radius of* 1 *mm. We render the scene with the method of Meng et al., with explicit path tracing (EPT), and with our method. Our technique is* 13× *faster than EPT, whereas the method of Meng et al. is only* 2.5× *faster. The images on the right depict contributions of EPT and approximations used by each method, per-pixel visualizations of TTUV, and the switching criterion for our method (black: EPT, white: PPT on the first bounce).*

6.5.1 Low-Order Light Transport

The DUNES scene consists of quadrillions of procedurally instantiated, polydisperse sand grains using the tile-based approach by Meng et al. [2015]. Each tile was generated by placing grains of normally-distributed sizes using a dart-throwing algorithm. The resulting distribution of grain radii has a the mean of 0.28 mm, standard deviation of 0.19 mm, and the minimum and maximum of 0.023 mm and 1 mm, respectively.

Each grain has a dielectric boundary and is filled with one of four distinct continuous homogeneous media. These grain types are uniformly distributed throughout the entire scene. In Figure 6.11 we show renderings of the DUNES scene with just EPT, with the method by Meng et al. [2015], and with our method. The scene showcases both the high-frequency detail of grains close to the camera and the large-scale appearance of grains at a large distance. For a reference of scale, the dinosaur skeleton is about 12 m long, and the large dunes in the distance are circa 1 km away.

Both alternatives to EPT are close to the ground truth, but the previous approach produces visibly darker colors. In terms of performance, however, our method converges 5× faster than the algorithm by Meng et al., and 13× faster than EPT. In fact, most of the variance in our method comes from the region in the bottom-right where the individual grains are close and we can

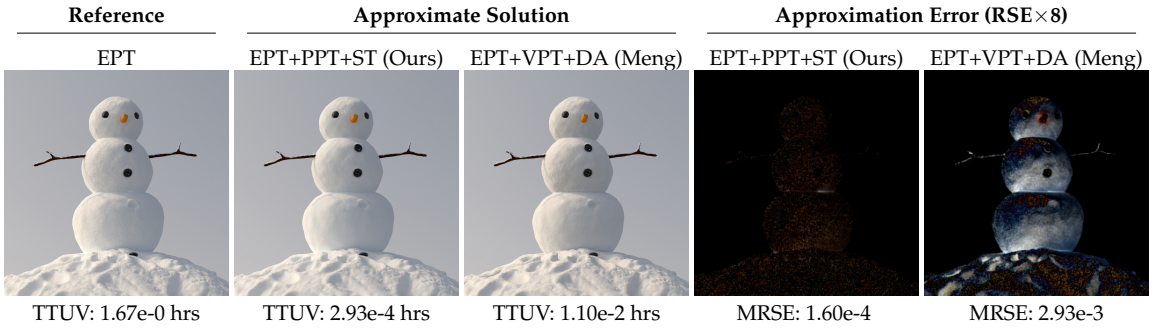


Figure 6.12: *We render the* SNOWMAN *scene by Meng et al. [2015] using their and our full method. Our method achieves significantly faster convergence (37.5×), because, in contrast, it can skip the expensive EPT prefix using our automatic switching criterion. Additionally, ST introduces less bias than the diffusion approximation of Meng et al. [2015], which we quantify as MRSE and visualize on the right.*

not use PPT from the beginning. When cropping away the bottom region of the image, or when neglecting glints, our method becomes about 80× faster than the method by Meng et al. Note, that even though the method of Müller [2016] could also accelerate rendering of this scene, it can not adaptively use EPT for the nearby grains and PPT for the far-away grains, resulting in either worse performance or in increased bias in the bottom region of the image.

6.5.2 High-Order Light Transport

The SNOWMAN scene by Meng et al. [2015] contains a 1 m tall snowman, made up from a homogeneous mixture of procedural mono-disperse snow grains, which have a radius of 0.1 mm. We render this scene in Figure 6.12 using our full method (EPT+PPT+ST) and the full method of Meng et al. [2015] (EPT+VPT+DA). Our method is able to render the scene 37.5× faster thanks to the automatic switching criterion that skips EPT and immediately employs PPT. The method of Müller [2016], although being able to achieve a similar speed-up, would require manually specifying the choice of skipping EPT in the scene configuration. We also compare the error introduced by Meng et al. [2015] and our method versus an explicitly path traced reference. Our method introduces over 18× less MRSE, which can be mostly attributed to the error-prone diffusion approximation.

6.5.3 Heterogeneity

The TWO PILES scene shows piles of blue and pink salt. The grains in each pile are polydisperse and distributed heterogeneously in layers based on a

Figure 6.13: *The* TWO PILES *scene consists of two heterogeneous piles made out of polydisperse salt grains. PPT (top) does not capture glints and achieves a speedup of* 28.6× *compared to the EPT reference. Our full method (EPT+PPT+STF) has an overall lower bias than PPT and is able to capture glints, with a smaller speedup of* 4.14× *over the EPT reference. On the right, we break down of the individual components of our method. EPT contributes mostly glints, PPT preserves most of the residual high-frequency granular structure, and ST captures low-frequency, high-order light transport.*

simulation which we performed with Houdini. The resulting distribution of grain radii has a the mean of 1.31 mm, standard deviation of 0.33 mm, and the minimum and maximum of 0.44 mm and 2.97 mm, respectively.

This scene demonstrates simultaneous usage of both of our novel features, which the previous state of the art does not support: automatic selection between EPT and PPT, and acceleration of high-order light transport in general heterogeneous mixtures. Note, that the method of Müller [2016] does support heterogeneous ST at the cost of a per-scene precomputation when only a small number of possible local grain mixtures exists. However, in scenes such as this one, there exist too many local grain mixtures to exhaustively precompute.

Figure 6.13 compares the convergence rates and appearances of EPT, PPT, and our full method (EPT+PPT+ST). PPT alone is 28.6× faster than EPT, whereas our full method only achieves a 4.14× speed improvement.

6.6 Discussion and Future Work

Variance and Per-sample Cost. The time per sample of our full method is only at most 5× larger than the per-sample time of rendering corresponding assemblies as *opaque diffuse* surfaces, suggesting there is only a relatively small potential for further optimizing our implementation. The per-sample variance of our method, however, is far higher than that of a diffuse surface. Future work should thus focus on better importance sampling techniques.

Reciprocity. Since we target forward path-tracing, we purposely did not enforce Helmholtz reciprocity to maximize the performance by retaining only the most visually salient features. As such, some parts of our tracing pipeline, e.g. the GSDFs, as well as the switching criteria and the order of individual accelerators would have to be revisited when incorporated into bidirectional algorithms.

Simulating Orientation of Grains. Currently, all dynamic simulations of granular materials that we know of do not simulate the rotational dynamics of grains. In our experience, however, the dynamic behavior of grain orientations is in fact quite important for low-order detail to look convincingly realistic. In the BOWL scene we hallucinate such rotations based on the absolute movement of grains, which yields only moderate realism. Ideally, the input simulation data would include physically plausible grain orientations.

Beyond Non-overlapping Bounding Spheres. By modeling grains within non-overlapping bounding spheres grains cannot be packed tighter than spheres. This assumption is not a big problem for approximately spherical grains, but it breaks down for anisotropic grains such as rice. Without the assumption of non-overlapping bounding spheres, neighboring grains influence the orientations of each other, breaking our assumption of randomly oriented grains.

6.7 Acknowledgments

We would like to thank Maurizzio Nitti and Alessia Marra for modeling the scenes and individual grains from this chapter. Furthermore, we thank Romain Prévost for help with illustrations, and the Stanford 3D Scanning Repository for providing the model used in the LUCY scene.

CHAPTER 7

Practical Path Guiding

This chapter is based on the following publication by Müller et al. [2017]:

Practical Path Guiding for Efficient Light-Transport Simulation
Thomas Müller, Markus Gross, Jan Novák
Computer Graphics Forum (Proc. EGSR), vol. 36, no. 4, pp. 91—100, Jun 2017

7.1 Overview

In this chapter, we present a robust, unbiased technique for importance sampling the rendering equation in a data-driven manner. This type of importance sampling has recently found renewed interest [Vorba et al. 2014; Vorba and Křivánek 2016; Herholz et al. 2016, 2018] and was dubbed "path guiding" due to systematically leading path-tracing algorithms towards regions of large contribution.

Like previous approaches, our algorithm *learns* an approximate representation of the 5-dimensional incident radiance field $L_\mathrm{i}(\mathbf{x}, \omega)$ and uses it to guide paths towards brightly lit scene regions. Unlike previous work, however, we perform this step *on the fly* during rendering with the goal of having the renderer produce low-variance images as quickly as possible after start up. In this context, we propose a principled method for combining learning and rendering computations: we partition the traced paths into subsets with geometrically growing power-of-two cardinalities and use only the *last* subset to synthesize the displayed image. This scheme prevents initial high-variance

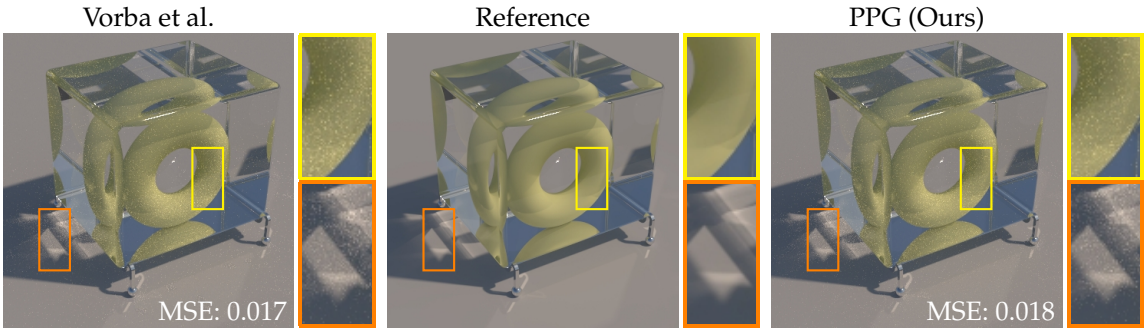


Figure 7.1: *Our method (PPG) allows efficient guiding of path-tracing algorithms as demonstrated in the* TORUS *scene. We compare equal-time (4.2 min) renderings of PPG (right) to previous work on path guiding [Vorba et al. 2014; Vorba and Křivánek 2016] (left). PPG automatically estimates how much training time is optimal, displays a rendering preview during training, and requires no parameter tuning. Despite being fully unidirectional, PPG achieves similar MSE values compared to Vorba et al.'s method, which trains bidirectionally.*

estimates (due to the learned distributions being less accurate in the beginning) from "polluting" the statistics of the final image. The power-of-two scheme ensures that at *least* half of the total computation time is used to synthesize the final image. We additionally derive a provably optimal (under mild assumptions) scheme for allocating a larger fraction of our compute budget to the final image by automatically detecting when the quality of our learned incident radiance approximation stops improving.

For approximating the spatio-directional incident radiance field, we introduce a hierarchical tree data structure, which we call spatio-directional-tree (dubbed *SD-tree*). It consists of an upper part—a binary tree that partitions the 3-dimensional spatial domain—and a lower part—a quadtree that partitions the 2-dimensional directional domain. The SD-tree models the incident radiance as a piecewise-constant function as described in Section 5.3. We design the construction and refinement rules of the SD-tree in such a manner that it works robustly and automatically throughout a multitude of scenes with difficult light-transport characteristics. Due to its simple tree structure, the SD-tree lends itself to efficient importance sampling and refinement such that its usage adds negligible computational overhead over other common rendering operations such as ray casting.

Our technique compares favorably on difficult-to-render scenes against a multitude of previous techniques: stochastic progressive photon mapping (SPPM) [Hachisuka and Jensen 2009], manifold exploration metropolis light transport (MEMLT) [Jakob and Marschner 2012], bidirectional path tracing [Veach and Guibas 1994; Lafortune and Willems 1993], and previous

work on incident-radiance-based path guiding [Vorba et al. 2014; Vorba and Křivánek 2016].

All our design decisions can be traced back to the goal of producing a robust path-guiding algorithm that does not require any tuning of its hyperparameters[1], hopefully making the algorithm suitable for movie production environments. For this reason, we dub our technique "Practical Path Guiding", abbreviated as PPG.

Our contributions in this chapter can be summarized as

- an on-line path tracing algorithm that learns the incident radiance field on the fly during rendering,
- a principled method for combining learning and rendering computations to achieve optimal variance,
- and a spatiodirectional data structure—the SD-tree—designed for robust adaptation to the incident radiance and for high efficiency.

7.2 Problem Statement and Related Work

Recall, that the amount of radiance $L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o})$ leaving point $\mathbf{x}$ in direction ω_o is quantified by the rendering equation [Kajiya 1986] as previously discussed in Section 4.3.1

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + \underbrace{\int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}| \, \mathrm{d}\Omega(\omega_\mathrm{i})}_{L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o})}, \tag{7.1}$$

where $L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o})$ is radiance emitted from $\mathbf{x}$ in direction ω_o, $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$ is *incident* radiance at $\mathbf{x}$ from direction ω_i, and f_s is the BSDF. Further recall, that unidirectional path tracing algorithms estimate the scattering integral $L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o})$ numerically via Monte Carlo integration

$$\left\langle L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o}) \right\rangle = \frac{L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) \, f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}|}{q(\omega_\mathrm{i} \,|\, \mathbf{x}, \omega_\mathrm{o})},$$

In this chapter we address the problem of reducing the variance of the Monte Carlo estimator $\langle L_\mathrm{s} \rangle$ via importance sampling. The ultimate goal of importance sampling is for q to be *proportional* to the integrand—the product

[1] Although we never observed the SD-tree occupying more than 20 MB in practice, we allow limiting its maximum memory footprint to a chosen amount, which could be considered a hyperparameter.

$L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \, |\cos \gamma_\mathrm{i}|$—which results in the estimator having a variance of zero. We refer back to Section 3.3.2 for more detail.

In the general case, however, being *perfectly* proportional to the integrand is not practical because it would require knowing the integral we would like to estimate beforehand. A common strategy is to break the integrand down into individual components and to devise sampling strategies for each of them. They can later be combined again to form a holistic sampling strategy that is *approximately* proportional to the integrand.

Since sampling proportional to just the BSDF f_s and the foreshortening term $\cos \gamma_\mathrm{i}$ is often possible we focus on the usually more difficult problem of sampling as-proportional-as-possible to L_i. BSDF/foreshortening sampling and approximate sampling from L_i can then be combined via multiple importance sampling [Veach and Guibas 1995] as previously explained in Section 3.3.4 or—even better if it is possible—by sampling from their product.

7.2.1 Prior Work on Path Guiding

The key idea behind path guiding techniques is to *reuse* information from an a-priori traced set of paths to devise an informed importance sampling strategy for paths that are traced later on. Jensen [1995] and Lafortune and Willems [1995] first introduced the concept of guiding camera paths. Jensen [1995] populates the scene with sampling points, each equipped with a hemispherical histogram from an a-priori traced set of photons. The histograms are then used to sample directions when building camera paths. Lafortune and Willems [1995] propose to rasterize incident radiance in a 5-dimensional tree for later usage as a control variate and for importance sampling. Budge et al. [2008] apply a specialized form of Jensen's technique to improve sampling of caustics and Hey and Purgathofer [2002] recognize that regular histograms are ill-suited for this type of density estimation and instead propose to average cones of adaptive width centered around the incident directions of photons. What all these approaches have in common is that they precompute the 5-dimensional incident radiance field in a *single* pre-pass prior to rendering.

Recently, Vorba et al. [2014] proposed to instead use an iterative reinforcement-learning-style algorithm, the key idea being that the precomputed radiance field can not only reduce the variance when rendering the final image, but *also* when computing another, more refined version of the radiance field. Performing this type of iterative scheme multiple times in a row allowed them to obtain vastly superior approximations of the incident radiance over

previous work. Their parametrization of incident radiance is a parametric gaussian-mixture model and they present a maximum a-posteriori scheme for fitting them to Monte Carlo estimates of incident radiance in an on-line fashion. Their follow-up work added adjoint-based Russian roulette [Vorba and Křivánek 2016] to the method.

Dahm and Keller [2018] take the concept of reinforcement learning further and combine rendering and training into the *same* algorithm rather than performing multiple training passes prior to rendering. They leverage similarities between the rendering equation and Q-learning to learn incident radiance in a more efficient fashion, albeit at the cost of additional bias in the learned distributions.

Only few attempts have been made to importance sample according to the full 7-dimensional product of incident radiance and the BSDF. One approach is building a discrete representation of the product on the fly [Lafortune and Willems 1995; Steinhurst and Lastra 2006], which is prone to miss high-frequency components of the BSDF and requires many potentially expensive BSDF evaluations. Alternatively, is it possible to leverage precomputations over a select set of BSDFs [Herholz et al. 2016] or a parametric class of BSDFs [Herholz et al. 2018], which unfortunately does not work in the general setting of encountering many previously unseen BSDFs.

In PPG, we adopt iterative reinforcement learning and—like the work by Dahm and Keller [2018]—we additionally combine learning and rendering into the same algorithm. Unlike previous work, we split the *entire* algorithm into distinct passes—each guided by the previous pass—in such a way that each pass remains unbiased and such that initial high-variance estimates (due to the learned distribution being less accurate) can not negatively impact the final image.

Due to multiple difficulties associated with learning the full 7-dimensional product, we focus on learning just the 5-dimensional incident radiance field $L_\mathrm{i}(\mathbf{x}, \omega_i)$ and combine it with sampling of the other terms via multiple importance sampling. We discuss potential extensions of our work to the product in Section 7.7.4 and we present an alternative path guiding technique that can naturally handle the full product in Chapter 8.

7.2.2 Datastructure for Storing Incident Radiance

The other big difference between our approach and previous work is our data structure: the SD-tree, which was specifically tailored for robustly approximating incident radiance given only noisy Monte Carlo estimates. Since

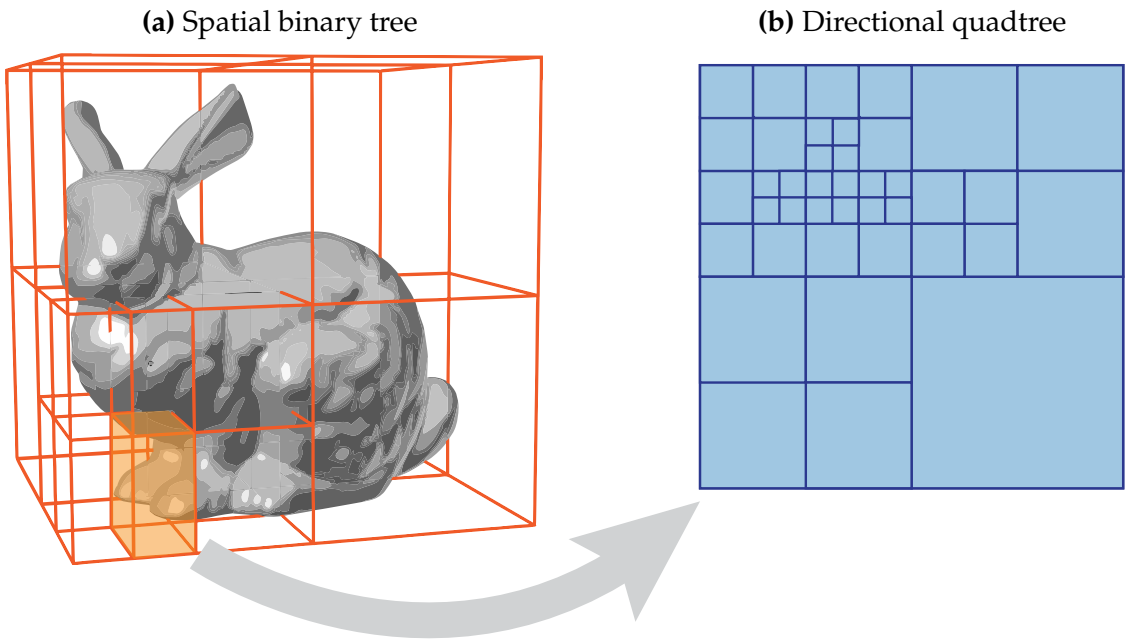


Figure 7.2: *The spatio-directional subdivision scheme of our SD-tree. Space is adaptively partitioned by a binary tree* ***(a)*** *that alternates between splitting the x, y, and z dimension in half. Each leaf node of the spatial binary tree contains a quadtree* ***(b)****, which approximates the spherical radiance field as an adaptively refined piecewise-constant function.*

our goal is to importance sample from a *directional* probability distribution *conditional* on a given spatial location, we treat the spatial and directional components of our datastructure differently. Hybrid data structures such as ours have been used successfully in the past in the field of rendering and geometry processing [Novák and Dachsbacher 2012; Boubekeur et al. 2006]. For the similar problem of *radiance caching*, Gassenbauer et al. [2009] used a spatial octree and a directional kd-tree to record individual radiance samples.

Rather than using an octree, we use a *binary* tree that splits each dimension in half via in a round-robin fashion. Our motivation is the smaller branching factor, which is essential in our progressive refinement scheme that ensures each *spatial* leaf node receives a roughly constant number of incident radiance samples (see Section 7.3.4).

We opt for a directional quadtree to retain the ease of fitting piecewise-constant approximations to Monte Carlo samples and sampling from them; we previously discussed piecewise-constant approximations and their fitting procedure in Section 5.3. With kd-trees [Gassenbauer et al. 2009] and gaussian-mixture models [Vorba et al. 2014; Herholz et al. 2016] it is generally more difficult to obtain a robust fit. The use of a tree structure, rather than a grid, is motivated by the incident radiance signal usually being very sparse,

leading to memory savings of multiple orders of magnitude in practice when compared to previous work [Jensen 1995]. We illustrate our SD-tree in Figure 7.2 and discuss our quadtree construction scheme that strives for equal amounts of flux in each leaf node in Section 7.3.3.

7.3 Path Guiding with SD-trees

PPG is based on iteratively learning an approximation of the incident radiance field

$$L_\mathrm{i}(\mathbf{x}, \omega) \approx \hat{L}_\mathrm{i}^{k}(\mathbf{x}, \omega)\,, \tag{7.2}$$

where the approximate incident radiance $\hat{L}_\mathrm{i}$ is represented by an SD-tree and $k \in \mathbb{N}$ is the iteration number in which $\hat{L}_\mathrm{i}$ is computed. We employ a reinforcement-learning-style scheme where the incident radiance of one iteration $\hat{L}_\mathrm{i}^{k}$ is learned from Monte Carlo estimates that are importance sampled via the SD-tree learned in the previous iteration $q \propto \hat{L}_\mathrm{i}^{k-1}$. This scheme greatly improves performance over naïve estimation of $\hat{L}_\mathrm{i}$ without any adaptation or reinforcement. We describe the collection of Monte Carlo estimates in an SD-tree in Section 7.3.1.

Once $\hat{L}_\mathrm{i}^{k}$ has been computed, it will not only be used for importance sampling the computation of $\hat{L}_\mathrm{i}^{k+1}$, which we describe in Section 7.3.4, but it will also affect the *topology* of the SD-tree capturing $\hat{L}_\mathrm{i}^{k+1}$ in a way that reflects newly gained information about the shape of the incident radiance field. How, precisely, this adaptivity is achieved is discussed in Section 7.3.2 for the spatial component and in Section 7.3.3 for the directional component.

7.3.1 Collecting Estimates of L_i

We train $\hat{L}_\mathrm{i}^{k}$ by following the procedure that we previously described in Section 5.3. When tracing paths throughout the scene we keep track of all the vertices encountered. Whenever a path is then connected to a light source—either by chance via BSDF sampling or any other importance sampling technique such as next event estimation—we accumulate incident radiance not only in the image plane but also in the leaf nodes of the current SD-tree $\hat{L}_\mathrm{i}^{k}$ at *all* vertices v of the path.

For each vertex, the SD-tree is first traversed spatially to determine the leaf node containing the vertex position $\mathbf{x}_v$. Then, the directional quadtree of

Algorithm 7.1: Refine our spatial binary tree post training iteration

```
function subdivide(leaf):
    numSamples ← getNumSamples(leaf)
    children ← splitAlongAlternatingAxis(leaf)
    forall childNode ∈ children do
        setNumSamples(childNode, numSamples / 2)
function refineBinaryTree(c, k):
    forall leaf ∈ leafNodes do
        if getNumSamples(leaf) > c · √(2^k) then
            subdivide(leaf)
```

said leaf node is traversed towards its own leaf node that contains ω_v, the direction in which the path was continued from $\mathbf{x}_v$. The Monte Carlo estimate of incident radiance is accumulated in *all* quadtree nodes that contain it, not just within the leaf. This has the effect of estimating incident radiance not just at the finest resolution, but also at all coarser scales. Effectively, this leads to a mipmap-style representation of incident radiance, only that the number of levels is spatially varying. This representation will be important later on for sampling directions proportional to the radiance stored in the quadtree.

Once $\hat{L}_\mathrm{i}^{\,k}$ has been fully populated, it will be used not only for importance sampling the construction of paths during the next iteration, but also to determine the topology of $\hat{L}_\mathrm{i}^{\,k+1}$. We *copy* $\hat{L}_\mathrm{i}^{\,k}$ to produce an initial version of $\hat{L}_\mathrm{i}^{\,k+1}$ on which we perform all topology refining operations. After the topology of $\hat{L}_\mathrm{i}^{\,k+1}$ has been refined, we reset the radiance in all its nodes to zero to prepare it for accumulating radiance estimates in iteration $k+1$.

7.3.2 Adaptive Spatial Binary Tree

We first discuss our subdivision scheme of the spatial binary tree, which controls the resolution at which we capture the *spatial* component of the incident radiance. To make it easier to reason about the performance of our datastructure, we split alternatingly between the x, y, and z dimensions, always in the middle of the node to be split.

As mentioned before, the overarching goal of PPG is to be robust and practical. To ensure robustness, it is paramount that the learned radiance distribution performs *consistently*. The goal of consistent performance is the backbone of our spatial subdivision criterion. To ensure the *directional* radiance distri-

butions have consistent quality across spatial leaf nodes, we would like all spatial leaf nodes—i.e. all directional quadtrees—to receive a roughly equal number of Monte Carlo estimates. We therefore make spatial subdivision directly depend on the number of samples in each node.

Specifically, we split a node if there have been at least $c \cdot \sqrt{2^k}$ path vertices. The number 2^k will be proportional to the number of traced paths in the k-th iteration—we will double the number of paths between iterations—and c is derived from the resolution of the quadtrees; we detail this in Section 7.5.4. After a node has been split, we copy its quadtree into *both* children and we temporarily set the number of samples in each child to *half* of that of their parent. We then recursively apply the subdivision criterion.

Post subdivision, all leafs contain roughly $c \cdot \sqrt{2^k}$ path vertices except for pathological cases. Therefore, the total number of leaf nodes is proportional to $\frac{2^k}{c \cdot \sqrt{2^k}} = \frac{\sqrt{2^k}}{c}$. Effectively, our threshold ensures that the total number of leaf nodes and the number of samples in each leaf both grow at the same rate $\sqrt{2^k}$ across iterations. The constant c trades off convergence of the directional quadtrees with spatial resolution of the binary tree. Pseudocode for this subdivision scheme is listed in Algorithm 7.1 and our choice of c is explained later in Section 7.5.4.

While refining the tree based only on the number of samples may seem rudimentary, it performs remarkably well since the iteratively learned distributions guide paths into regions with high contributions to the image; these thus get refined more aggressively than low-contribution regions due to the larger number of samples. Having a coarser radiance-function approximation in regions that receive fewer paths is tolerable, because the increase in relative noise is generally counteracted by the smaller contribution of such paths.

7.3.3 Adaptive Directional Quadtree

Like the spatial binary tree, we also update the topology of the quadtrees inbetween each iteration to better reflect the distribution of incident radiance. Our goal is to make the directional resolution *proportional* to incident radiance, such that high-power regions are represented in high quality and vice versa. This goal is equivalent with postulating an equal amount of flux—i.e. integrated radiance—in each leaf.

Let Φ be the total flux flowing through the quadtree. We then achieve a roughly-equal flux partitioning by pruning leaves of an interior node when its flux Φ_i—i being the node index—is smaller than a constant fraction ρ of

Algorithm 7.2: Refine our directional quadtree post training iteration

```
1  function subdivide(leaf):
2      Φ_leaf ← getFlux(leaf)
3      children ← splitQuadtree(leaf)
4      forall childNode ∈ children do
5          setFlux(childNode, Φ_leaf/4)
6  function refineQuadtree(nodes):
7      forall node ∈ nodes do
8          if getFlux(node)/Φ ≤ ρ then
9              pruneChildren(node)
10         else if isLeaf(node) then
11             subdivide(node)
```

Φ, i.e. if

$$\Phi_i < \rho\Phi. \tag{7.3}$$

Conversely, we subdivide any leaf node whose flux is greater or equal to the flux percentage

$$\Phi_i \geq \rho\Phi. \tag{7.4}$$

After subdividing a leaf node, we temporarily split its flux evenly over its children—all children will have a quarter of the original flux—and recursively apply the subdivision criterion to allow multiple subdivisions to occur at a time.

We list pseudocode for refining the topology of the quadtree in Algorithm 7.2. After all quadtrees have been refined, the leaves of the quadtrees contain at most a flux of $\rho\Phi$ and are therefore guaranteed to have *at least* a resolution proportional to their mean incident radiance. At this point we reset the flux in all nodes of the newly created quadtrees such that they can be populated with new estimates in the following rendering iteration.

The choice of ρ determines an additional constant factor on the resolution: the total number of leaves is roughly proportional to $1/\rho$. For choosing ρ, not only memory requirements are an important consideration, but also the fact that a larger resolution results in higher-variance Monte Carlo estimates of incident flux in each leaf. We found that a threshold of $\rho = 1\%$ works well in practice. We analyze its impact on the memory footprint in detail in Section 7.5.3.

Algorithm 7.3: Sample from our directional quadtree

```
function sampleQuadtree(node):
    if isLeaf(node) then
        return uniformRandomPositionIn(node)
    else
        childNode ← sampleChildByEnergy(node)
        return sampleQuadtree(childNode)
```

Algorithm 7.4: Compute PDF of sampling ω from our directional quadtree

```
function pdfQuadtree(node, ω):
    if isLeaf(node) then
        return 1/4π
    else
        childNode ← getChild(node, ω)
        α ← 4 · getFlux(childNode) / getFlux(node)
        return α · pdfQuadtree(childNode, ω)
```

7.3.4 Iterative Learning and Rendering

We train $\hat{L}_\mathrm{i}$ via path tracing during rendering, i.e. we use vertices from the *same* light paths that contribute to the image also for training $\hat{L}_\mathrm{i}$. Recall, that we train a *sequence* $\hat{L}_\mathrm{i}^{\,1}, \hat{L}_\mathrm{i}^{\,2}, \ldots, \hat{L}_\mathrm{i}^{\,M}, M \geq 1$ of SD-trees—one for each iteration $k \in \{1, \ldots, M\}$—each with light paths that are guided proportional to the SD-tree $\hat{L}_\mathrm{i}^{\,k-1}$ trained in the preceding iteration. This scheme often drastically accelerates the convergence compared to naïve Monte Carlo estimation. Only the first SD-tree $\hat{L}_\mathrm{i}^{\,1}$ is trained with an unguided path tracer: in our case unidirectional path tracing, but in practice it can use any other importance sampling scheme.

Given a path vertex v, we sample a direction from $\hat{L}_\mathrm{i}^{\,k-1}$ as follows. First, we descend through the *spatial* binary tree to find the leaf node containing the vertex position $\mathbf{x}_v$. Next, we sample the direction ω_v from the quadtree contained in the spatial leaf node via hierarchical sample warping as described by McCool and Harwood [1997]. We provide pseudocode for sampling in Algorithm 7.3 and for evaluating the PDF of a given sample in Algorithm 7.4.

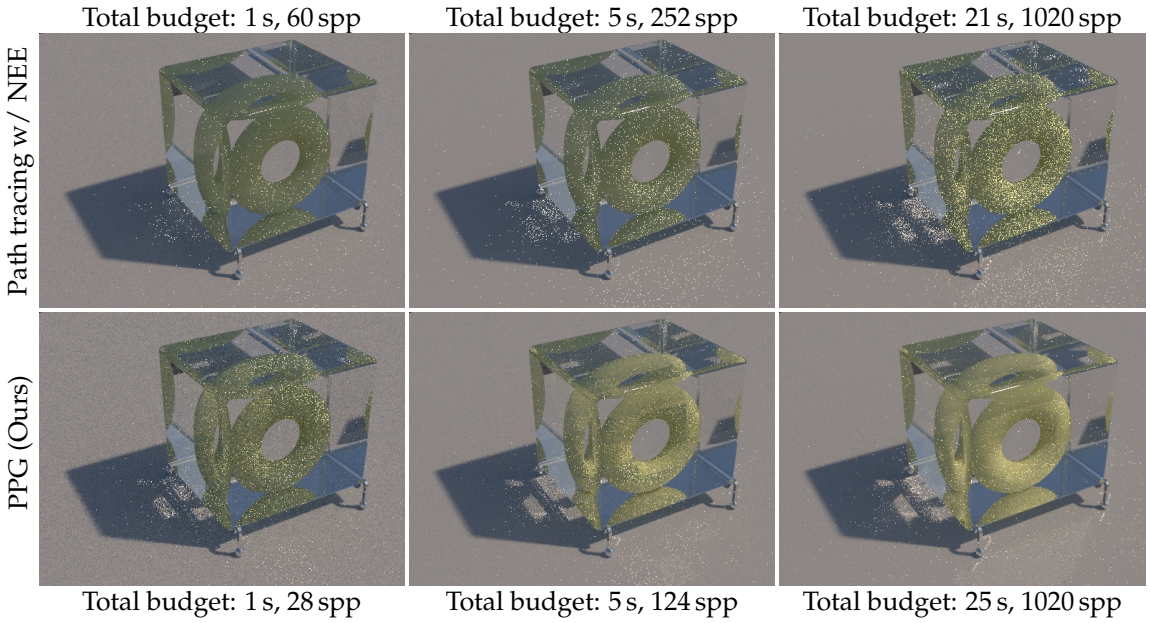


Figure 7.3: *PPG converges most rapidly in the beginning of rendering, yielding a reasonable preview of the final image far quicker than traditional path tracing with next event estimation (NEE). Except for Dahm and Keller [2018], existing path guiding approaches perform training in a separate pass, neglecting the rendering preview.*

7.3.5 Geometrically Growing Sample Count

To prevent high-variance estimates from initial iterations "polluting" the image, we propose to discard *all* images but the one produced by the last iteration. Additionally, to avoid wasting a large proportion of the total number of samples, we geometrically grow the number of samples in each iteration. We *double* the number of samples in each iteration, but in practice any other geometric growth factor $b > 1$ could be used, resulting in b^k samples being invested in the k-th iteration. The fraction of *discarded* samples is then

$$\left(\sum_{i=1}^{M-1} b^{i-1}\right) \Big/ \left(\sum_{i=1}^{M} b^{i-1}\right) = \frac{1-b^{M-1}}{1-b}\frac{1-b}{1-b^M} = \frac{1-b^{M-1}}{1-b^M}. \tag{7.5}$$

Since this fraction is monotonic in M, letting M go to infinity gives us an upper bound on the fraction of discarded samples

$$\lim_{M\to\infty} \frac{1-b^{M-1}}{1-b^M} = \frac{1}{b}. \tag{7.6}$$

In our special case of $b = 2$ we therefore never "waste" more than half of the total number of samples just on learning the sequence $\hat{L_\mathrm{i}}^1, \hat{L_\mathrm{i}}^2, \ldots, \hat{L_\mathrm{i}}^M$.

Another important property of doubling the sample count in each iteration surfaces when considering our spatial subdivision scheme. Since spatial subdivision of a binary-tree leaf node halves its volume, doubling the number of samples ensures that approximately the same number of samples reaches both new leaf nodes. Therefore, even locally, $\hat{L}_\mathrm{i}^{k}$ usually does not become noisier than $\hat{L}_\mathrm{i}^{k-1}$.

7.3.6 On-line Rendering Preview

To provide the user with quick visual feedback, we progressively display images synthesized using the path samples of the current iteration. Since each iteration starts rendering the image from "scratch", naïvely displaying the latest result would lead to sudden quality degradations whenever a new iteration starts. We avoid this by switching to the image of the current iteration only once it accumulated more samples than the previous iteration. In Figure 7.3 we demonstrate the rendering preview of the TORUS scene comparing against a path tracer.

7.3.7 Balancing Learning and Rendering

In this section, we describe how to split a given compute budget B, which can be defined either as time or number of samples, between learning and rendering such that the variance of the final image is minimized. For iteration k, we define the *budget to unit variance* $\tau_k = V_k \cdot B_k$, i.e. the product of variance of image I_k computed using paths traced in iteration k, and the budget B_k spent on constructing these paths. Variance V_k is computed as the *mean* variance of pixels in I_k. Assuming we keep using $\hat{L}_\mathrm{i}^{k}$ for guiding the paths until we reach B, we can estimate the variance of the *final* image as

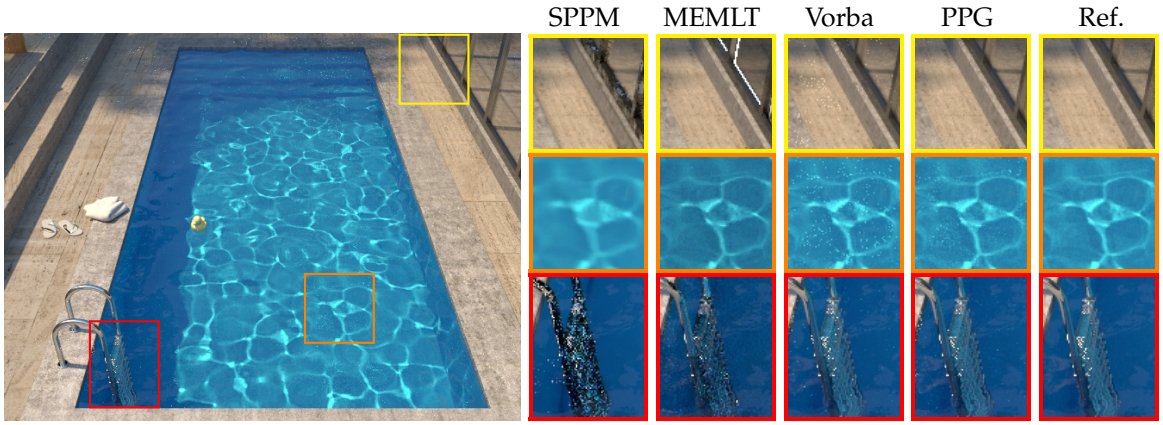
$$\hat{V}_k = \frac{\tau_k}{\hat{B}_k}, \tag{7.7}$$

where $\hat{B}_k$ is the remaining budget from the *start* of the k-th iteration:

$$\hat{B}_k = B - \sum_{i=1}^{k-1} B_i \,. \tag{7.8}$$

Our goal is to find the optimal iteration $\hat{k}$ that minimizes the final-image variance, i.e.

$$\hat{k} = \arg\min_k \hat{V}_k \,. \tag{7.9}$$



	SPPM	MEMLT	Vorba	PPG
Minutes (training + rendering):	0 + 7.5	0 + 7.5	5.3 + 6.4	1.1 + 6.4
Samples per pixel:	n/a	n/a	4264	5812
MSE:	0.936	2067.754	0.150	**0.144**

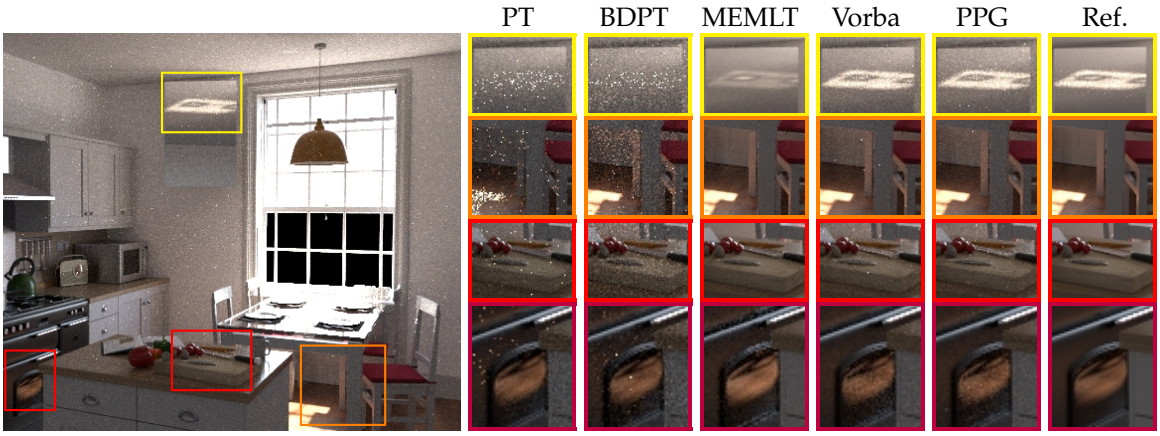
Figure 7.4: *Equal-time comparison of our method (PPG) versus previous work on the* SWIMMING POOL *scene; we report training + rendering time in minutes. The caustics inside the pool consist of difficult "specular, diffuse, specular" light transport that varies spatially due to the waves. The optimal training-rendering budgeting of PPG is in this case automatically determined to be* 15% *and* 85%*, respectively.*

To that end, we assume that training has monotonically diminishing returns; more precisely, the sequence τ_k is monotonically decreasing and convex. It follows that $\hat{V}_k$ is also convex (see Appendix C.1). We can then find $\hat{k}$ as the smallest k for which $\hat{V}_{k+1} > \hat{V}_k$ holds. Since we need to evaluate $\hat{V}_{k+1}$, we need to perform one more iteration than would be optimal, but the wasted computation is greatly outweighed by the variance reduction due to our automatic budgeting mechanism.

We can use a similar approach to optimally trade-off training and rendering when aiming for a target variance. In this case, we can estimate the *rendering* budget $\bar{B}_k$ required to reach a target variance $\bar{V}$ via $\bar{B}_k = \tau_k / \bar{V}$, and training is stopped whenever the *total* budget $\tilde{B}_k > \tilde{B}_{k-1}$, where

$$\tilde{B}_k = \bar{B}_k + \sum_{i=1}^{k-1} B_i \,. \tag{7.10}$$

This successfully finds $\hat{k} = \arg\min_k \tilde{B}_k$, because the sequence $\tilde{B}_k$ is convex whenever B_k is monotonically increasing, which is the case with the exponential sample count.



	PT	BDPT	MEMLT	Vorba	PPG
Minutes (training + rendering):	0 + 5.1	0 + 5.1	0 + 4.85	5.5 + 3.9	1.1 + 3.9
Samples per pixel:	3100	2560	n/a	1104	1812
MSE:	7.949	2.800	0.742	1.052	**0.694**

Figure 7.5: *The glass tabletop and the mirror next to the window in the* COUNTRY KITCHEN *scene challenge the efficiency of most light-transport algorithms. Unidirectional path tracing is unable to capture most of the non-trivial indirect-illumination patterns. Bidirectional path tracing struggles with constructing light sub-paths through the window (just like SPPM would). MEMLT avoids "fireflies" but has convergence issues. Both guiding methods are capable of finding difficult light paths, with our approach (PPG) performing similar to Vorba et al.'s despite being purely unidirectional.*

7.4 Results

We integrated our algorithm, dubbed PPG, into the Mitsuba renderer [Jakob 2010][2]. We compare PPG to several other approaches, including bidirectional path tracing [Veach and Guibas 1994], stochastic progressive photon mapping (SPPM) [Hachisuka and Jensen 2009], manifold exploration metropolis light transport (MEMLT) [Jakob and Marschner 2012], and, closely related to PPG, the technique by [Vorba et al. 2014; Vorba and Křivánek 2016], which represents incident radiance using gaussian-mixture models (GMM); we use an adjusted version of the authors' implementation for comparison. To ensure the path-guiding GMMs are properly trained, we always use 30 pre-training passes, 300000 importons and photons, adaptive environment sampling, and we leave all other parameters at their default values. In all comparisons images were rendered with an equal time budget. For PPG, training happens *within* its time budget. We *do not* count pre-training of the GMMs as part of their budget; we give GMMs as much *rendering* time as PPG uses.

Both path-guiding methods—Vorba et al.'s and PPG—render with unidi-

[2]The implementation is publicly available under the GPLv3 license at `https://github.com/Tom94/practical-path-guiding`.

rectional path tracing *without next event estimation* (NEE) to emphasize the difference in guiding distributions. Application to more sophisticated algorithms such as BDPT or VCM [Georgiev et al. 2012] would only mask the shortcomings of path guiding and obscure the comparisons. Lastly, none of the methods perform product importance sampling, since its benefits are orthogonal (and complementary) to path guiding. Extending our work to perform product importance sampling is discussed in Section 7.7.4.

The TORUS scene contains very long chains of specular interactions and a significant amount of *specular-diffuse-specular* (SDS) light transport, which is notoriously difficult to simulate with most unbiased algorithms. Path guiding approaches are able to learn and importance sample the high-frequency transport as long as it can be discovered by the underlying algorithm. In Figure 7.1 we render the scene with PPG comparing at equal-time to the method by Vorba et al. [2014]. The GMMs struggle with learning the correct distribution on the torus consistently, manifesting as uneven convergence; see Figure 7.11. PPG, overall still only achieves a slightly worse MSE as Vorba et al.'s method, while our automatic budgeting mechanism assigned 44 s to training out of the total 298 s compute time.

The SWIMMING POOL scene features difficult SDS light transport in a realistic scenario: rendering under-water caustics caused by waves. Standard (unguided) path tracing performs very poorly on this scene; the reference image in Figure 7.4 (right) took 18 h to render and still exhibits residual noise in some regions. By the nature of density estimation, the SPPM algorithm can handle such scenes without bright pixels ("fireflies"), but it instead struggles with preserving the sharpness of caustics and produces splotchy artifacts on the window frames. The manifold-walk-enhanced MLT (MEMLT) preserves the caustics' sharpness, but its uneven visual convergence manifests on the over-exposed window frame and darker pool ladder. Both guided unidirectional path tracers suffer from occasional outliers, but estimate the overall brightness more reliably than MEMLT and without the bias of SPPM. Compared to the method by Vorba et al., our SD-trees slightly reduce the number of "fireflies" as well as the average noise at roughly two thirds of the memory; see Table 7.1.

The COUNTRY KITCHEN scene in Figure 7.5 consists of various glossy materials and complex geometries that are lit by sunlight entering through a glass-free window and being reflected by a glass tabletop. The reflection of the sun on the ceiling viewed through the mirror—an SDS interaction depicted in the yellow inset—poses a challenge for most algorithms, including MEMLT. When properly guided, a simple unidirectional path tracer *without* NEE is capable of efficiently sampling these narrow, high-energy regions

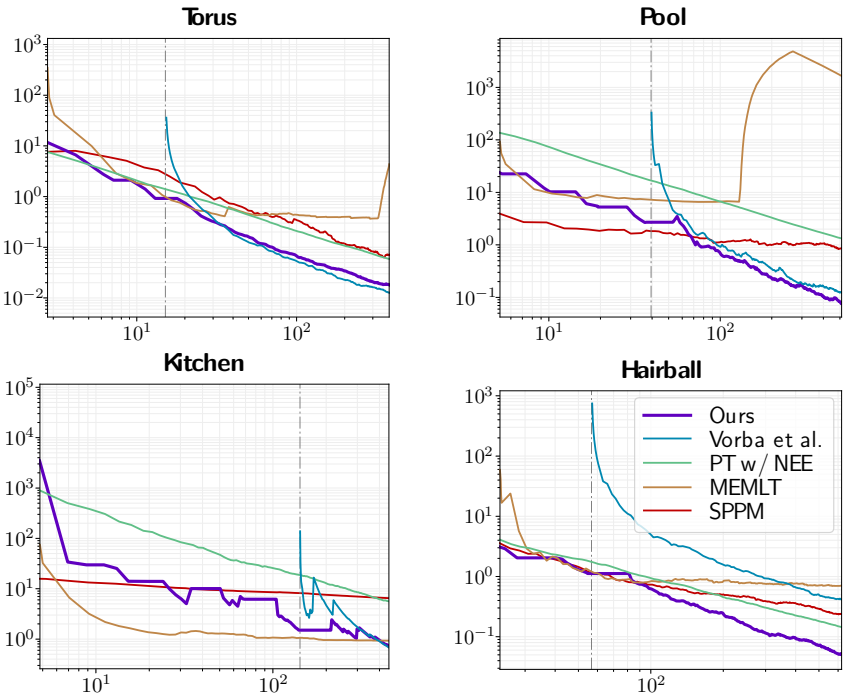


Figure 7.6: *Convergence of various algorithms in terms of* mean squared error *(MSE) plotted as a function of time. The dash-dotted vertical lines indicate when PPG (Ours) stops training and switches to render-only mode for the rest of the given budget: 5, 7.5, 5, and 20 min. In these plots the rendering component of Vorba et al.'s algorithm is synchronized with that of PPG.*

of path space even without Markov-chain mutations. Despite learning the incident light field only unidirectionally, PPG performs the best in terms of MSE thanks to the ability to render faster than Vorba et al.'s method. Our technique also requires slightly less memory in this scene.

7.5 Analysis

In this section, we analyze the convergence and memory usage and discuss the threshold for subdividing the binary tree.

7.5.1 Comparison to Other Methods

Figure 7.6 compares the convergence of our algorithm to several competing methods. Pre-training time of Vorba et al.'s algorithm is not included in these plots, but instead its rendering phase is synchronized with ours. In the beginning of the rendering process, MEMLT and SPPM often outperform PPG. However, the inherently unpredictable nature of MEMLT and

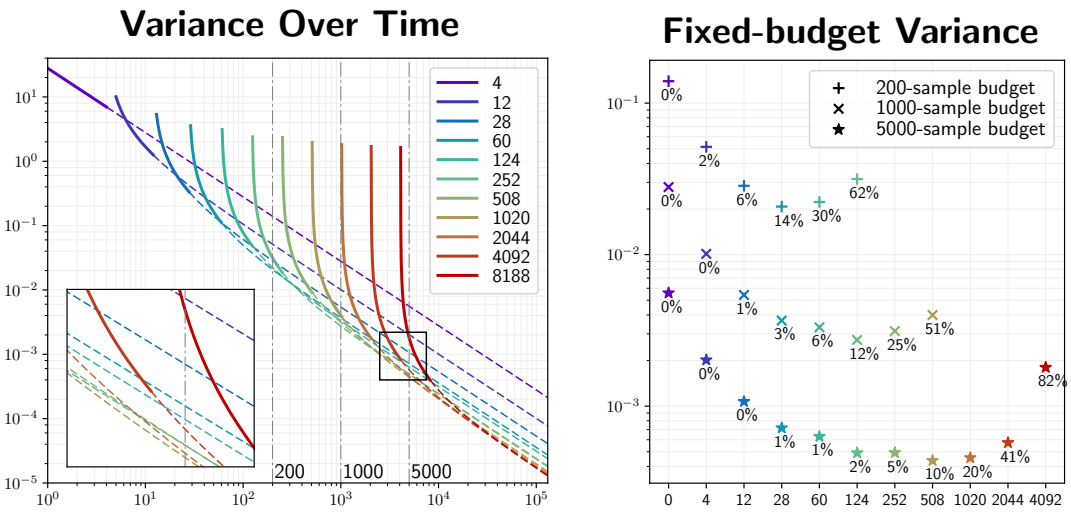


Figure 7.7: *Left: variance as a function of time plotted for subsequent training iterations. The dashed extension lines indicate convergence if we continued rendering, rather than beginning the next iteration. Right: intersections of the convergence curves with three dash-dotted lines representing three sample budgets. The intersections form convex sequences; the annotations show the percentage of the sample budget that was spent on training.*

the asymptotically slower convergence of SPPM allow PPG to eventually overtake. The "flat" regions in the training phase of our algorithm are due to delayed switching of the on-line preview to the result of the current iteration, as described in Section 7.3.4.

7.5.2 Convergence as a Function of Training Budget

Figure 7.7 demonstrates the convergence as a function of how many samples are used for training the guiding distributions. The curves represent the convergence of subsequent, geometrically growing iterations. Later iterations converge faster to a lower error thanks to their better-trained guiding distributions. When a fixed compute budget is given, our balancing scheme, as described in Section 7.3.7, automatically chooses when to stop the iterative training and continue rendering until the budget is exhausted. In the scatter plot on the right, we show the estimated final variance for the increasingly-trained distributions for the target budgets of 200, 1000, and 5000 samples. The three series empirically demonstrate the convexity of this optimization problem (the small deviations from a perfectly convex shape are due to the discrete refinement decisions).

Table 7.1: *Quantitative comparison against the method of Vorba et al. on all scenes in this chapter.*

Scene	Method	Memory	Training	Rendering	SPP	MSE
TORUS	Vorba et al.	2.5 MB	5.1 min	4.2 min	8932	**0.017**
	Ours	4.2 MB	0.73 min	4.2 min	11568	0.018
HAIRBALL	Vorba et al.	511.2 MB	60 min	17.3 min	1492	0.246
		7.2 MB	6.7 min	17.3 min	476	1.250
	Ours	6.1 MB	1.7 min	17.3 min	20596	**0.022**
SWIMMING POOL	Vorba et al.	12.8 MB	5.3 min	6.4 min	4264	0.150
	Ours	8.0 MB	1.1 min	6.4 min	5812	**0.144**
COUNTRY KITCHEN	Vorba et al.	23.8 MB	5.5 min	3.9 min	1104	1.052
	Ours	15.6 MB	1.1 min	3.9 min	1812	**0.694**

7.5.3 Memory Usage

The memory usage of our directional quadtrees can be estimated using the subdivision threshold ρ: with the energy-based refinement from Section 7.3.3 there can be no fewer than $1/\rho$ leaf nodes. The maximum number is unbounded for arbitrarily narrow energy distributions (e.g. a delta light source). We remedy this by limiting the maximum depth of the quadtree to 20, which is sufficient for guiding towards extremely narrow radiance sources without precision issues. The worst case of refining $1/\rho$ quads to the maximum depth results in the upper bound of $4 \cdot 20/\rho$ on the number of nodes.

We found $\rho = 0.01$ to work well, which in practice results in an average of roughly 300 nodes per quadtree. The maximum number of nodes across all quadtrees and results we tested is 792; well below the theoretical maximum of $4 \cdot 20/\rho = 8000$. Our quadtree nodes require 5 bytes of memory each, translating to an average footprint of 1.5 kb, and an upper bound of 40 kb, for each quadtree. A single mixture model with 8 gaussians requires 273 bytes.

According to the iterative training and rendering scheme, only two SD-trees $\hat{L}_i^{k-1}$ and $\hat{L}_i^{k}$ have to be kept in memory at the same time. However, because the spatial binary tree of $\hat{L}_i^{k}$ is merely a more refined version of the spatial tree of $\hat{L}_i^{k-1}$, it is straightforward to use the *same* spatial tree for both distributions, where each leaf contains *two* directional quadtrees; one for $\hat{L}_i^{k-1}$ and one for $\hat{L}_i^{k}$. This means, that only a single spatial binary tree has to be traversed for each path vertex.

In all our scenes, the full guiding distribution never exceeded 20 mb (see Table 7.1) without anyhow limiting the subdivision of the spatial binary tree.

However, we allow specifying an upper bound of nodes in the spatial tree to *ensure* memory requirements are kept. Once the limit is reached, the spatial tree is not subdivided further.

7.5.4 Binary Tree Subdivision Threshold

As described in Section 7.3.3, we subdivide a leaf node of the binary tree whenever it records more then $c \cdot \sqrt{2^k}$ path vertices during the previous iteration. We derive c from the desired number of samples s that each *quadtree leaf node* should receive. All quadtree leaves carry a roughly equal amount of energy and are thus sampled with similar probability during training. The average number of samples a quadtree leaf receives is then $s = S/N_l$, where S is the total number of samples drawn from a quadtree, and N_l is the number of leaf nodes in it. Our quadtrees have 300 nodes on average (see Section 7.5.3), and almost never more than that. We found, that $s = 40$ samples per quadtree leaf node results in a reasonably converged quadtree, and we thus choose c such that binary tree leaf nodes get subdivided after the first iteration ($k = 0$) when this number of samples is reached. We therefore obtain

$$c = \frac{s \cdot N_l}{\sqrt{2^k}} = \frac{40 \cdot 300}{1} = 12000\,. \tag{7.11}$$

7.6 Extensions

In this section we present two optional extensions to our PPG algorithm that were not used in the above experiments and analysis, but that can further enhance its performance. These extensions were developed in collaboration with the Walt Disney Animation Studios and were recently published as part of Chapter 10 of the *ACM SIGGRAPH 2019* "Path Guiding in Production" course [Vorba et al. 2019].

7.6.1 Increased Robustness by Filtering

The main advantage of our SD-tree radiance accumulation scheme is performance: it is very efficient to traverse the SD-tree towards a leaf node containing a particular vertex position $\mathbf{x}_v$ and direction ω_v for recording incident radiance estimates. Unfortunately, this scheme can result in visible variance discontinuities around the edges of spatial and directional leaf nodes, especially when the incident radiance field has high-frequency content

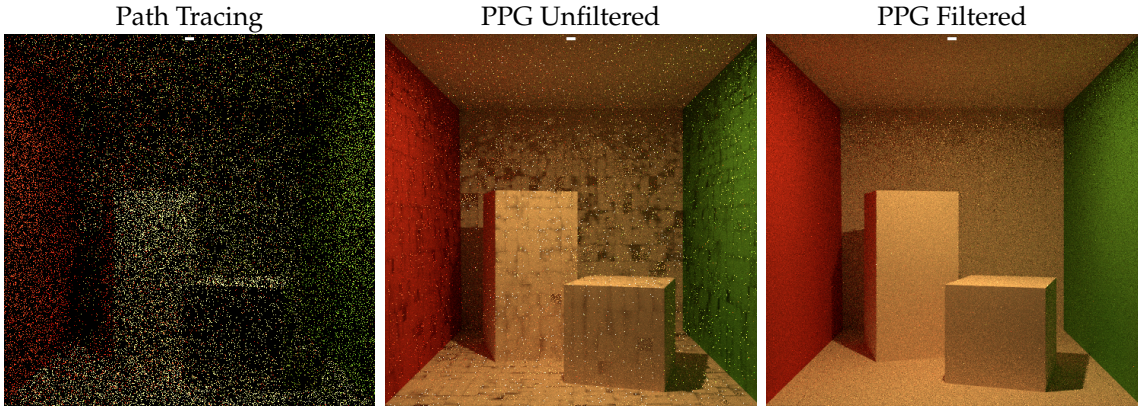


Figure 7.8: *Improved handling of narrow illumination by spatio-directional filtering of radiance estimates that are splatted into the SD-tree. We illustrate a contrived difficult situation: a small light source that is not sampled with next-event estimation (NEE). Although PPG (middle) dramatically improves overall variance compared to an unguided path tracer (left), residual noise is distributed non-uniformly along the spatial structure of the SD-tree, which is visually unpleasing. Spatio-directional filtering (right) addresses the structured noise by distributing it evenly across the scene and at the same time reducing it overall.*

in *both* the spatial and directional domain; see the middle image in Figure 7.8. To address this issue, we experimented with spatio-directional filtering of Monte Carlo estimates that improves robustness at additional computational cost.

Instead of recording radiance estimates within the leaf node that contains the vertex position $\mathbf{x}_v$ and direction ω_v, we record the estimates within those leaf nodes that fall into a *spatial neighborhood* around $\mathbf{x}_v$ and a directional neighborhood around ω_v.

More concretely, given an estimate, we begin by traversing the spatial tree to obtain the spatial footprint and corresponding volume V of the leaf containing ω_v; this footprint determines the filter radius and thereby the size of the spatial neighborhood. We then traverse the spatial tree again, this time visiting each node that has non-zero volume overlap with the spatial footprint from before, centered around $\mathbf{x}_v$. For each spatial leaf with non-zero volume overlap V_o that we find this way, we record the radiance estimate weighted by the fraction of overlapped volume V_o/V.

Directional filtering works analogously, only that we perform area-based filtering over the cylindrical domain as opposed to spatial filtering over space.

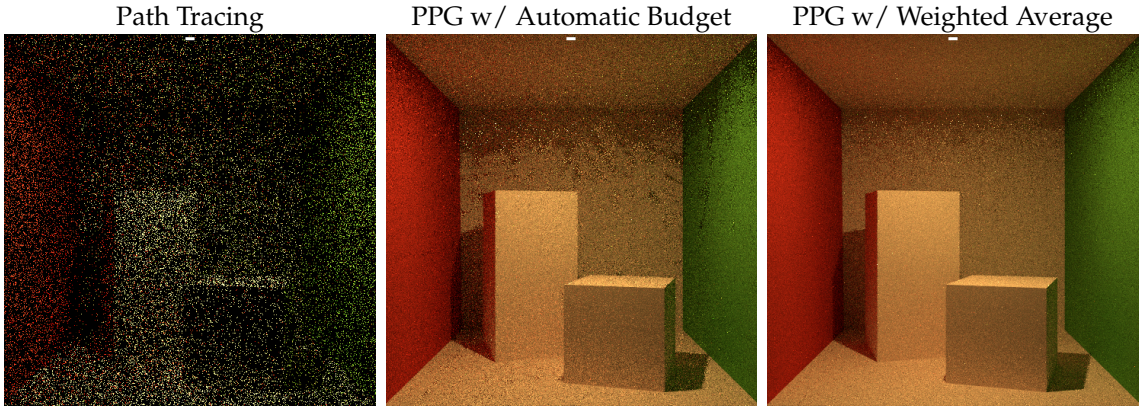


Figure 7.9: *Improved robustness by weighting the results of iterations rather than terminating training of the SD-tree early. We illustrate a contrived difficult situation: a small light source that is not sampled with next-event estimation (NEE). We compare unidirectional path tracing (left) against PPG with an automatically chosen training budget (middle; Section 7.3.7) and PPG with a weighted average of all iterations (right; Section 7.6.2) at an equal total sample count of 1024 spp.*

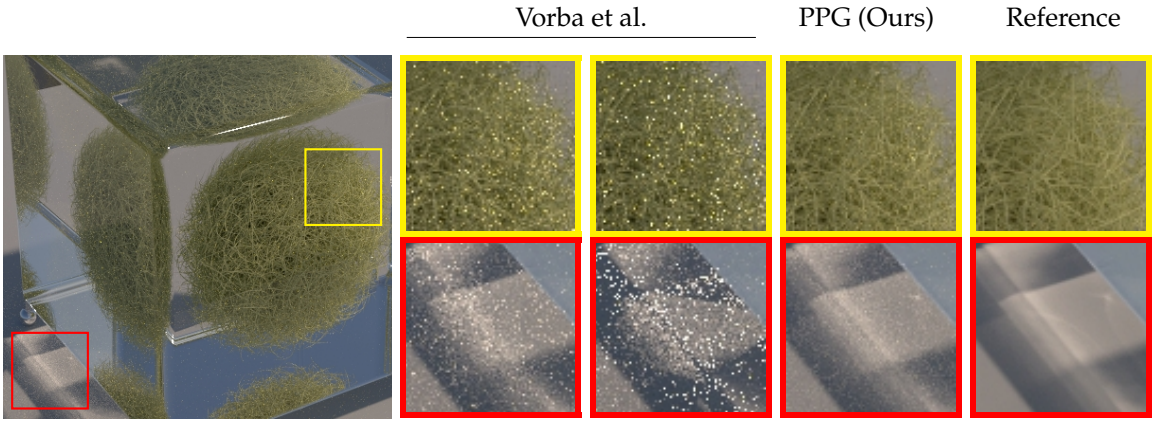
Stochastic Filtering. When implemented as described above, the filtering operation comes with significant computational cost. This cost can be avoided by replacing the deterministic filtering that traverses entire sub-trees with stochastic filtering that traverses only towards a single leaf. After obtaining the spatial footprint, the position $\mathbf{x}_v$ is simply jittered within the footprint's volume (again, centered around $\mathbf{x}_v$) and then recorded in the SD-tree as done in the original algorithm.

We found that performing only the spatial filtering stochastically and using deterministic filtering in the directional domain gave the best results at negligible computational cost. Figure 7.8 demonstrates the gained quality by spatio-directional filtering in a difficult lighting situation (a tiny quad light shining directly at a wall) at equal computation time.

7.6.2 Automatic Budget vs. Combining of All Iterations

In Section 7.3.7 we detailed how learning and rendering budgets can be optimally split under the assumption that the samples of previous iterations are discarded. We will now discuss an alternative in which we always iterate training until the very end and *combine* the resulting images of *all* iteration via carefully chosen weights.

To minimize the variance of the final image, the optimal weights to combine samples from all iterations are proportional to the samples' *inverse* variance. Let I^k_{xy} be the value of pixel x, y in the image produced by the k-th iteration,



	Vorba et al.		PPG (Ours)	Reference
Minutes (training + render):	60.0 + 17.3	6.7 + 17.3	1.7 + 17.3	
Memory overhead:	511 MB	7.2 MB	6.1 MB	
Samples per pixel:	1492	476	20596	
MSE:	0.246	1.250	**0.022**	

Figure 7.10: *Comparison of our method (PPG) and the method by Vorba et al. [2014] on a heavily tesselated hairball inside a glass cube. With its default parameters, Vorba et al.'s algorithm (left) exhibits worse performance and 83× higher memory consumption than PPG (right). The benefits of our approach become more prominent in an equal-memory comparison (middle vs right).*

then the corresponding pixel value of the final image with minimal variance is

$$I_{xy} = \left(\sum_{i=1}^{M} \frac{1}{\mathbb{V}\left[I_{xy}^{k} \right]} \right)^{-1} \sum_{i=1}^{M} \frac{I_{xy}^{k}}{\mathbb{V}\left[I_{xy}^{k} \right]} \tag{7.12}$$

Since we do not have access to the sample variance in closed form, we estimate it. Furthermore, since the variance of individual pixels is difficult to estimate robustly, we estimate the *mean* pixel variance of each image and use that in place of the per-pixel variance.

Using estimates of the variance unfortunately introduces a small amount of bias, because the variance estimates (and therefore the combination weights) *depend on the samples themselves*. However, due to our geometrically growing sample count across iterations, the variance estimate becomes more accurate in later iterations, making its correlation with the samples vanishes. This results in the weighting scheme being consistent.

In Figure 7.9, we demonstrate the benefit of the inverse-variance-weighted sample combination over utilizing our automatic budgetting scheme from Section 7.3.7.

7.7 Discussion

7.7.1 Spherical vs. Hemispherical Domain.

Our directional quadtree distributions cover the entire sphere of directions parameterized using world-space-aligned cylindrical coordinates. This has two key benefits compared to most previous work [Jensen 1995; Vorba et al. 2014; Herholz et al. 2016] that covers only the upper (oriented) hemisphere. Firstly, we do not need to discriminate distributions according to their angular distance to the normal at the shading point. This simplifies the search to merely selecting the spatially nearest distribution and avoids the need for rotating the distribution to prevent overblurring in the directional domain. Secondly, spherical distributions naturally generalize to volumetric path tracing and typically perform better on organic structures, such as foliage and hair.

We demonstrate this on the HAIRBALL scene in Figure 7.10 consisting of cylindrical hairs inside a glass cube. We compare our orientation-independent cylindrical parametrization to the hemispherical parameterization of Vorba et al. [2014]. To remain accurate, Vorba et al.'s method has to densely populate the hairball with hemispherical distributions, leading to a significant memory and performance cost. The cylindrical parametrization is decoupled from geometric properties, and results in almost 83× lower memory consumption and significantly improved convergence rate. Adjusting the parameters of Vorba et al.'s method to approximately match our memory consumption yields significantly slower convergence.

7.7.2 Quadtree vs. Gaussian Mixture Model

The main advantage of quadtrees over gaussian-mixture models is the increased robustness. The expectation-maximization algorithm utilized by Vorba et al. [2014] is not guaranteed to find the global optimum, and the distribution can vary dramatically across nearby spatial locations; see Figure 7.11. In contrast, our quadtrees adapt to the energy distribution hierarchically, top-down and adapt the resolution such that each leaf contains roughly the same amount of energy. The convergence of the rendering algorithm (within one iteration) is thus more stable.

Within Figure 7.11, at the location marked in yellow, Vorba et al.'s method exhibits high variance as the GMM failed to represent the light reflected by the glass cube. The distributions at the locations marked in orange and red demonstrate the instability of the GMM under similar incident illumination,

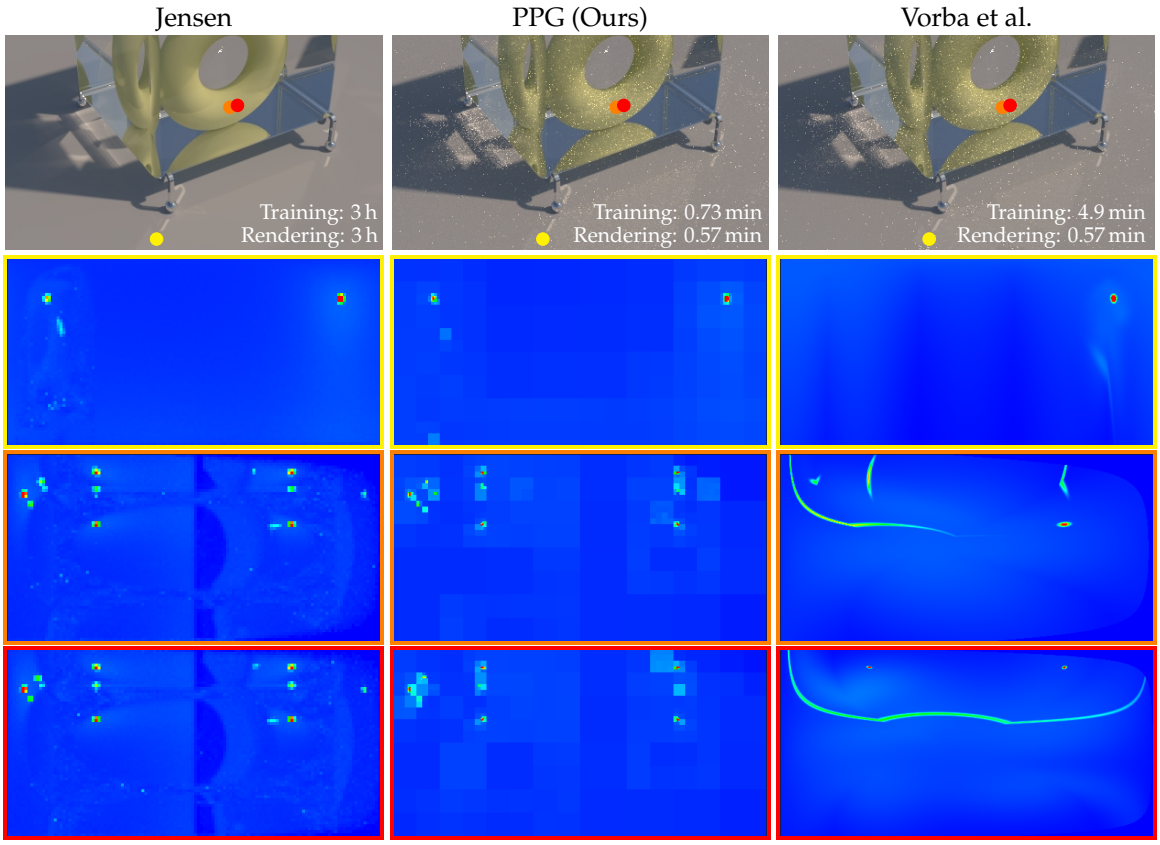


Figure 7.11: *We show directional radiance distributions discretized in the datastructure by Jensen [1995] (left), our quadtrees (middle), and the gaussian mixture model (GMM) by Vorba et al. [2014] (right). We used the same spatial locations to train all three directional distribution models. The left column was trained and rendered for 6 h to provide a reasonably converged reference. Our and Vorba et al.'s distributions have each been computed within 5 min.*

and its struggle to capture multiple modes robustly. Our quadtrees reliably capture all 5 important modes with high precision.

7.7.3 Temporal path guiding

We described an algorithm for guiding light transport in a static scene. For time-dependent effects, e.g. motion blur, it would be beneficial to refine the SD-tree also over the temporal domain. We suspect that adding a fourth dimension t to our spatial binary tree and including it in the dimension-alternating subdivision scheme is a straightforward extension which is capable of handling temporal effects.

7.7.4 Multiple and Product Importance Sampling

Sampling scattering directions purely from our quadtrees can lead to arbitrarily large variance in regions where the incident radiance is approximated badly. We avoid this problem by combining directions from our quadtrees with BSDF samples via multiple importance sampling, which guarantees that the variance can not be much worse compared to pure BSDF sampling. Although we use a multiple-importance-sampling ratio of 0.5 in this chapter, we demonstrate in Chapter 8 that it is possible to *lean* the ratio using gradient descent during rendering. The public implementation of PPG contains experimental code that explores this avenue.

The performance of sampling with our quadtrees could be further improved by ignoring quads in the bottom hemisphere. This can be done explicitly with a rectifying parameterization by Bitterli et al. [2015], though the mapping would no longer preserve areas.

Another possible solution is to generate several samples from the BSDF and the quadtree, and to then perform importance resampling [Talbot et al. 2005] towards their product. This technique never generates samples in the wrong hemisphere and approaches product importance sampling as the number of candidate samples for resampling approaches infinity.

Lastly, it might be possible to directly sample the product of the BSDF and the incident radiance. Herholz et al. [2016, 2018] demonstrate how product importance sampling between the BSDF f_s and $\hat{L}_\mathrm{i}$ can be achieved with Vorba et al.'s mixtures of gaussians by approximating all BSDFs in the scene also with mixtures of gaussians. We could similarly extend PPG by adopting the approach of Jarosz et al. [2009]. This would require converting the incident radiance distributions into Haar wavelets and representing BSDFs in spherical harmonics. Because our quadtrees only approximate the incident radiance, however, it is advisable to still combine the approximate product with BSDF sampling via multiple importance sampling to bound the error.

7.7.5 Guiding Sophisticated Sampling Algorithms

We presented results with guiding applied to a unidirectional path tracer. The deliberate choice of using a simple path tracer was made to best highlight the strengths and weaknesses of different guiding approaches; more sophisticated path-construction algorithms would only mask the shortcomings in regions where they already sample the integrand well. In practice, however, we recommend combining path-guiding with NEE via multiple importance

sampling due to the ease of implementation and low performance overhead. More sophisticated light-transport algorithms typically also benefit from path guiding as demonstrated by Vorba et al. [2014] for BDPT and VCM.

7.8 Acknowledgments

We thank Brian McWilliams for helpful discussions and insights into various machine learning topics that we considered for this chapter, and Ryusuke Villemin for valuable feedback; specifically, for pointing out the applicability of importance resampling. We further thank Disney's Hyperion team—Laura Lediaev, Joseph Schutte, Peter Kutz, Ralf Habel, Yining Karl Li, and David Adler—for their collaboration in developing the extensions presented in Section 7.6. We are grateful to Marios Papas for proofreading this chapter's text, to Olesya Jakob for remodeling the TORUS scene by Cline et al. [2005], to Ondřej Karlík for the SWIMMING POOL scene, to Blend Swap user Jay-Artist for the COUNTRY KITCHEN scene [Bitterli 2016] (CC BY 3.0), which we modified slightly, and to Samuli Laine for the Hairball model [McGuire 2011].

CHAPTER 8

Neural Importance Sampling

This chapter is based on the following work by Müller et al. [2018]:

Neural Importance Sampling
Thomas Müller Brian McWilliams, Fabrice Rousselle, Markus Gross, Jan Novák
ACM Trans. Graph. (to appear)

The above work is currently accepted for publication in the journal *ACM Trans. Graph.* and was already presented at *ACM SIGGRAPH 2019*.

8.1 Overview

In this chapter, we improve the effectiveness of path guiding by parameterizing the importance-sampling PDF $q(x)$ using neural networks. Generative neural networks have been successfully leveraged to draw samples from a to-be-learned probability distribution in many fields, including signal processing, variational inference, and probabilistic modeling, but their application to Monte Carlo integration—in the form of sampling PDFs—remains to be investigated. We will initially focus on the task of importance sampling *general* Monte Carlo estimators (Sections 8.2 through 8.5) and will only then apply our sampling PDFs to the task of path guiding (Section 8.6) by replacing the SD-tree in the practical-path-guiding framework of Chapter 7. Our neural PDFs are able to learn not only the incident radiance, but the full product of the scattering integral, yielding much lower variance than the

SD-tree of PPG. We also present an alternative technique to path guiding—primary-sample-space path sampling—where we, however, achieve only limited success compared to the path-guiding setting.

Since the effectiveness of a sampling PDF $q(x)$ to reduce variance via importance sampling heavily depends on how well it matches the distribution of the integrand $p(x) = f(x)/F$, it is crucial for the PDF's underlying model to be expressive enough to model the shape of $f(x)$ with high fidelity. Additionally, to avoid negatively affecting the performance of the underlying Monte Carlo estimator, generating samples $X_i \sim q$ must be *fast* (relative to the cost of evaluating f), and *invertible*. That is, given a sample X_i, we require an efficient and exact evaluation of its corresponding density $q(X_i)$—a necessity for evaluating the unbiased estimator

$$F \approx \langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i)}.$$

Being expressive, fast to evaluate, and invertible are the key properties of good sampling PDF models, and all our design decisions in this chapter can be traced back to these desired properties.

We focus on the general function-approximation setting from Section 5.2 where little to no prior knowledge about f is given, but f can be estimated at a sufficiently high number of points. Our goal is to extract the sampling PDF from these observations while handling complex high-dimensional distributions with possibly many modes and arbitrary frequencies. To that end, we approximate the ground-truth $p(x)$ using a generative probabilistic parametric model $q(x;\theta)$ that utilizes deep neural networks.

Our work builds on prior work that is capable of compactly representing complex manifolds in high-dimensional spaces and permits fast and exact sampling and PDF evaluation. More specifically, we extend the work of Dinh et al. [2014, 2016] who learn highly non-linear invertible mappings between an observation x and a latent variable z with fixed, prescribed PDF $q(z)$ by stacking a number of simple invertible mappings that are called *coupling layers*. The composition of many simple coupling layers can model a complex function in a similar fashion to how deep neural networks consisting of many simple layers can model complex functions. We introduce two new types of coupling layers, *piecewise-linear* and *piecewise-quadratic* (collectively referred to as *piecewise-polynomial*), that are much more expressive than the coupling layers used by Dinh et al. [2016]. This larger expressivity allows us to model complex functions using much fewer coupling layers, thereby reducing the total cost of training and evaluation.

After reviewing related work on generative modeling using neural networks in Section 8.2, we detail the framework of "non-linear independent components estimation" (NICE) Dinh et al. [2014, 2016] in Section 8.3, which forms the foundation of our approach. In Section 8.4, we describe a new class of *invertible piecewise-polynomial* coupling transforms that replace affine transforms proposed in the original work, and *one-blob*-encoded network inputs, which stimulate localization of computation and improve inference. We illustrate the benefits of our extensions on a few low-dimensional density-estimation problems and test the performance when learning the (high-dimensional) distribution of natural images. In Section 8.5, we demonstrate how NICE can learn from Monte Carlo estimates by deriving unbiased loss gradients of the KL and χ^2 divergences for use in a stochastic-gradient-descent-based optimizer as described in Section 5.4.2. Finally, we apply the proposed approach to light-transport problems in Section 8.6: we use NICE with our piecewise-polynomial coupling layers and our one-blob encoding to guide the construction of light paths and demonstrate that we outperform the state of the art at equal sample count in path guiding and primary-sample-space path sampling.

In summary, our contributions in this chapter are

- two *piecewise-polynomial* coupling transforms (piecewise-linear and piecewise-quadratic) that improve expressiveness,
- *one-blob*-encoded network inputs (a generalization of one-hot encoding) for improving learning speed and quality,
- stochastic gradients that can be used for optimizing the KL and χ^2 divergences when only Monte Carlo estimates of the unnormalized target distribution are available, and
- an application of NICE with the aforementioned tools to the problem of light-transport simulation.

8.2 Related Work

Since our goal is to learn a probability distribution and—among other things—draw samples from it, our approach falls into the category of *generative modeling*. Since we already provided an introduction to neural networks and surveyed prior usage of them in the field of physically based rendering in Section 5.4, we will focus in this section on generative neural networks outside of rendering and highlight their strengths and weaknesses for usage in Monte Carlo integration.

To recap, we are interested in techniques that allow training a neural network that parameterizes a probability density function $q(x;\theta)$ that approximates a true PDF $p(x)$ that is proportional to the integrand $f(x)$, using only using unbiased estimates of $f(x)$ at some finite number of data points x. Furthermore, the trained model $q(x;\theta)$ must permit efficient sampling and PDF evaluation.

Various prior methods on generative modeling were built on less stringent requirements. For example, it is often the case that only the *synthesis* of samples is required without an explicit need for $q(x;\theta)$ [Goodfellow et al. 2014; Germain et al. 2015; van den Oord et al. 2016a,b]. This disqualifies a large number of existing methods for our use case of Monte Carlo integration. In the following, we will therefore focus on only those existing techniques that show promise in satisfying our requirements.

The Latent-Variable Model. Many existing generative models rely on auxiliary unobserved "latent" variables z with fixed, prescribed PDF $q(z)$, where each possible value of z gives rise to a unique conditional distribution $q(x|z;\theta)$ that is learnable via parameters θ. Since any particular value of x can be caused by multiple different values of z, one must resort to integration to obtain $q(x;\theta)$

$$q(x;\theta) = \int q(x|z;\theta)q(z)\,\mathrm{d}\mu(z)\,, \tag{8.1}$$

In this context, $q(x;\theta)$ is referred to as the "marginal likelihood" and $q(z)$ as the "prior distribution". The prior is often modeled as a simple distribution (e.g. Gaussian or uniform) and it is the task of a computation graph parameterized by a neural network to learn the "likelihood" $q(x|z;\theta)$.

Unfortunately, the above integral is often not solvable in closed form, necessitating its estimation with another Monte Carlo estimator. It may be tempting to use such estimates of $q(x;\theta)$ in the denominator of our desired Monte Carlo estimator

$$\langle F\rangle_N \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{\langle q(X_i)\rangle}\,,$$

but this scheme introduces undesirable bias[1].

[1]This can be verified using Jensen's inequality.

Normalizing Flows. To avoid the limitation of having to estimate $q(x;\theta)$, a growing literature emerged that models x as a deterministic bijective mapping of z, a so-called "normalizing flow" $x = h(z;\theta)$. This has the effect of assigning only a *single* value of x to each possible value of z (and vice versa), thereby avoiding the difficult integration when computing the marginal-likelihood. Mathematically, $q(x|z;\theta)$ becomes a Dirac-delta function $\delta\big(x - h(z;\theta)\big)$, resulting in

$$q(x;\theta) = \int \delta\big(x - h(z;\theta)\big) q(z)\, \mathrm{d}\mu(z) = q(z) \left| \det\left(\frac{\partial h(z;\theta)}{\partial z^T} \right) \right|^{-1}. \tag{8.2}$$

Here, the inverse Jacobian determinant accounts for the change in density due to h in the infinitesimal neighborhood around x.

Although Equation 8.2 no longer contains a difficult integral, there exist a number of additional requirements on h to make the usage of $q(x;\theta)$ in Monte Carlo integration practical. Since $q(x;\theta)$ is expressed in terms of z, one must know the value of z that corresponds to x. Generally, this requires a tractable inverse $z = h^{-1}(x)$.[2] Additionally, to ensure efficiency, the evaluation of both h and h^{-1} as well as the corresponding Jacobian determinant must be *fast* relative to the cost of evaluating f.

Prior Work on Normalizing Flows. In the following, we mention a number of existing techniques based on normalizing flows that satisfy some (but not necessarily all) of our requirements. Rezende and Mohamed [2015] apply normalizing flows to model the posterior distribution in variational inference. Unfortunately, their computation graph is difficult to invert, not allowing the exact evaluation of $q(x;\theta)$.

Chen et al. [2018] propose a continuous analog of normalizing flows that utilizes the *instantaneous* change-of-variable formula, which only requires computing the *trace* of the Jacobian as opposed to the determinant. This reduces computational cost in many situations. Unfortunately, the evaluation of their continuous normalizing flows relies on a numeric ODE solver, which reintroduces computational cost in other places and results in approximation error when computing $q(x;\theta)$. This approximation error causes bias in our use case of Monte Carlo integration and therefore disqualifies their approach.

[2] Monte Carlo estimators that only sample from $q(x;\theta)$ are an exception, because they can simply use the z that generated x. However, when using auxiliary PDFs (as in multiple importance sampling) one must be able to evaluate $q(x;\theta)$ for x that were sampled externally. We note, that in the context of MIS, it is also possible *learn* the MIS weights rather than basing them on $q(x;\theta)$, which however invalidates the optimality guarantees of commonly used MIS heuristics.

Kingma and Dhariwal [2018] extend RealNVP with invertible 1×1 convolutions, achieving better results. The 1×1 convolutions, however, require a deep computation graph to be effective, which is oppsite to the shallow computation graph we desire for efficiency.

A number of recent approaches investigate the usage of normalizing flows for auto-regressive density estimation [Kingma et al. 2016; Huang et al. 2018; Papamakarios et al. 2017]. These "autoregressive flows" offer the desired exact evaluation of $q(x;\theta)$. Unfortunately, they generally only permit *either* efficient sample generation *or* efficient evaluation of $q(x;\theta)$, which makes them prohibitively expensive for our needs.

Lastly, "non-linear independent components estimation" (NICE) [Dinh et al. 2014, 2016] is a special case of autoregressive flows that allows *both* fast sampling *and* density evaluation through the use of so-called "coupling layers", the composition of which constituting a normalizing flow. Because of NICE's efficient sample generation and its efficient and exact density evaluation, NICE satisfies all our postulated requirements for usage in Monte Carlo integration and we therefore base our work on it. In the following section, we introduce NICE in detail and proceed with describing our piecewise-polynomial coupling layers that greatly increase its modeling capacity.

8.3 Non-linear Independent Components Estimation

In this section, we detail the works of Dinh et al. [2014, 2016] which form the basis of our approach. The authors propose to learn a mapping between the data and the latent space as a bijective compound function $h = h_L \circ \cdots \circ h_2 \circ h_1$, where each h_i is a relatively simple bijective transformation (warp). The choice of the type of h is different in the two prior works and in this chapter (details follow in Section 8.4), but the key design principle remains: h needs to be bijective with (computationally) tractable Jacobians. This enables exact and fast inference of latent variables as well as exact and fast probability density evaluation: recall, that given a differentiable mapping $h : \mathcal{X} \to \mathcal{Y}$ of points $x \sim p_{\mathcal{X}}(x)$ to points $y \in \mathcal{Y}$, we can compute the PDF $p_{\mathcal{Y}}(y)$ of transformed points $y = h(x)$ using the change-of-variables formula (Equation 2.45):

$$p_{\mathcal{Y}}(y) = p_{\mathcal{X}}(x) \left| \det \left(\frac{\partial h(x)}{\partial x^T} \right) \right|^{-1},$$

where $\frac{\partial h(x)}{\partial x^T}$ is the Jacobian of h at x. The cost of computing the determinant grows superlinearly with the dimensionality of the Jacobian. If $\mathcal{X}$ and $\mathcal{Y}$ are high-dimensional, computing $p_{\mathcal{Y}}(y)$ is therefore computationally intractable.

The key proposition of Dinh et al. [2014] is to focus on a specific class of mappings—referred to as *coupling layers*—that admit Jacobian matrices where determinants reduce to the product of diagonal terms.

8.3.1 Coupling Layers

A single coupling layer takes a D-dimensional vector and partitions its dimensions into two groups. It leaves the first group intact and uses it to parameterize the transformation of the second group.

Let $x \in \mathbb{R}^D$ be an input vector, A and B denote disjoint partitions of $[\![1, D]\!]$, and m be a function on $\mathbb{R}^{|A|}$, then the output of a coupling layer $y = (y^A, y^B) = h(x)$ is defined as

$$y^A = x^A\,, \tag{8.3}$$
$$y^B = C\big(x^B; m(x^A)\big)\,, \tag{8.4}$$

where the *coupling transform* $C : \mathbb{R}^{|B|} \times m(\mathbb{R}^{|A|}) \to \mathbb{R}^{|B|}$ is a separable and bijective map.

The invertibility of the coupling transform, and the fact that partition A remains unchanged, enables a trivial inversion of the coupling layer $x = h^{-1}(y)$ as:

$$x^A = y^A\,, \tag{8.5}$$
$$x^B = C^{-1}\big(y^B; m(x^A)\big) = C^{-1}\big(y^B; m(y^A)\big)\,. \tag{8.6}$$

If partition A was allowed to change, then the input to m for a correct inversion would be near impossible to know. The invertibility is crucial in our setting as we require both density evaluation and sample generation in Monte Carlo integration.

The second important property of C is separability. Separable C ensures that the Jacobian matrix is triangular and the determinant reduces to the product of diagonal terms; see Dinh et al. [2014] or Section D.1 for a full treatment. The computation of the determinant thus scales linearly with D and is therefore tractable even in high-dimensional problems.

8.3.2 Affine Coupling Transforms

Additive Coupling Transform. Dinh et al. [2014] describe a very simple coupling transform that merely translates the signal in individual dimensions

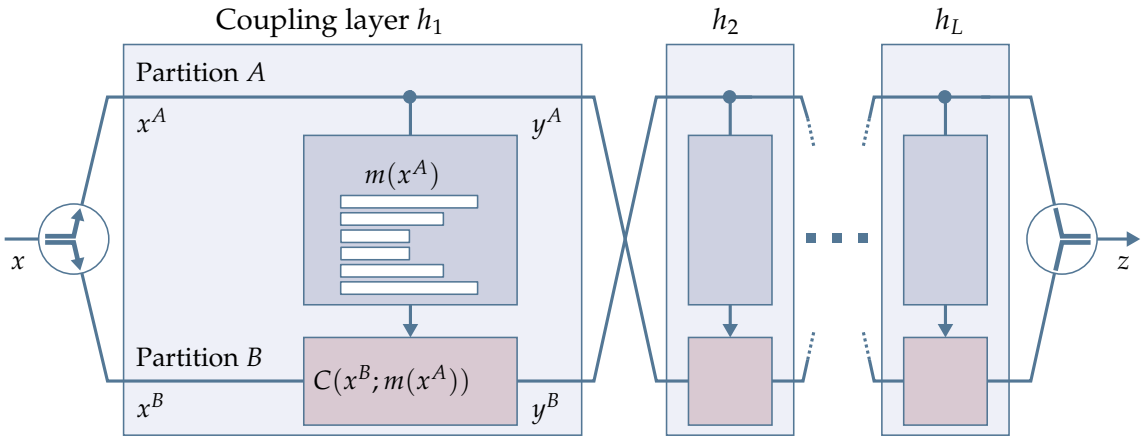


Figure 8.1: *A coupling layer splits the input x into two partitions A and B. One partition is left untouched, whereas dimensions in the other partition are warped using a parametric coupling transform C driven by the output of a neural network m. Multiple coupling layers may need to be compounded to achieve truly expressive transforms.*

of B:

$$C(x^B; t) = x^B + t\,, \tag{8.7}$$

where the translation vector $t \in \mathbb{R}^{|B|}$ is produced by function $m(x^A)$.

Multiply-add Coupling Transform. Since additive coupling layers have unit Jacobian determinants, i.e. they preserve volume, Dinh et al. [2016] propose to add a multiplicative factor e^s:

$$C(x^B; s, t) = x^B \odot e^s + t\,, \tag{8.8}$$

where $\odot$ represents element-wise multiplication and vectors t and $s \in \mathbb{R}^{|B|}$ are produced by m: $(s, t) = m(x^A)$. The Jacobian determinant of a multiply-add coupling layer is simply $\exp \sum s_i$.

The coupling transforms above are relatively simple. The trick that enables learning *non-linear* dependencies across partitions is the parametric function m. This function can be arbitrarily complex, e.g. a neural network, as we do not need its inverse to invert the coupling layer and its Jacobian does not affect the determinant of the coupling layer (cf. Section D.1). Using a sophisticated m allows extracting complex non-linear relations between the two partitions. The coupling transform C, however, remains simple, invertible, and permits tractable computation of determinants even in high-dimensional settings.

8.3.3 Compounding Multiple Coupling Layers

As mentioned initially, the complete transform between the data space and the latent space is obtained by chaining a number of coupling layers. A different instance of neural network m is trained for each coupling layer. To ensure that all dimensions can be modified, the output of one layer is fed into the next layer with the roles of the two partitions swapped; see Figure 8.1. Compounding two coupling layers in this manner ensures that every dimension can be altered.

The number of coupling layers required to ensure that each dimension can influence every other dimension depends on the total number of dimensions. For instance, in a 2D setting (where each partition contains exactly one dimension) we need only two coupling layers. 3D problems require three layers, and for any high-dimensional configuration there must be at least four coupling layers.

In practice, however, high-dimensional problems (e.g. generating images of faces), require significantly more coupling layers since each affine transform is fairly limited. In the next section, we address this limitation by providing more expressive mappings that allow reducing the number of coupling layers and thereby the sample-generation and density-evaluation costs. This improves the performance of Monte Carlo estimators presented in Section 8.6.

8.4 Piecewise-polynomial Coupling Layers

In this section, we propose piecewise-polynomial, invertible maps as coupling transforms instead of the limited affine warps reviewed previously. Specifically, we introduce the usage of polynomials with degrees 1 and 2, i.e. piecewise-linear and piecewise-quadratic warps. In contrast to Dinh et al. [2014, 2016], who assume $x, y \in (-\infty, +\infty)^D$ and Gaussian latent variables, we choose to operate in a unit hypercube (i.e. $x, y \in [0,1]^D$) with uniformly distributed latent variables, as most practical problems span a finite domain. Unbounded domains can still be handled by warping the input of h_1 and the output of h_L e.g. using the sigmoid and logit functions.

Similarly to Dinh and colleagues, we ensure computationally tractable Jacobians via separability. We transform each dimension independently:

$$C\left(x^B; m(x^A)\right) = \left(C_1\left(x_1^B; m(x^A)\right), \cdots, C_{|B|}\left(x_{|B|}^B; m(x^A)\right)\right)^T. \tag{8.9}$$

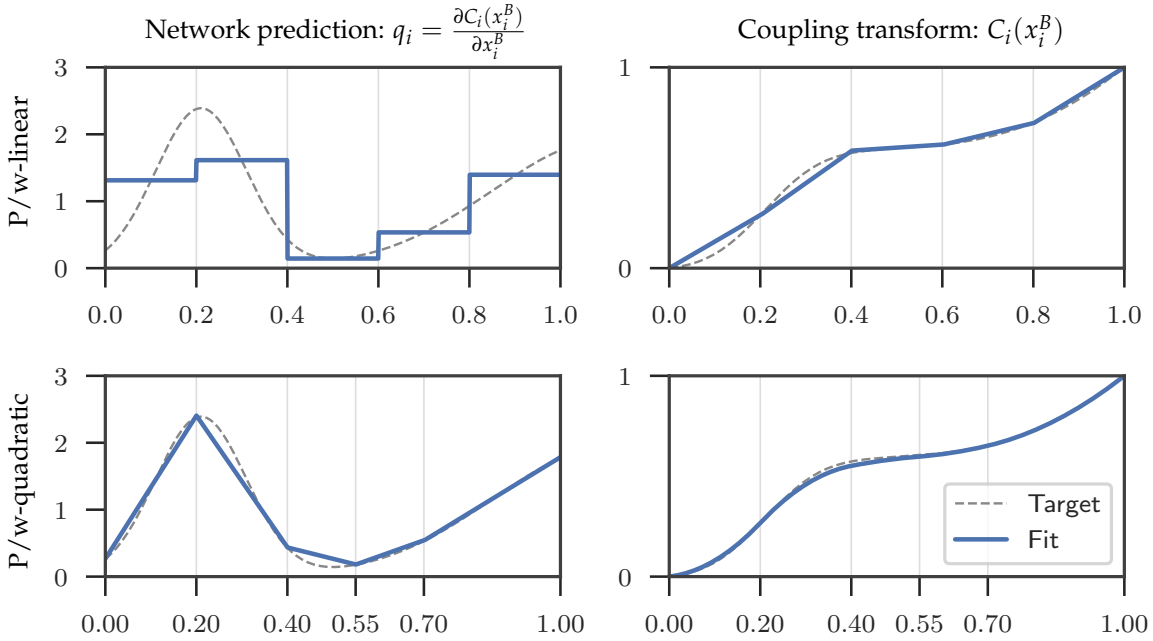


Figure 8.2: *Predicted probability density functions (PDFs, left) and corresponding cumulative distribution functions (CDFs, right) with $K = 5$ bins fitted to a target distribution (dashed). The top row illustrates a piecewise-linear CDF that only supports fixed bin sizes and the bottom row a piecewise-quadratic CDF with learned bin sizes.*

Operating on unit intervals allows interpreting the warping function C_i as a cumulative distribution function (CDF). To produce each C_i, we instrument the neural network to output the corresponding unnormalized probability density q_i, and construct C_i by integration; see Figure 8.2 for an illustration.

In order to further improve performance, we propose to encode the inputs to the neural network using *one-blob encoding*, which we discuss in Section 8.4.3.

8.4.1 Piecewise-Linear Coupling Transform

Motivated by their simplicity, we begin by investigating the simplest continuous piecewise-polynomial coupling transforms: piecewise-linear ones. Recall that we partition the D-dimensional input vector in two disjoint groups, A and B, such that $x = (x^A, x^B)$. We divide the unit dimensions in partition B into K bins of equal width $w = K^{-1}$. To define all $|B|$ transforms at once, we instrument the network $m(x^A)$ to predict a $|B| \times K$ matrix, denoted $\widehat{Q}$. Each i-th row of $\widehat{Q}$ defines the unnormalized probability mass function (PMF) of the warp in the i-th dimension of x^B; we normalize the rows using the softmax function Σ and denote the normalized matrix Q; $Q_i = \Sigma(\widehat{Q}_i)$.

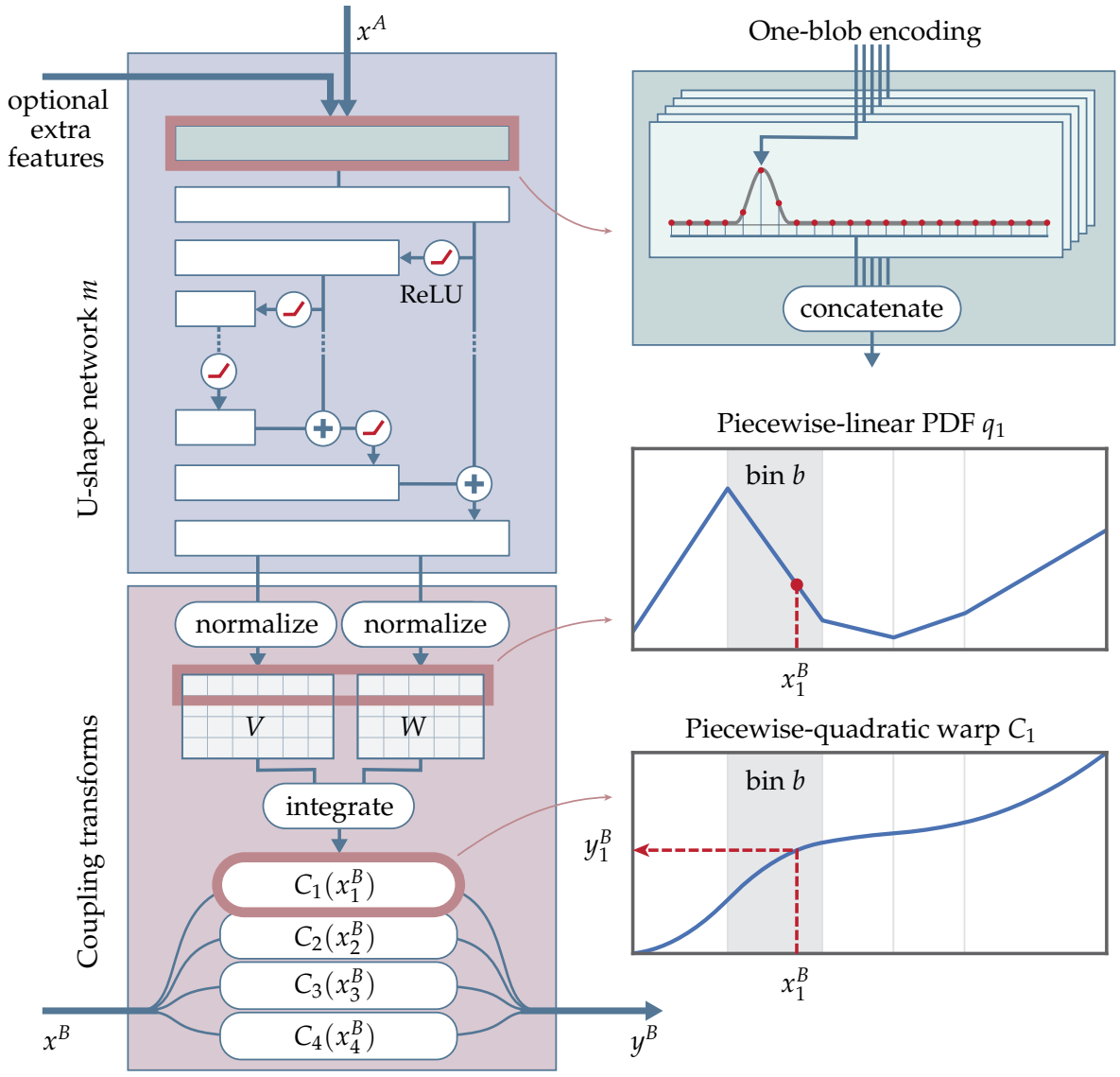


Figure 8.3: *Our coupling layer with a piecewise-quadratic transform for $|B| = 4$. Signals in partition A (and additional features) are encoded using one-blob encoding and fed into a U-shape neural network m with fully connected layers. The outputs of m are normalized yielding matrices V and W that define warping PDFs. The PDFs are integrated analytically to obtain piecewise-quadratic coupling transforms; one for warping each dimension of x^B.*

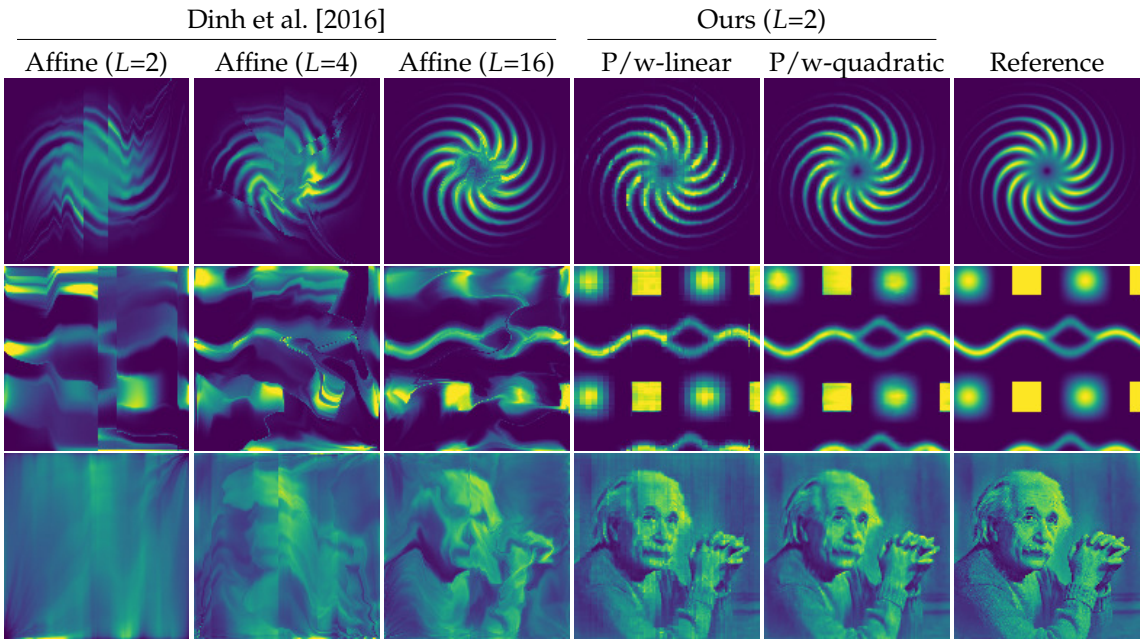


Figure 8.4: *Our 32-bin piecewise-linear (4-th column) and 32-bin piecewise-quadratic (5-th column) coupling layers achieve superior performance compared to affine (multiply-add) coupling layers [Dinh et al. 2016] on three 2-dimensional regression problem. The false-colored distributions were obtained by optimizing KL divergence with uniformly drawn i.i.d. samples (weighted by the reference value) over the 2D image domain.*

The PDF in i-th dimension is then defined as $q_i(x_i^B) = Q_{ib}/w$, where $b = \lfloor Kx_i^B \rfloor$ is the bin that contains the scalar value x_i^B. We integrate the PDF to obtain our invertible piecewise-linear warp C_i:

$$C_i(x_i^B) = \int_0^{x_i^B} q_i(t)\,\mathrm{d}t = \alpha Q_{ib} + \sum_{k=1}^{b-1} Q_{ik}\,, \tag{8.10}$$

where $\alpha = Kx_i^B - \lfloor Kx_i^B \rfloor$ represents the relative position of x_i^B in b.

In order to evaluate the change of density resulting from the coupling layer, we need to compute the determinant of its Jacobian matrix; see Equation 2.45. Since $C(x^B)$ is separable by definition, its Jacobian matrix is diagonal and the determinant is equal to the product of the diagonal terms. These can be computed using Q:

$$\det\left(\frac{\partial C(x^B;Q)}{\partial (x^B)^T}\right) = \prod_{i=1}^{|B|} q_i(x_i^B) = \prod_{i=1}^{|B|} \frac{Q_{ib}}{w}\,, \tag{8.11}$$

where b again denotes the bin containing the value in the i-th dimension. To reduce the number of bins K required for a good fit we would like the network

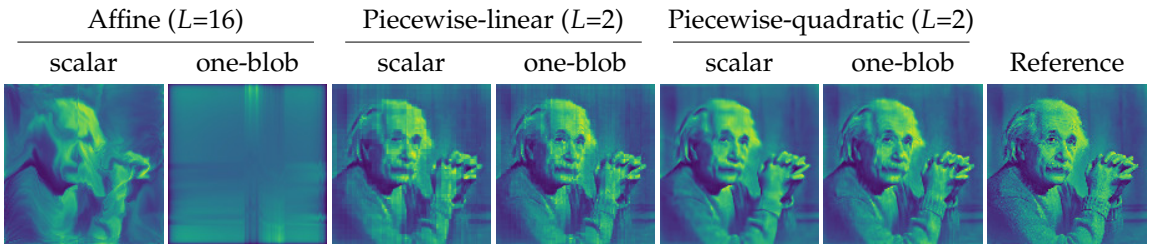


Figure 8.5: *Results with scalar- and our one-blob input encoding. The experimental setup is the same as in Figure 8.4. While the affine coupling transforms fail to converge with one-blob encoded inputs, the distributions learned by the piecewise-polynomial coupling functions become sharper and more accurate.*

to also predict bin widths. These can unfortunately *not* easily be optimized with gradient descent in the piecewise-*linear* case, which we demonstrate in Section D.2. To address this, and to improve accuracy, we propose piecewise-*quadratic* coupling transforms.

8.4.2 Piecewise-Quadratic Coupling Transform

Piecewise-quadratic coupling transforms admit a piecewise-linear PDF, which we model using $K+1$ vertices; see Figure 8.2, bottom left. We store their vertical coordinates (for all dimensions in B) in $|B| \times (K+1)$ matrix V, and horizontal differences between neighboring vertices (bin widths) in $|B| \times K$ matrix W.

The network m outputs unnormalized matrices $\widehat{W}$ and $\widehat{V}$. We again normalize the matrices using the standard softmax $W_i = \Sigma(\widehat{W}_i)$, and an adjusted version in the case of V:

$$V_{i,j} = \frac{\exp\left(\widehat{V}_{i,j}\right)}{\sum_{k=1}^{K} \frac{\exp\left(\widehat{V}_{i,k}\right)+\exp\left(\widehat{V}_{i,k+1}\right)}{2} W_{i,k}}\,, \tag{8.12}$$

where the denominator ensures that V_i represents a valid PDF.

The PDF in dimension i is defined as

$$q_i(x_i^B) = \text{lerp}(V_{ib}, V_{ib+1}, \alpha)\,, \tag{8.13}$$

where $\alpha = (x_i^B - \sum_{k=1}^{b-1} W_{ik})/W_{ib}$ represents the relative position of scalar x_i^B in bin b that contains it, i.e. $\sum_{k=1}^{b-1} W_{ik} \leq x_i^B < \sum_{k=1}^{b} W_{ik}$.

The invertible coupling transform is obtained by integration:

$$C_i(x_i^B) = \frac{\alpha^2}{2}(V_{ib+1} - V_{ib})W_{ib} + \alpha V_{ib} W_{ib} + \sum_{k=1}^{b-1} \frac{V_{ik} + V_{ik+1}}{2} W_{ik}\,. \tag{8.14}$$

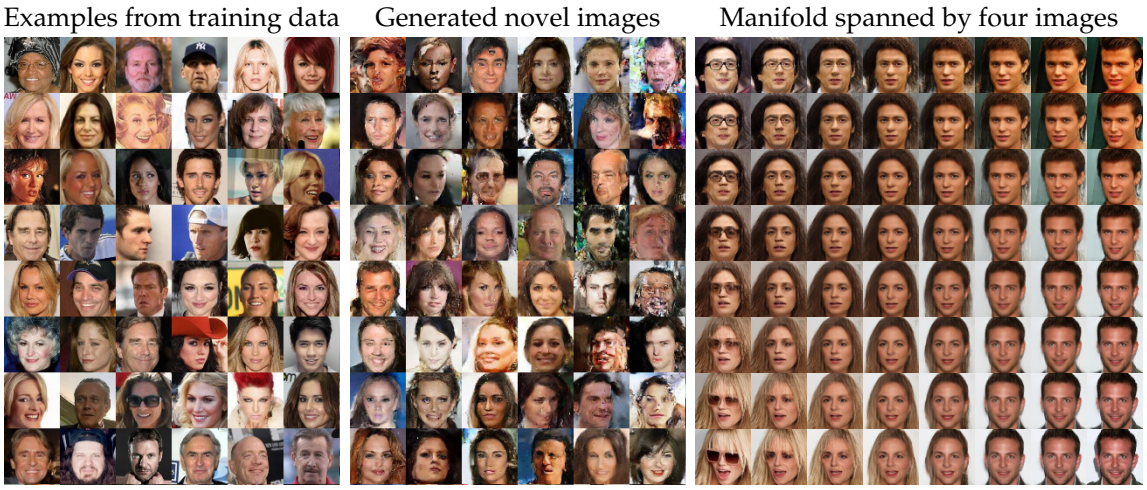


Figure 8.6: *Generative modeling of facial photographs using the architecture of Dinh et al. [2016] with our piecewise-quadratic coupling transform. We show training examples (left), faces generated by our trained model (middle), and a manifold of faces spanned by linear interpolation of 4 training examples in latent space (right; training examples are in the corners).*

Note that inverting $C_i(x_i^B)$ involves solving the root of the quadratic term, which can be done efficiently and robustly.

Computing the determinant of the Jacobian matrix follows the same logic as in the piecewise-linear case, with the only difference being that we must now interpolate the entries of V in order to obtain the PDF value at a specific location (cf. Equation 8.13).

8.4.3 One-Blob Encoding

An important consideration is the encoding of the inputs to the network. We propose to use the *one-blob* encoding—a generalization of the *one-hot* encoding—where a kernel is used to activate multiple adjacent entries instead of a single one. Assume a scalar $s \in [0, 1]$ and a quantization of the unit interval into k bins (we use $k = 32$). The one-blob encoding amounts to placing a kernel (we use a Gaussian with $\sigma = 1/k$) at s and discretizing it into the bins. With the proposed architecture of the neural network (placement of ReLUs in particular, see Figure 8.3), the one-blob encoding effectively shuts down certain parts of the linear path of the network, allowing it to specialize the model on various sub-domains of the input.

In contrast to one-hot encoding, where the quantization causes a loss of information if applied to continuous variables, the one-blob encoding is lossless; it captures the exact position of s.

8.4.4 Analysis

We compare the proposed piecewise-polynomial coupling transforms to multiply-add affine transforms Dinh et al. [2016] on a 2D regression problem in Figure 8.4. To produce columns 1–5, we sample the 2D domain using *uniform* i.i.d. samples (16 384 samples per training step), evaluate the reference function (column 6) at each sample, and optimize the neural networks that control the coupling transforms using KL divergence described in Section 8.5.1. We also ran the same experiment with equally weighted i.i.d. samples drawn *proportional* to the reference function—i.e. in a density-estimation setting—producing near-identical results (not shown). Every per-layer network has a U-net (see Figure 8.3) with 8 fully connected layers, where the outermost layers contain 256 neurons and the number of neurons is halved at every nesting level. We use 2 additional layers to adapt the input and output dimensionalities to and from 256, respectively. The networks only differ in their output layer to produce the desired parameters of their respective coupling transform (s and t, $\widehat{Q}$, or $\widehat{W}$ and $\widehat{V}$).

We use adaptive bin sizes only in the piecewise-quadratic coupling transforms because gradient descent can not easily be applied to optimize the piecewise-linear case, which we demonstrate in Section D.2.

When using $L = 2$ coupling layers—i.e. 2×10 fully connected layers—the piecewise-polynomial coupling layers consistently perform better thanks to their significantly larger modeling power, and outperform even large numbers (e.g. $L = 16$) of multiply-add coupling layers, amounting to 16×10 fully connected layers.

Figure 8.5 demonstrates the benefits of the one-blob encoding when combined with our piecewise coupling transforms. While the encoding helps our coupling transforms to learn sharp, non-linear functions more easily, it also causes the multiply-add transforms of Dinh et al. [2016] to produce excessive high frequencies that inhibit convergence. Therefore, in the rest of this chapter we use the one-blob encoding only with our piecewise-polynomial transforms; results with affine transforms do not utilize one-blob encoded inputs.

We tested the piecewise-quadratic coupling layers also on a high-dimensional density-estimation problem: learning the manifold of a specific class of natural images. We used the CelebFaces Attributes dataset [Liu et al. 2015] and reproduced the experimental setting of Dinh et al. [2016]. Our architecture is based on the authors' publicly available implementation and differs only in the coupling layer used and the depth of the network—we use 4 recursive

subdivisions while the authors use 5, resulting in 28 versus 35 coupling layers. We chose $K = 4$ bins and *did not* use our one-blob encoding due to GPU memory constraints. Since our coupling layers operate on the $[0, 1]^D$ domain, we do not use batch normalization on the transformed data.

Figure 8.6 shows a sample of the training set, a sample of generated images, and a visualization of the manifold given by four different faces. The visual quality of our results is comparable to those obtained by Dinh and colleagues. We perform marginally better in terms of negative log-likelihood (lower means better): we yield 2.85 and 2.89 bits per dimension on training and validation data, respectively, whereas Dinh et al. [2016] reported 2.97 and 3.02 bits per dimension. We tried decreasing the number of coupling layers while increasing the number of bins within each of them, but the results became overall worse. We hypothesize that the high-dimensional problem of learning distributions of natural images benefits more from having many coupling layers rather than having fewer but expressive ones.

8.5 Monte Carlo Integration with NICE

In this section, we reduce the variance of Monte Carlo integration by extracting sampling PDFs from observations of the integrand. Denoting $q(x; \theta)$ the to-be-learned PDF for drawing samples (θ represents the trainable parameters) and $p(x)$ the ground-truth distribution of the integrand, we can rewrite the MC estimator from Equation 3.3as

$$\langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i; \theta)} = \frac{1}{N} \sum_{i=1}^{N} \frac{p(X_i)\, F}{q(X_i; \theta)}. \tag{8.15}$$

In the ideal case when $q(x; \theta) = p(x)$, the estimator returns the exact value of F. Our objective here is to leverage NICE to learn *q from data* while optimizing the neural networks in coupling layers so that the distance between p and q is minimized.

We follow the standard approach of quantifying the distance using one of the commonly used divergence metrics. While all divergence metrics reach their minimum if both distributions are equal, they differ in shape and therefore produce different q in practice.

In Section 8.5.1, we optimize using the popular Kullback-Leibler (KL) divergence. We further consider directly minimizing the variance of the resulting MC estimator in Section 8.5.2 and demonstrate that it is equivalent to minimizing the χ^2 divergence.

8.5.1 Minimizing Kullback-Leibler Divergence

Most generative models based on deep neural networks do not allow evaluating the likelihood $q(x;\theta)$ of data points x exactly and/or efficiently. In contrast, our work is based on bijective mappings with tractable Jacobian determinants that easily permit such evaluations. In the following, we show that minimizing the KL divergence with gradient descent amounts to directly maximizing a weighted log likelihood.

The KL divergence between $p(x)$ and the learned $q(x;\theta)$ reads

$$\begin{aligned} D_{\mathrm{KL}}(p \,\|\, q;\theta) &= \int_\Omega p(x) \log \frac{p(x)}{q(x;\theta)} \,\mathrm{d}x \\ &= \int_\Omega p(x) \log p(x) \,\mathrm{d}x \underbrace{- \int_\Omega p(x) \log q(x;\theta) \,\mathrm{d}x}_{\text{Cross entropy}} . \end{aligned} \tag{8.16}$$

To minimize D_{KL} with gradient descent, we need its gradient with respect to the trainable parameters θ. These appear only in the cross-entropy term, hence

$$\nabla_\theta D_{\mathrm{KL}}(p \,\|\, q;\theta) = -\nabla_\theta \int_\Omega p(x) \log q(x;\theta) \,\mathrm{d}x \tag{8.17}$$

$$= \mathbb{E}\left[-\frac{p(X)}{q(X;\theta)} \nabla_\theta \log q(X;\theta) \right], \tag{8.18}$$

where the expectation is over $X \sim q(x;\theta)$, i.e. the samples are drawn from the learned generative model[3]. In most integration problems, $p(x)$ is only accessible in an unnormalized form through $f(x)$: $p(x) = f(x)/F$. Since F is unknown—this is what we are trying to estimate in the first place—the gradient can be estimated only up to the global scale factor F. This is not an issue since common gradient-descent-based optimization techniques such as Adam [Kingma and Ba 2014] scale the step size by the reciprocal square root of the gradient variance, cancelling F. Furthermore, if $f(x)$ can only be estimated via Monte Carlo, the gradient is still correct due to the linearity of expectations. Equation 8.18 therefore shows that minimizing the KL divergence via gradient descent is equivalent to minimizing the negative log likelihood weighted by MC estimates of F.

[3] If samples could be drawn directly from the ground-truth distribution—as is common in computer vision problems—the stochastic gradient would simplify to that of just the log likelihood. We discuss a generalization of log-likelihood maximization.

8.5.2 Minimizing Variance via χ^2 Divergence

The most attractive quantity to minimize in the context of (unbiased) Monte Carlo integration is the variance of the estimator. Inspired by previous works that strive to directly minimize variance [Pantaleoni and Heitz 2017; Vévoda et al. 2018], we demonstrate how this can be achieved for the MC estimator $p(X)/q(X;\theta)$, with $X \sim q(x;\theta)$, via gradient descent. We begin with the variance definition and simplify:

$$\begin{aligned}\mathbb{V}\left[\frac{p(X)}{q(X;\theta)}\right] &= \mathbb{E}\left[\frac{p(X)^2}{q(X;\theta)^2}\right] - \mathbb{E}\left[\frac{p(X)}{q(X;\theta)}\right]^2 \\ &= \int_\Omega \frac{p(x)^2}{q(x;\theta)}\,\mathrm{d}x - \underbrace{\left(\int_\Omega p(x)\,\mathrm{d}x\right)^2}_{1}.\end{aligned} \tag{8.19}$$

The stochastic gradient of the variance for gradient descent is then

$$\begin{aligned}\nabla_\theta \mathbb{V}\left[\frac{p(X)}{q(X;\theta)}\right] &= \nabla_\theta \int_\Omega \frac{p(x)^2}{q(x;\theta)}\,\mathrm{d}x \\ &= \int_\Omega p(x)^2\,\nabla_\theta \frac{1}{q(x;\theta)}\,\mathrm{d}x \\ &= \int_\Omega -\frac{p(x)^2}{q(x;\theta)}\,\nabla_\theta \log q(x;\theta)\,\mathrm{d}x \\ &= \mathbb{E}\left[-\left(\frac{p(X)}{q(X;\theta)}\right)^2 \nabla_\theta \log q(X;\theta)\right].\end{aligned} \tag{8.20}$$

Relation to the Pearson χ^2 divergence Upon close inspection it turns out the variance objective (Equation 8.19) is equivalent to the Pearson χ^2 divergence $D_{\chi^2}(p \,\|\, q;\theta)$:

$$\begin{aligned}D_{\chi^2}(p \,\|\, q;\theta) &= \int_\Omega \frac{(p(x) - q(x;\theta))^2}{q(x;\theta)}\,\mathrm{d}x \\ &= \int_\Omega \frac{p(x)^2}{q(x;\theta)}\,\mathrm{d}x - \underbrace{\left(2\int_\Omega p(x)\,\mathrm{d}x - \int_\Omega q(x;\theta)\,\mathrm{d}x\right)}_{1}.\end{aligned} \tag{8.21}$$

As such, minimizing the variance of a Monte Carlo estimator amounts to minimizing the Pearson χ^2 divergence between the ground-truth and the learned distributions.

Connection between the χ^2 and KL divergences Notably, the gradients of the KL divergence and the χ^2 divergence differ only in the weight applied to the log likelihood. In $\nabla_\theta D_{\mathrm{KL}}$ the log likelihood is weighted by the MC weight, whereas when optimizing $\nabla_\theta D_{\chi^2}$, the log likelihood is weighted by the *squared* MC weight. This illustrates the difference between the two loss functions: the χ^2 divergence penalizes large discrepancies stronger, specifically, low values of q in regions of large density p. As such, it tends to produce more conservative q than D_{KL}, which we observe in Section 8.6 as fewer outliers at the cost of slightly worse average performance.

8.6 Neural Path Sampling and Path Guiding

In this section, we take NICE (Section 8.3) with piecewise-polynomial warps (Section 8.4) and apply it to sequential MC integration of light transport using the methodology described in Section 8.5.

Our goal is to reduce estimation variance by "guiding" the construction of light paths using sampling densities learned on the fly by leveraging the on-line-learning framework with iterations of geometrically growing sample counts developed in Chapter 7. We explore two different settings: a global setting that leverages the path-integral formulation of light transport and employs high-dimensional sampling in the primary sample space (PSS; see Section 4.5.5) to build complete light-path samples (Section 8.6.1), and a local setting, natural to the rendering equation, where the integration spans a 2D (hemi-)spherical domain and the path is built incrementally (Section 8.6.2).

8.6.1 Primary-Sample-Space Path Sampling

Operating in PSS (see Section 4.5.5) has a number of compelling advantages. The sampling routine has to be evaluated only once per path, instead of once per path vertex, and the generic nature of PSS coordinates enables treating the path construction as a black box. Importance sampling of paths can thus be applied to any single path-tracing technique, and, with some effort, also to multiple strategies Lafortune and Willems [1995]; Veach and Guibas [1994]; Kelemen et al. [2002]; Hachisuka et al. [2014]; Guo et al. [2018]. Lastly, the sampling routine directly benefits from existing importance-sampling techniques in the underlying path-tracing algorithm since those make the path-contribution function smoother in PSS and thus easier to learn.

Methodology Given that NICE scales well to high-dimensional problems, applying it in PSS is straightforward. We split the dimensions of $\mathbb{U}$ into two equally-sized groups A and B, where A contains the even dimensions and B contains the odd dimensions. One group serves as the input of the neural network (each dimension is processed using the one-blob encoding) while the other group is being warped; their roles are swapped in the next coupling layer. To infer the parameters θ of the networks, we minimize one of the losses from Section 8.5 against $p(\overline{\mathbf{x}}) = L_\mathrm{i}(\mathbf{x}_1 \leftarrow \mathbf{x}_2) T(\overline{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \leftarrow \mathbf{x}_k)\, F^{-1}$, ignoring the unknown normalization factor, i.e. assuming $F = 1$.

In order to obtain a path sample $\overline{\mathbf{x}}$, we generate a random vector z, warp it using the reversed inverted coupling layers, and apply the path-construction technique: $\overline{\mathbf{x}} = \rho\Big(h_1^{-1}\Big(\cdots h_L^{-1}(z)\Big)\Big)$; please refer back to Equations 8.5 and 8.6 for details on the inverses. In this project, all our experiments were conducted with ρ corresponding to a unidirectional path tracer, but in principle other path construction techniques could be used. Their advantages and disadvantages in the primary-sample-space setting remain to be investigated.

Before we analyze the performance of primary-sample-space path sampling in Section 8.6.4, we discuss *neural path guiding*, which applies NIS at each vertex of the path—essentially replacing the SD-tree in PPG from Chapter 7—and typically yields higher performance.

8.6.2 Path Guiding

The concept of *path guiding* by leveraging observations of the directional integrand to construct sampling densities and previous work on the topic has already been discussed in detail in Section 7.2. Therefore, in this section, we will focus on the main differences of applying NIS to path guiding, which we dub *neural path guiding* (NPG), as opposed to using the SD-tree from Chapter 7 or other previous work [Vorba et al. 2014].

All previous path-guiding approaches rely on carefully chosen data structures (e.g. BVHs, kD-trees) in combination with relatively simple PDF models (e.g. histograms, quad-trees, gaussian mixture models), which are populated in a data-driven manner either in a pre-pass or on line during rendering. Our goal is also to learn accurate local sampling densities, but we utilize NICE to represent and sample from $q(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o})$.

Methodology. We use a single instance of NICE, which is trained and sampled from in an interleaved manner: drawn samples are immediately used for

training, and training results are immediately used for further sampling. In the most general setting, we consider learning $q(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o})$ that is proportional to the product of *all* terms in the integrand. Since the integration domain is only 2D, partitions A and B in all coupling layers contain only one dimension each—one of the two cylindrical coordinates that we use to parameterize the sphere of directions.

To produce the parameters of the first piecewise-polynomial coupling function, the neural network m takes the cylindrical coordinate from A, the position $\mathbf{x}$ and direction ω_o that condition the density, and additional information that may improve inference; we also input the normal of the intersected shape at $\mathbf{x}$ to aid the network in learning distributions that correlate with the local shading frame. The subsequent piecewise-polynomial coupling functions are parameterized by their corresponding neural networks analogously as described in Section 8.3.3.

We one-blob encode all of the network inputs as described in Section 8.4.3 with $k = 32$. In the case of $\mathbf{x}$, we normalize it by the scene bounding box, encode each coordinate independently, and concatenate the results into a single array of $3 \times k$ values. We proceed analogously with directions, which we parameterize using world-space cylindrical coordinates: we transform each coordinate to $[0, 1]$ interval, encode it, and append to the array. The improved performance enabled by our proposed one-blob encoding is reported in Table 8.1.

At any given point during rendering, a sample is generated by drawing a random pair $u \in [0, 1]^2$, passing it through the inverted coupling layers in reverse order, $h_1^{-1}(\cdots h_L^{-1}(u))$, and transforming to the range of cylindrical coordinates to obtain ω_i.

MIS-aware Optimization. In order to optimize the networks, we use Adam with one of the loss functions from Section 8.5, but with an important, problem-specific alteration. To sample ω_i, most current renderers apply multiple importance sampling (MIS) [Veach and Guibas 1995] to combine multiple sampling densities (e.g. to estimate direction illumination or importance sample the BSDF). When learning the product, we take this into account by optimizing the networks with respect to the *final* (MIS) PDF instead of the density learned using NICE. If certain parts of the product are already covered well by existing densities, the networks will therefore be optimized to only handle the remaining problematic case.

We therefore optimize $D(p \,\|\, q')$ where $q' = wq + (1 - w)p_{f_\mathrm{s}}$ and

$p(\omega_\mathrm{i}|\mathbf{x},\omega_\mathrm{o}) = L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\to\omega_\mathrm{o})|\cos\gamma| F^{-1}$ (we again ignore the normalization constant F) and p_{f_s} is the PDF for sampling the BSDF.

We use an additional network $\hat{m}$ to learn optimal MIS weights $w = l(\hat{m}(\mathbf{x},\omega_\mathrm{o}))$ and optimize it jointly with the other networks; all networks use the same architecture and l is the logistic function. To prevent overfitting to local optima with degenerate MIS weights, we use loss funtion $\beta(\tau)D(p\,\|\,q) + \big(1-\beta(\tau)\big)D(p\,\|\,q')$ where $\tau \in [0,1]$ is the fraction of exhausted render budget (either time or sample count) and $\beta(\tau) = 1/2 \cdot (1/3)^{5\tau}$.

BSDFs that are a mixture of delta and smooth functions—such as plastic—require a small amount of special handling. While our stochastic gradient in Section 8.5 in theory is well behaved with delta functions, they need to be treated as finite quantities in practice due to the limitations of floating point numbers. When the path tracer samples delta components, continuous densities therefore need to be set to 0 and optimization of our coupling functions disabled (by setting their loss to 0), effectively only optimizing for MIS weights.

Discussion. Our approach to sampling the full product of incident radiance, the BSDF, and foreshortening $L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i}) f_\mathrm{s}(\mathbf{x},\omega_\mathrm{i}\to\omega_\mathrm{o})|\cos\gamma|$ has three distinct advantages. First, it is agnostic to the number of dimensions that the 2D domain is conditioned on. This allows for high-dimensional conditionals without sophisticated data structures. One can simply input extra information into the neural networks and let them learn which dimensions are useful in which situations. While we only pass in the surface normal, the networks could be supplied with additional information—e.g. textured BSDF parameters—to further improve the performance in cases where the product correlates with such information. In that sense, our approach is more automatic then previous works.

The second advantage is that our method does not require any precomputation, such as fitting of (scene-dependent) materials into a mixture of gaussians [Herholz et al. 2016, 2018]. While a user still needs to specify the hyperparameters as is also required by most other approaches, we found our configuration of hyperparameters to work well across all tested scenes. Note, however, that our lack of explicit factorization can also be detrimental in situations where the individual factors are much simpler functions to learn than their product and said product can be efficiently sampled from.

Lastly, our approach offers trivial persistence across renders. A set of networks trained on one camera view can be reused from a different view

or within a slightly modified scene; see Section 8.6.4. Unlike previous approaches, where the learned data structure requires explicit support of adaptation to new scenes, neural networks can be adapted by the same optimization procedure that was used in initial training. Applying our approach to animations could thus yield sub-linear training cost by amortizing it over multiple frames.

8.6.3 Experimental Setup

We implemented our technique in Tensorflow Abadi et al. [2015] and integrated it with the Mitsuba renderer Jakob [2010]. Before we start rendering, we initialize the trainable parameters of our networks using Xavier initialization Glorot and Bengio [2010]. While rendering the image, we optimize them using Adam Kingma and Ba [2014]. Our rendering procedure is implemented as a hybrid CPU/GPU algorithm, tracing rays in large batches on the CPU while two GPUs perform all neural-network-related tasks. One GPU is responsible for computing MIS weights and evaluating and sampling from q, while the other trains the networks using Monte Carlo estimates from completed paths. Both GPUs use minibatch sizes of 100 000 samples. Communication between the CPU and GPUs happens via asynchronous buffers to aid parallelization. MIS weight computation and q-evaluation and -sampling communicate via asynchronous *queues* that are processed as fast as possible. Our training buffer is configured to always contain the latest 2 000 000 samples of which minibatches are *randomly* selected for optimization. This procedure decorrelates samples that are nearby in the image plane.

In order to obtain the final image with N samples, we perform $M = \lfloor \log_2(N+1) \rfloor$ iterations with power-of-two sample counts $2^i; i \in \{0, \dots, M\}$, as proposed in Chapter 7. In contrast to Chapter 7, however, we do not reset the learned distributions at every power-of-two iteration and keep training the same set of networks from start to finish. At any point during rendering the latest samples are used for training the neural networks and the latest training results are used for drawing samples. Furthermore, instead of discarding the samples of earlier iterations, we weight the images produced within each iteration by their reciprocal mean pixel variance, which we estimate on the fly. This strategy was discussed in Section 7.6.2 as an alternative to automatically determining a training-rendering budget split and we obtained the best results with it. We apply the same weighting scheme to PPG when comparing against it in this chapter to ensure a fair comparison.

All results in this chapter were produced on a workstation with two Intel Xeon E5-2680v3 CPUs (24 cores; 48 threads) and two NVIDIA Titan Xp

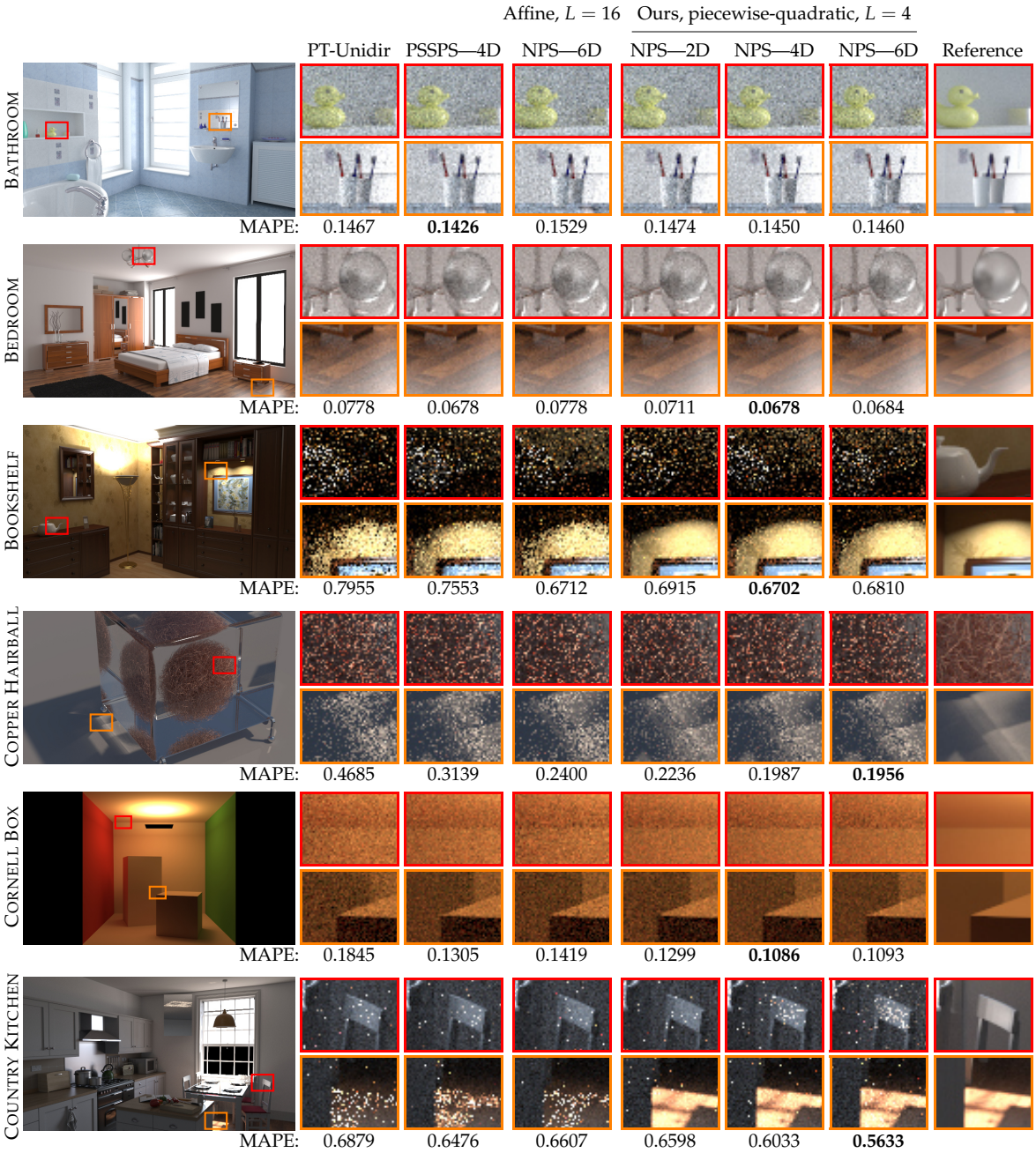


Figure 8.7: *Neural path sampling in primary sample space 1. We compare a standard unidirectional path tracer (PT-Unidir), the method by Guo et al. [2018] (PSSPG), neural path sampling using $L = 16$ multiply-add coupling layers Dinh et al. [2016], and $L = 4$ of our proposed piecewise-quadratic coupling layers, both optimized using the KL divergence. We experimented with sampling the 1, 2, or 3 first non-specular bounces (NPS–2D, NPS–4D and NPS–6D). Overall, our technique performs best in terms of* mean absolute percentage error *(MAPE) in this experiment, but only rarely offers improvement beyond the 4D case.*

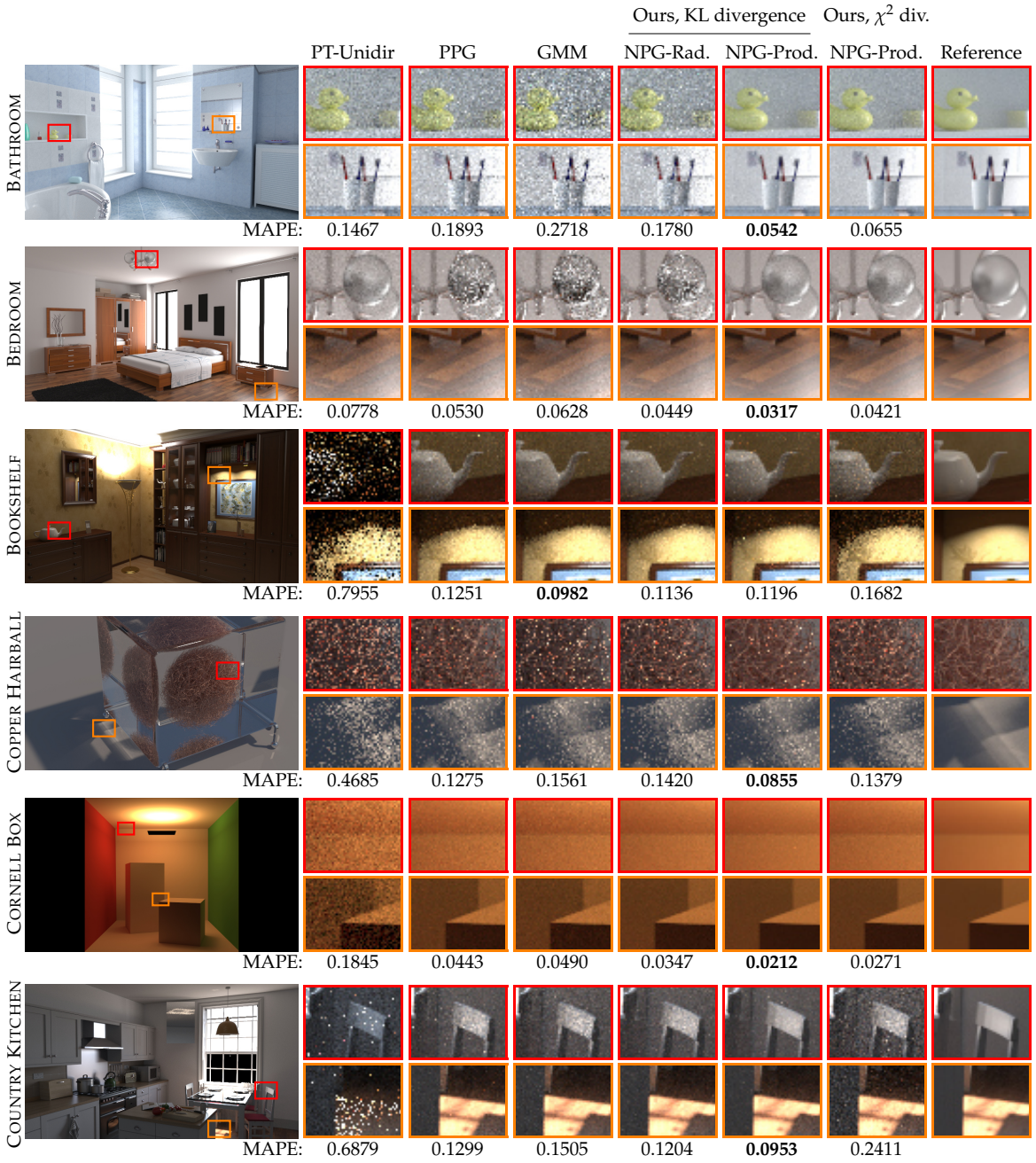


Figure 8.8: *Neural path guiding 1. We compare a uni-directional path tracer (PT-Unidir), our PPG algorithm, the gaussian mixture model (GMM) of Vorba et al. [2014], and variants of our framework with $L = 4$ coupling layers sampling the incident radiance alone (NPG-Radiance) or the whole integrand (NPG-Product), when optimizing either the KL and χ^2 divergences. Overall, sampling the whole integrand with the KL divergence yields the most robust results.*

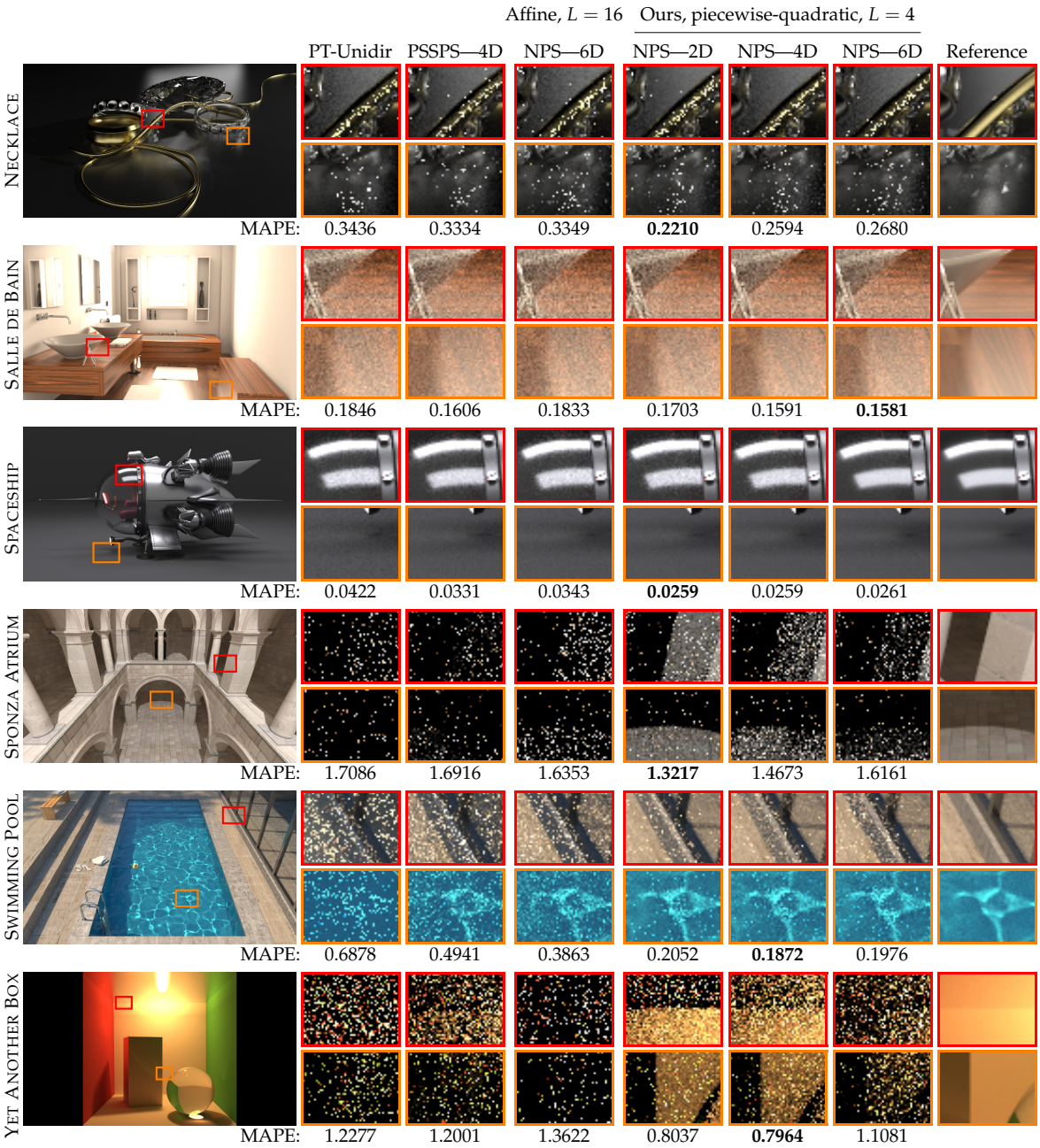


Figure 8.9: *Neural path sampling in primary sample space 2. We compare a standard unidirectional path tracer (PT-Unidir), the method by Guo et al. [2018] (PSSPG), neural path sampling using $L = 16$ multiply-add coupling layers Dinh et al. [2016], and $L = 4$ of our proposed piecewise-quadratic coupling layers, both optimized using the KL divergence. We experimented with sampling the 1, 2, or 3 first non-specular bounces (NPS–2D, NPS–4D and NPS–6D). Overall, our technique performs best in terms of* mean absolute percentage error *(MAPE) in this experiment, but only offers improvement beyond the 4D case if paths stay coherent, e.g. in the top crop of the* Spaceship *scene.*

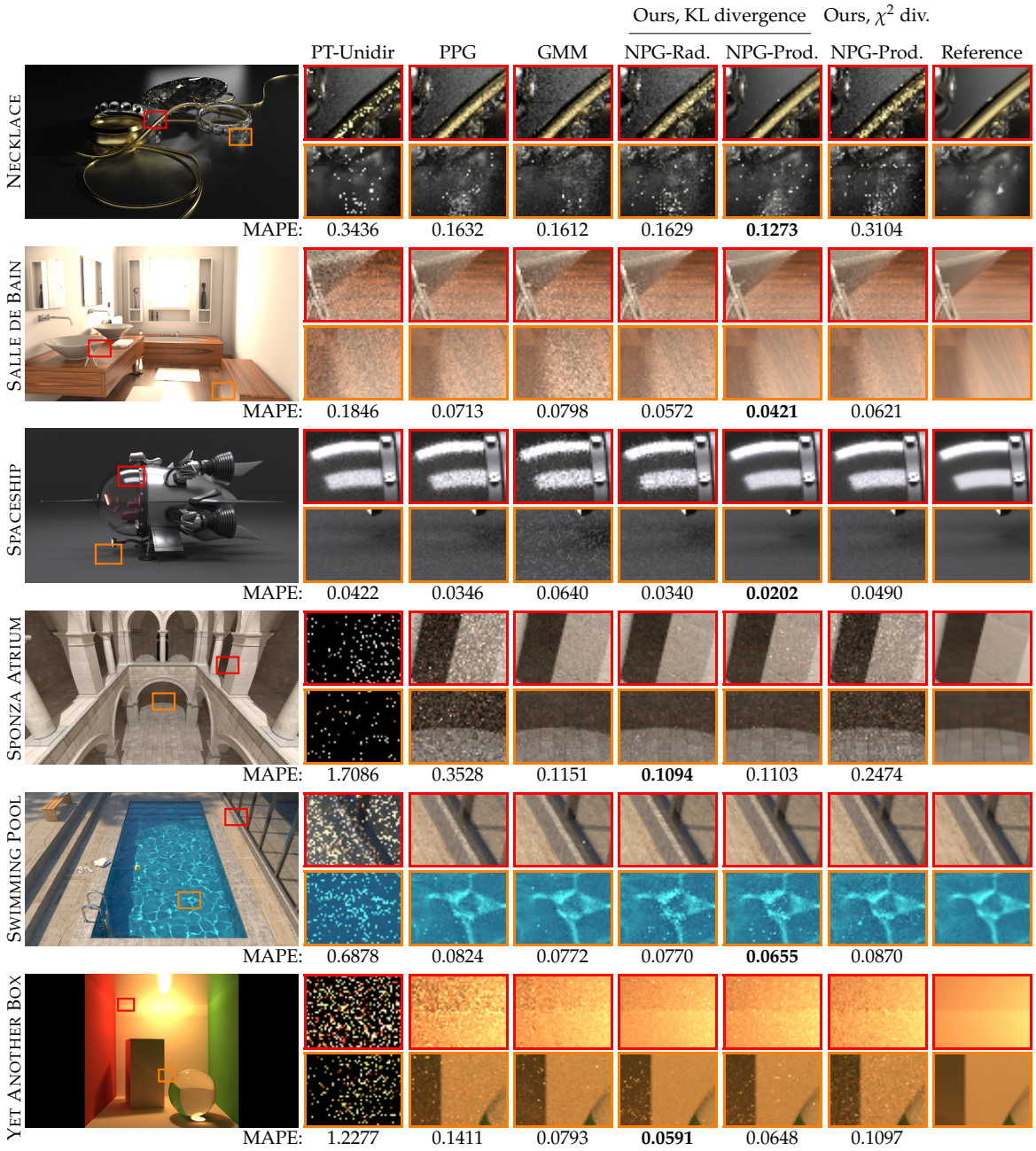


Figure 8.10: *Neural path guiding 2. We compare a uni-directional path tracer (PT-Unidir), our PPG algorithm, the gaussian mixture model (GMM) of Vorba et al. [2014], and variants of our framework with $L = 4$ coupling layers sampling the incident radiance alone (NPG-Radiance) or the whole integrand (NPG-Product), when optimizing either the KL and χ^2 divergences. Overall, sampling the whole integrand with the KL divergence yields the most robust results. Note how optimizing the χ^2 divergence tends to produce higher variance overall, but fewer outliers, in particular in the* Swimming Pool *scene.*

Table 8.1: *Mean average percentage error (MAPE) and render times of various importance-sampling approaches. At equal sampling rates—we report the number of samples in each scene as mega samples (MS)—our neural path guiding performs on par or better than our PPG algorithm and the bidirectionally trained gaussian mixture model (GMM) of Vorba et al. [2014], but incurs a large computational overhead. Since the GMMs are trained in a pre-pass, we report both their training and rendering times. Please note, that the provided implementation of the GMM training does not scale well beyond 8 threads. Our neural path sampling (NPS) likewise compares favorably against the method by Guo et al. [2018] (PSSPG). Using one-blob encoding significantly improves the quality of our results; see Figure 8.11 for a histogram visualization of these metrics.*

		PT-Unidir		PPG		GMM		PSSPS		Ours, KL divergence								Ours, χ^2 div.	
										NPS		NPG-Rad.		NPG-Product		NPG-Product		NPG-Product	
										one-blob		one-blob		scalar		one-blob		one-blob	
Bathroom	236 MS	0.147	88s	0.189	2.3m	0.272	9.1m+ 48s	0.143	89s	0.146	3.5m	0.178	9.3m	0.071	11m	**0.054**	12m	0.066	15m
Bedroom	236 MS	0.078	75s	0.053	1.8m	0.063	34m+ 60s	0.068	77s	0.068	3.5m	0.045	6.4m	0.037	7.2m	**0.032**	7.7m	0.042	10m
Bookshelf	236 MS	0.796	74s	0.125	2.5m	**0.098**	16m+ 66s	0.755	78s	0.681	3.5m	0.114	8.1m	0.250	10m	0.120	10m	0.168	12m
Copper Hairball	472 MS	0.468	2.0m	0.128	1.8m	0.156	11m+2.0m	0.314	2.0m	0.196	6.9m	0.142	4.9m	0.092	15m	**0.086**	16m	0.138	17m
Cornell Box	268 MS	0.185	23s	0.044	82s	0.049	6.2m+ 24s	0.130	26s	0.109	4.0m	0.035	6.6m	0.027	8.5m	**0.021**	10m	0.027	9.1m
Country Kitchen	236 MS	0.688	48s	0.130	81s	0.151	13m+ 33s	0.648	49s	0.563	3.5m	0.120	5.4m	0.123	6.9m	**0.095**	7.8m	0.241	8.1m
Glossy Kitchen	236 MS	1.476	78s	0.308	87s	—		1.452	77s	1.491	3.5m	0.243	5.6m	0.810	11m	**0.136**	11m	0.251	13m
Necklace	236 MS	0.344	29s	0.163	42s	0.161	3.2m+ 15s	0.333	31s	0.268	3.5m	0.163	2.6m	0.249	10m	**0.127**	11m	0.310	10m
Salle de Bain	236 MS	0.185	51s	0.071	89s	0.080	11m+ 37s	0.161	54s	0.158	3.5m	0.057	5.4m	0.052	5.7m	**0.042**	6.1m	0.062	7.9m
Spaceship	236 MS	0.042	27s	0.035	56s	0.064	4.4m+4.6m	0.033	28s	0.026	3.5m	0.034	2.8m	0.027	3.5m	**0.020**	3.9m	0.049	4.1m
Sponza Atrium	236 MS	1.709	84s	0.353	91s	0.115	11m+ 53s	1.692	81s	1.616	3.5m	**0.109**	7.4m	0.213	9.0m	0.110	11m	0.247	11m
Staircase	236 MS	0.163	57s	0.057	81s	0.065	14m+ 41s	0.138	58s	0.130	3.5m	0.044	5.5m	0.038	5.7m	**0.033**	6.3m	0.043	7.7m
Swimming Pool	236 MS	0.688	40s	0.082	63s	0.077	29m+ 22s	0.494	41s	0.198	3.5m	0.077	3.0m	0.073	4.5m	**0.066**	5.1m	0.087	5.1m
Veach Door	236 MS	0.910	38s	0.227	70s	—		0.903	41s	0.716	3.5m	0.135	9.0m	0.135	12m	**0.099**	13m	0.137	12m
White Room	236 MS	0.102	79s	0.066	1.9m	0.076	24m+ 55s	0.090	79s	0.093	3.5m	0.058	7.0m	0.046	7.8m	**0.040**	8.3m	0.045	11m
Yet Another Box	1073 MS	1.228	1.9m	0.141	4.7m	0.079	6.0m+1.7m	1.200	2.1m	1.108	15m	**0.059**	23m	0.222	30m	0.065	36m	0.110	33m

Figure 8.11: *MAPE achieved by the bidirectionally trained gaussian mixture model by Vorba et al. [2014] (the* Glossy Kitchen *and* Veach Door *are omitted because of limitations of their implementation), the primary-sample-space method by Guo et al. [2018], and our neural importance-sampling approaches on different scenes (the order and colors of bars follows Table 8.1). The bars are normalized with respect to our PPG algorithm (see Chapter 7); a height below 1 signifies better performance. Some bars exceed outside of the displayed range; Table 8.1 provides the actual numbers. Primary-sample-space techniques generally perform worse than path-guiding approaches. The product-driven neural path guiding usually performs the best.*

GPUs.Due to the combined usage of both the CPU and the GPU, runtimes of different techniques depend strongly on the particular setup. We therefore focus on comparing the performance using *equal-sample-count* metrics that are independent of hardware. Absolute timings and an equal-time comparison of a subset of the scenes and methods are provided for completeness.

We quantify the error using the *mean absolute percentage error* (MAPE), which is defined as $\frac{1}{N}\sum_{i=1}^{N}|v_i - \hat{v}_i|/(\hat{v}_i + \epsilon)$, where $\hat{v}_i$ is the value of the i-th pixel in the ground-truth image, v_i is the value of the i-th rendered pixel, and $\epsilon = 0.01$ serves the dual objective of avoiding the singularity at $\hat{v}_i = 0$ and down-weighting close-to-black pixels.

8.6.4 Results

In order to best illustrate the benefits of different neural-importance-sampling approaches, we compare their performance when used on top of a unidirectional path tracer that uses BSDF sampling only. While none of the results utilized next-event estimation (including prior works), we recommend using it in practice for best performance. In the following, all results with our piecewise-polynomial coupling functions utilize $L = 4$ coupling layers. We use 1023 samples per pixel (spp) on all scenes except for the COPPER HAIRBALL (2047 spp) and YET ANOTHER BOX (4095 spp). Our images were rendered at low resolutions (640×360 and 512×512), which means lower spp would be required at high definition to achieve the same total sample count, leading to learned distributions of the same quality ($9\times$ fewer spp at 1920×1080). We therefore also report the total sample count as mega samples (MS) as it reflects the quality of learned distributions independent of resolution.

In Figures 8.7 and 8.9, we study primary-sample-space path sampling using our implementation of the technique by Guo et al. [2018] (PSSPG) and our *neural path sampling* (NPS) with piecewise-polynomial and affine coupling transforms. We apply the sampling to only a limited number of non-specular interactions in the beginning of each path and sample all other interactions using uniform random numbers. We experimented with three different prefix dimensionalities: 2D, 4D, and 6D, which correspond to importance sampling path prefixes of 1, 2, and 3 non-specular vertices, respectively. As shown in the figure, going beyond 4D brings typically little improvement, except for the highlights in the SPACESHIP, where even longer paths are correlated thanks to highly-glossy interactions with the glass of the cockpit[4]. This confirms

[4]Due to faster training of lower-dimensional distributions, the 2D case still has the least overall noise in the SPACESHIP scene.

the observation of Guo et al. [2018] that cases where more than two bounces are needed to connect to the light source offer minor to no improvement. We speculate that the poor performance in higher dimensions is due to the relatively weak correlation between path geometries and PSS coordinates, i.e. paths with similar PSS coordinates may have drastically different vertex positions. The correlation tends to weaken at each additional bounce (e.g. in the diffuse CORNELL BOX) unless the underlying path importance-sampling technique preserves path coherence.

In Figures 8.8 and 8.10, we analyze the performance of different path-guiding approaches, referring to ours as *neural path guiding* (NPG). We compare our work to the respective authors' implementations of *practical path guiding* (PPG) by Müller et al. [2017] and the bidirectionally trained *gaussian mixture model* (GMM) by Vorba et al. [2014] which are both learning sampling distributions that are, in contrast to ours, proportional to incident radiance only. We extended the public GMM implementation to (oriented) spherical domains. To isolate the benefits of using NICE with piecewise-quadratic coupling layers, we created a variant of our approach that learns densities proportional to just incident radiance and disregards MIS (NPG-Radiance). The *radiance-driven* neural path guiding outperforms PPG and GMM in 13 out of 16 scenes and follows closely in the others (BOOKSHELF, COPPER HAIRBALL, and SPONZA ATRIUM), making it the most robust method out of the three.

The performance of our neural approach is further increased by learning and sampling proportional to the full product and incorporating MIS into the optimization—this technique yields the best results in nearly all scenes. As seen on the COPPER HAIRBALL, our technique can learn the product even under high-frequency spatial variation by passing the surface normal as an additional input to the networks. We trained all techniques with the same number of samples as used for rendering. The SD-tree and our neural networks used between 5 MB and 10 MB, the gaussian mixture model used between 5 MB and 118 MB; the references required about 5 GB.

Table 8.1 reports the MAPE metric for a set of 16 tested scenes. We also visualize the results of all methods using bar charts in Figure 8.11; the height is normalized with respect to PPG. We exclude GMM results for the GLOSSY KITCHEN and VEACH DOOR scenes due to crashes and bias, respectively. Path sampling in PSS typically yields significantly worse results than all path-guiding approaches. Neural path guiding always benefits (sometimes significantly) from encoding the inputs with one-blob encoding as opposed to inputing raw (scalar) values.

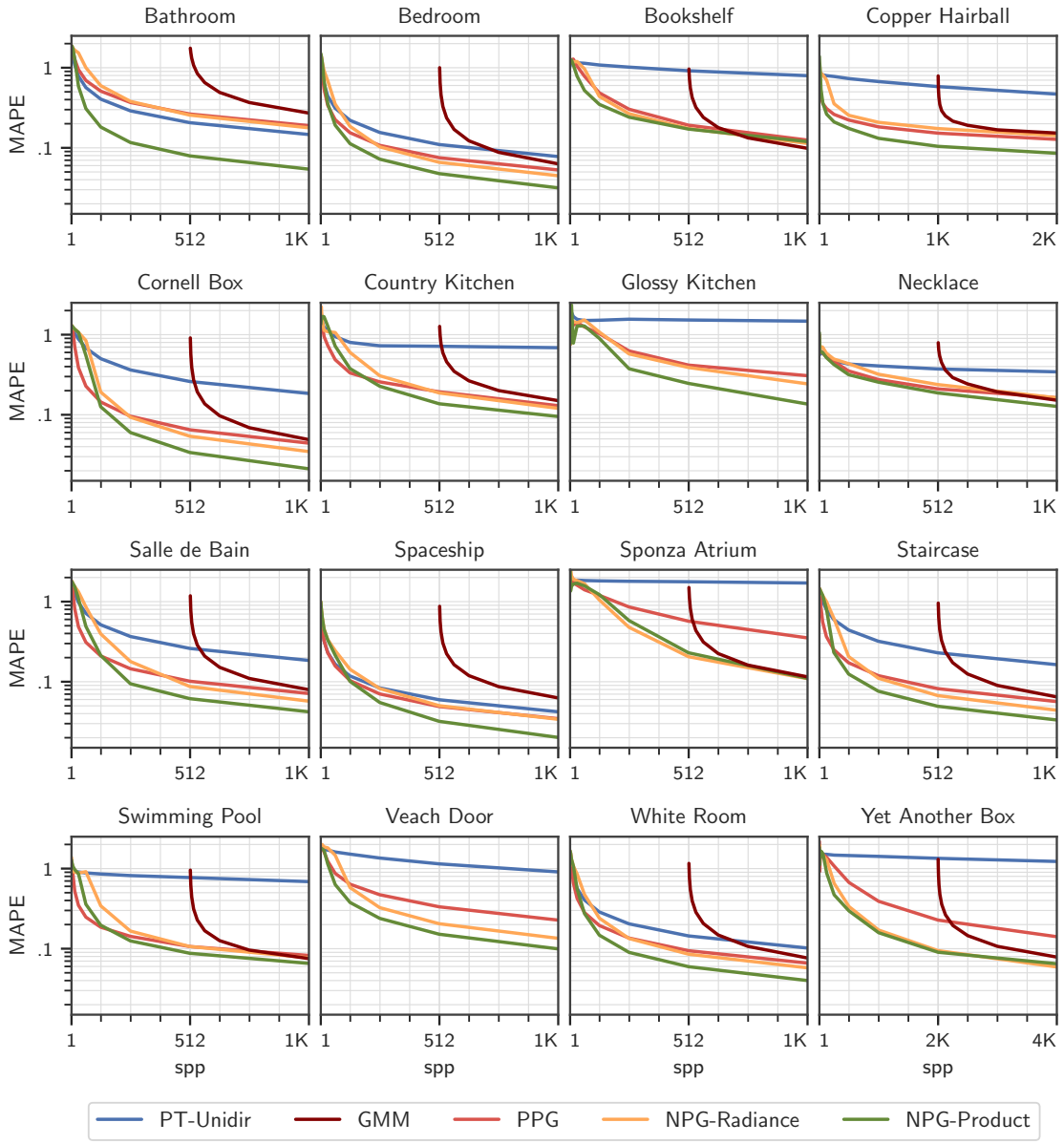


Figure 8.12: *Convergence plots of unidirectional path tracing (PT-Unidir), practical path guiding (PPG) [Müller et al. 2017], the algorithm of Vorba et al. [2014] (GMM), and our radiance- and product-based neural path guiding (NPG-Radiance and NPG-Product, respectively). We plot MAPE as a function of samples per pixel (spp) on a logarithmic scale. All guiding methods perform slightly worse than naïve path tracing initially, but overtake it rapidly on most scenes as they learn to importance sample. PPG tends to learn slightly faster than our NPG, but falls behind due to learning a worse distribution. The algorithm of Vorba et al. [2014] only begins converging at the half-way point due to being trained offline using the first half of the sample budget.*

Empirical Convergence Analysis. Convergence plots in Figure 8.12 provide further insight into the differences between unidirectional path tracing, the GMMs by Vorba et al. [2014], PPG (Chapter 7), and our radiance- and product-based neural-path-guiding algorithms. In most cases, the on-line path-guiding algorithms quickly learn a superior sampling density compared to the baseline path tracer. The GMM algorithm is trained offline and therefore is inferior in the beginning of rendering. Nonetheless, the GMMs produce competitive results after the total sample budget is exhausted. Our neural approaches produce the best results most of the time, with our product-based approach usually being superior to our radiance-based approach.

Optimizing KL vs. χ^2 Divergence. We compare variants of product-driven neural path guiding optimized using the Kullback-Leibler (KL) and χ^2 divergences during training. The squared Monte Carlo weight in the χ^2 gradient causes a large variance, making it difficult to optimize with. We remedy this problem by clipping the minibatch gradient norm to a maximum of 50. While the χ^2 divergence in theory minimizes the estimator variance most directly (see Section 8.5.2), it performs worse in practice according to all tested metrics on all test scenes (see Table 8.1 and Figures 8.8 and 8.10). A notable aspect of optimizing the χ^2 divergence is that it tends to produce results with higher variance overall, but fewer and less-extreme outliers.

Accuracy of Learned Distributions. We visualize learned radiance distributions in Figure 8.13, comparing our technique against the SD-tree of PPG, the GMM, and a reference solution. In most cases, NPG learns more accurate directional distributions than the competing methods in terms of the MAPE metric. Additionally, NPG produces a spatially *and* directionally continuous function.

MIS-Aware Optimization. In Figure 8.15, we demonstrate the increased robustness of neural path guiding offered by optimizing MIS weights. The impact is particularly noticeable on the cockpit of the spaceship seen through specular interactions, which are handled nearly optimally by sampling the material BSDF. In this region, a standard path tracer outperforms the learned sampling PDFs. With MIS-aware optimization—including the learning of MIS weights—the system downweights the contribution of the learned PDF on the cockpit, but increases it in regions where it is more accurate, resulting in significantly improved results overall.

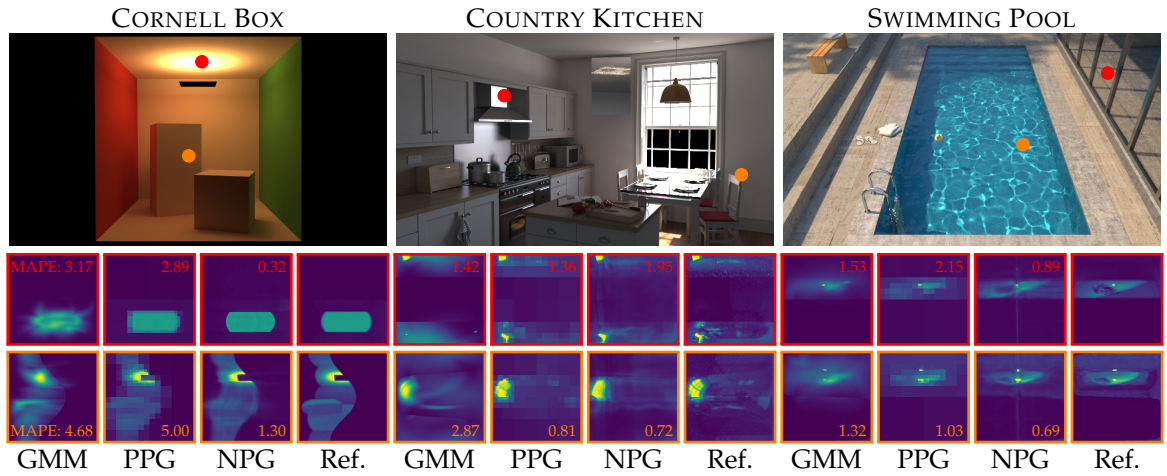


Figure 8.13: *Directional radiance distributions. From left to right: we visualize the distributions learned by a Gaussian mixture model (GMM) [Vorba et al. 2014], an SD-tree (PPG) [Müller et al. 2017], our neural path-guiding approach trained on radiance (NPG-Rad.), and a spatial binary tree with a directional regular* 128×128 *grid (Reference). The first three approaches were trained with an equal sample count and require roughly equal amounts of memory in the above scenes (around 10 MB). We used* 2^{16} *samples per pixel to generate the reference distributions, which require roughly* 5 GB *per scene. Despite its large computational cost, the reference solution is still slightly blurred (*CORNELL BOX*, red).*

Weight Reuse Across Camera Views. Figure 8.14 demonstrates the benefits of reusing networks, optimized for a particular camera view, in a novel view of the scene. We took network weights that resulted from generating images for Figures 8.8 and 8.10 as the initial weights for rendering images in the right column of Figure 8.14. Similarly to training from scratch, we keep optimizing the networks. If the initial distributions are already a good fit, our weighting scheme by the reciprocal mean pixel variance automatically keeps initial samples rather than discarding them.

Equal-Time Comparison. Lastly, we analyze the computational overhead of neural importance sampling in an equal-time comparison of unidirectional path tracing, PPG, and our product-driven NPG; see Figure 8.16. All techniques utilize a CPU for tracing paths. In addition, PPG uses the CPU for building and sampling the SD-tree, while our NPG also leverages two GPUs.

The radiance-driven PPG often performs best due to its small computational overhead, except when light transport is simple and/or the radiance-driven distribution is a poor fit to the product (e.g. the scenes in the top row). Despite utilizing two extra GPUs, the product-driven NPG is comparatively slow, on average constructing only about a quarter of the number of samples

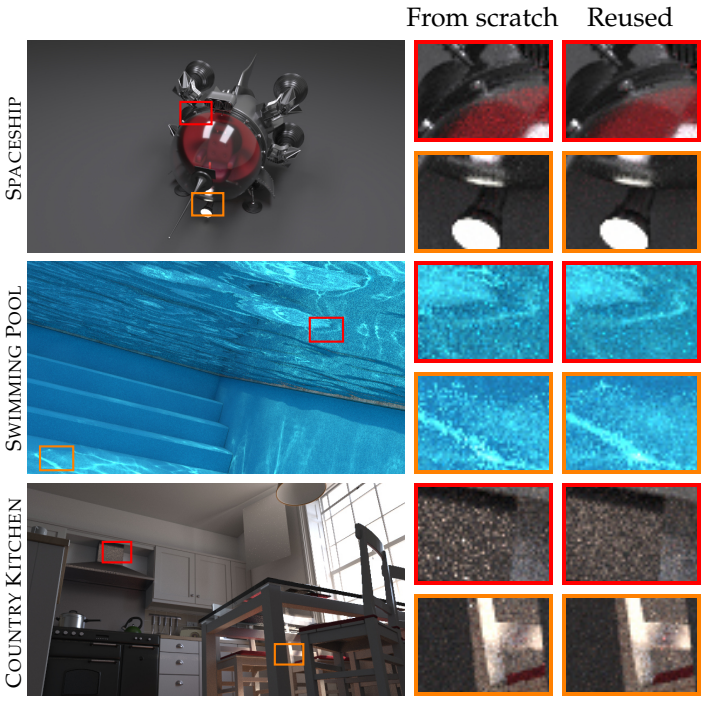


Figure 8.14: *Learned distributions can be reused for novel camera views to improve performance. The right column shows results where the network weights were initialized with weights learned for camera views in Figures 8.8 and 8.10.*

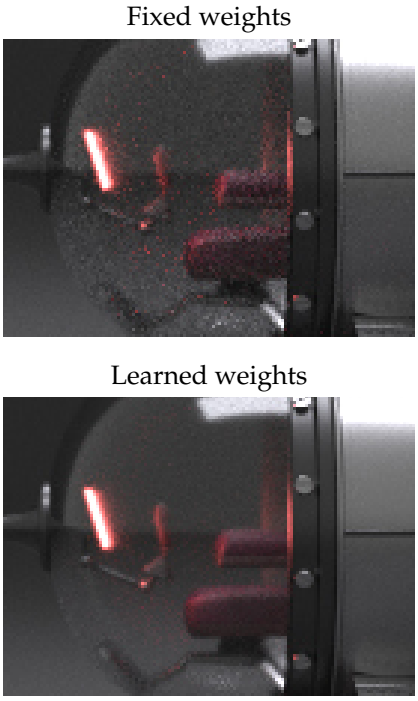


Figure 8.15: *Learning MIS weights leads to significantly better results on the* Spaceship *cockpit, where BSDF sampling is near optimal.*

that PPG constructs. However, since these samples are of "higher quality", the technique manages to close most of the performance gap to PPG and unidirectional path tracing, in some cases producing the best results (e.g. in Salle de Bain).

8.7 Discussion

Runtime Cost. An important property of practical sampling strategies is a low computational cost of generating samples and evaluating their PDF, relative to the cost of evaluating the integrand. In our path-guiding applications, the cost is dominated by the evaluation of coupling layers: roughly 10% of the time is spent on one-blob encoding, 60% on fully connected layers, and 30% on the piecewise-polynomial warp. This makes the overhead of our implementation prohibitive in simple scenes. While we focused on the theoretical challenge of applying neural networks to the problem of importance sampling in this work, accelerating the computation to make our

Figure 8.16: *Equal-time comparison of unidirectional path tracing (PT-Unidir), practical path guiding (PPG), and our product-driven neural path guiding (NPG-Product). Despite the large computation cost of NPG, it performs competitively with PPG and unidirectional path tracing in scenes with difficult light transport (bottom two rows, sorted by difficulty in ascending order).*

approach more practical is an important and interesting future work. We believe specialized hardware (e.g. NVIDIA's TensorCores) and additional computation graph optimization (e.g. NVIDIA's TensorRT) are promising next steps, which alone might be enough to bring our approach into the realm of practicality.

Optimizing for Multiple Integrals. In Section 8.5.1, we briefly discussed that the ground-truth density may be available only in unnormalized form. We argued that this is not a problem since the ignored factor F scales all gradients uniformly; it thus does not impact the optimization. These arguments pertain to handling a *single* integration problem. In Section 8.6, we demonstrated applications to path sampling and path guiding, where the learned density is conditioned on additional dimensions and we are thus solving many different integrals at once. Since the normalizing F varies between them, our arguments do not extend to this particular problem. Because neglecting the normalization factors is potentially negatively influencing the optimization, we experimented with tabulating F, but we did not experience

noticeable improvements. This currently stands as a limitation of applying our work to path guiding/sampling and it would be worth addressing in future work.

Convergence of Optimization. Although our optimization based on stochastic gradients has many advantages, it also brings certain disadvantages. Techniques based on stochastic gradient descent do not converge to local optima, but oscillate around them. This also happens during neural path guiding. The problem is well known in machine learning literature and is usually addressed by decaying the learning rate over time. We opted not to decay our learning rate for simplicity, because finding an optimal decay schedule is a difficult problem. Solving this issue in the future would likely improve our results further, perhaps significantly.

Alternative Training Schemes. Dahm and Keller [2017] learn the 5D radiance field by minimizing an approximation (via Q-learning) of the total-variation divergence. While they cannot sample from the resulting 5D distribution, they are able to learn near-optimal light selection probabilities for next event estimation. This optimization strategy and variations thereof are an interesting alternative to our KL and χ^2 divergence loss functions. Another attractive goal is a unified optimization across multiple different scenes, rather than training from scratch for each one. A potentially fruitful extension our approach would be to apply a higher-level optimization strategy in the spirit of "learning to learn" Andrychowicz et al. [2016]; Chen et al. [2017].

Failure Cases. In the pathologically difficult YET ANOTHER BOX scene, the theoretically inferior radiance-based NPG produces slightly better results than product-based NPG. We suspect that this is caused by the product distribution being much more complicated and therefore more difficult to learn than the radiance distribution. Furthermore, in the BOOKSHELF scene, our approaches perform worse than the GMM algorithm by Vorba et al. [2014]. Although in our experiments our method exhibits fewer of such failure cases than PPG and the GMMs, a detailed investigation into their causes is still to be carried out and could gain interesting insights.

Scene Scale. We studied the performance of neural path guiding when all positions that are input to it are relatively close compared to the scene bounding box. We artificially scaled the positional inputs by 10^{-5} in the COUNTRY KITCHEN scene, observing a roughly $2\times$ larger error. While the method

still outperforms path tracing by a big margin, alleviating this limitation is promising future work.

Alternative Variance-Reduction Techniques. In this chapter, we studied the application of neural networks to importance sampling. Other variance-reduction techniques, such as control variates, could enjoy analogous benefits. We believe similar derivations to Section 8.5 can be made, leading to an interconnected gradient-descent-based optimization of multiple variance reduction techniques.

8.8 Acknowledgments

We thank Sebastian Herholz and Yining Karl Li for valuable feedback on the text of this chapter, Vorba et al. [2014] and Dinh et al. [2016] for releasing the source code of their work, and Thijs Vogels for bringing RealNVP to our attention. We also thank the following people for providing scenes and models that appear in this chapter: Benedikt Bitterli [2016], Ondřej Karlík (SWIMMING POOL), Johannes Hanika (GLOSSY KITCHEN, NECKLACE), Samuli Laine and Olesya Jakob (COPPER HAIRBALL), Jay-Artist (COUNTRY KITCHEN, WHITE ROOM), Marko Dabrović (SPONZA ATRIUM), Miika Aittala, Samuli Laine, and Jaakko Lehtinen (VEACH DOOR), Nacimus (SALLE DE BAIN), SlykDrako (BEDROOM), thecali (SPACESHIP) Tiziano Portenier (BATHROOM, BOOKSHELF), and Wig42 (STAIRCASE).

CHAPTER 9

Conclusion

In this dissertation, we presented three data-driven techniques that increase the efficiency of path-traced light-transport simulation. We conclude by summarizing the results and findings from each of the techniques, outlining promising future research directions, and providing an outlook towards the future of machine learning in the field of light-transport simulation.

9.1 Multiple Scattering in Translucent Materials

Shell Transport Functions. We presented a new technique for rendering translucent materials that is based on shell transport functions [Moon et al. 2007; Lee and O'Sullivan 2007]. Our technique only requires a single scene-independent precomputation and has comparable performance to diffusion-based approximations while yielding significantly higher accuracy. We achieved scene-independent precomputation of shell transport functions by introducing a parameterization that allows the usage of white Monte Carlo [Alerstam et al. 2008] which we imported from the field of biomedical optics.

Heterogeneous Granular Media. Since our shell transport functions operate on continuous participating media, we were able to apply them to rendering of heterogeneous *granular* media by extending previous work by [Meng et al. 2015; Müller 2016] that can approximate granular media as continuous media. We additionally introduced an error-analysis-based heuristic that combines explicit path tracing, proxy path tracing [Müller 2016], and our

shell tracing in a scene-adaptive manner, resulting in a holistic acceleration of rendering heterogeneous granular media. Parts of the algorithm were eventually implemented in Disney's Hyperion renderer.

Future Work on Shell Transport Functions. The main limitation of our parameterization of shell transport functions is that it relies on the Henyey-Greenstein phase function. Incorporating other phase functions would increase the STF dimensionality infeasibly. In the future, it would be interesting to investigate phase-function representations that are independent from the underlying model, similar to the BSDF representation of Herholz et al. [2016, 2018], to permit a more general low-dimiensional STF parameterization.

Future Work on Granular Media. The continuous-volume approximation that we use [Meng et al. 2015; Müller 2016] only supports independently oriented grains that are situated within non-overlapping bounding spheres. Real granular media are often packed much more tightly in such a way that neighboring grains influence each other's orientation. It is therefore paramount to take such situations into account in order to synthesize the granular appearance more accurately. Since it is difficult to parameterize the mutual physical interactions between large numbers of grains with few dimensions, it may be worthwhile to investigate the applicability of deep neural networks, which excel at modeling high-dimensional functions.

9.2 Path Guiding

Our path-guiding algorithms accelerate path tracing by learning from data to importance sample the rendering equation in such a way that rendering remains consistent and unbiased.

Practical Path Guiding. The first path-guiding algorithm we introduced is PPG: a practical and easy-to-implement approach with a low-latency rendering preview. PPG performs path guiding by learning the 5-dimensional incident radiance field in a spatio-directional tree in an on-line fashion and does not require tuning of hyper-parameters, owing to several heuristics that we carefully designed to be as robust as possible. While more sophisticated algorithms may outperform our technique in specific situations, its high performance and robustness make it appealing for production environments. We demonstrated this on a simple unidirectional path tracer that

outperformed more sophisticated techniques in all our comparisons. Another benefit is the fairly even spatial convergence, which we expect to increase the attractiveness of PPG for animation rendering, when combined with a suitable technique for removing fireflies. Finally, since PPG is largely agnostic to the underlying path-generation algorithm, our algorithm can be easily applied to bidirectional path tracing, progressive photon mapping, metropolis light transport, or other techniques, improving their performance beyond the demonstrated results. This algorithm has been implemented in several production renderers, including Disney's Hyperion.

Neural Importance Sampling. Lastly, we introduced a general technique for importance sampling Monte Carlo estimators by utilizing neural networks. We then used our technique to enhance the PPG algorithm we presented beforehand. Our neural-importance-sampling approach builds on prior work on normalizing flows, RealNVP [Dinh et al. 2014, 2016], which we extended in three novel manners. First, we proposed piecewise-polynomial coupling transforms that have increased modeling power over prior coupling transforms. Second, we encoded our network inputs with a novel one-blob encoding, a generalization of one-hot encoding, that helped our neural networks to learn irregular high-frequency signals. And third, we derived an optimization strategy that directly minimizes the variance of Monte Carlo estimators that draw samples from trainable probabilistic models such as ours. After replacing the SD-tree of PPG with an on-line-learned version of our approach, we obtained higher-quality results—sometimes by significant margins—at equal sample counts. Unfortunately, a remaining challenge of our technique is its large computational cost which limits its usefulness compared to the highly optimized PPG algorithm in simple settings. We believe that our work may be beneficial in other high-dimensional Monte Carlo integration problems beyond the simulation of light transport.

Future Work. Since we ignored volumetric light transport in our path-guiding methods it would be interesting to investigate path guiding in the context of rendering participating media. To this end, we anticipate the need for an adapted data structure that not only learns to importance sample scattering directions, but also free-flight distances through volumes. In the realm of *neural* path guiding specifically, it would be interesting to investigate computationally cheaper architectures and easier-to-optimize loss functions that go beyond the simple divergence metrics that we used. Furthermore, due to the advent of real-time ray-traced graphics [Burnes 2018], we believe further research towards path-guiding that can quickly adapt to rapid changes

in scene geometry and illumination is important. Lastly, we think that the data-driven approach to path guiding can also be applied to alternative (and sometimes orthogonal) variance-reduction, for example to the technique of control variates.

9.3 Outlook

We found machine learning to be a remarkably useful tool to increase the efficiency of path-traced light-transport simulation. This was the case not only when we used deep learning, but also when we used simpler models based on piecewise-linear approximations, suggesting that there still is a large body of interesting cross-domain applications of existing machine-learning techniques to be explored. Researchers already began to draw parallels between reinforcement learning and path guiding [Dahm and Keller 2017, 2018], making us excited to see which other connections may be discovered in the future.

This is not to say that the recent advances in deep learning are any less interesting. Given the success of deep learning across many fields, it seems likely that many more approaches aside from ours will emerge in light-transport simulation. We hope that our findings are useful stepping stones in this direction towards the eventual goal of real-time photorealistic rendering.

APPENDIX A

Light Transport in Computer Graphics

A.1 Derivation of the Path Integral

In Section 4.5.4 we claimed that the rendering equation can be expressed as the path integral

$$I_{xy} = \int_{\mathcal{P}} L_e(\mathbf{x}_1 \to \mathbf{x}_2) T(\bar{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \, \mathrm{d}\Pi(\bar{\mathbf{x}}) \tag{A.1}$$

$$T(\bar{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \prod_{i=2}^{k-1} T(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) E(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \,, \tag{A.2}$$

but we did not provide a proof of the validity of the path throughput $T(\bar{\mathbf{x}})$. In this section, we postulate

$$I_{xy} = \int_{\mathcal{P}} L_e(\mathbf{x}_1 \to \mathbf{x}_2) T(\bar{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \, \mathrm{d}\Pi(\bar{\mathbf{x}}) \tag{A.3}$$

$$= \sum_{k=2}^{\infty} \int_{\mathcal{P}_k} L_e(\mathbf{x}_1 \to \mathbf{x}_2) T(\bar{\mathbf{x}}_k) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \, \mathrm{d}\Pi(\bar{\mathbf{x}}_k) \tag{A.4}$$

and derive $T(\bar{\mathbf{x}})$ such that the above integral corresponds to the recursively expanded rendering equation. We will perform this derivation by induction over the number of path vertices.

Base case (surface vertices). Let us first consider the base case of $k = 2$ while restricting the path vertices to surfaces only, then the measurement equation becomes

$$I_{xy} = \int_{\partial\mathcal{V}} \int_{\mathcal{S}^2} L_e \mathbf{x}_1 \mathbf{x}_2 \tau(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) W_{xy}(\mathbf{x}_1 \to \mathbf{x}_2) \, \mathrm{d}\Omega(\omega_{1,2}) \, \mathrm{d}A^{\top}_{\omega}(\mathbf{x}_1) \,. \tag{A.5}$$

Our goal is to integrate with respect to $\mathrm{d}\Pi(\bar{\mathbf{x}}) = \mathrm{d}A(\mathbf{x}_1)\,\mathrm{d}A(\mathbf{x}_2)$ rather than $\mathrm{d}\Omega(\omega_{1,2})\,\mathrm{d}A^{\top}_{\omega}(\mathbf{x}_1)$. Using the identities

$$\mathrm{d}A^{\top}_{\omega}(\mathbf{x}_i) = \mathrm{d}A(\mathbf{x}_i)\,|N(\mathbf{x}_i)\cdot\omega_{ij}| = \mathrm{d}A(\mathbf{x}_i)\,|\cos\gamma_i|\,, \tag{A.6}$$

$$\mathrm{d}\Omega(\omega_{ij}) = \mathrm{d}A^{\top}_{\omega}(\mathbf{x}_j)\frac{1}{\|\mathbf{x}_j-\mathbf{x}_i\|^2} = \mathrm{d}A(\mathbf{x}_j)\frac{|\cos\gamma_j|}{\|\mathbf{x}_j-\mathbf{x}_i\|^2}\,, \tag{A.7}$$

we can adjust the integral to the desired form

$$\begin{aligned} I_{xy} &= \int_{\partial\mathcal{V}}\int_{\partial\mathcal{V}} L_{\mathrm{e}}(\mathbf{x}_1\to\mathbf{x}_2)\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\,\frac{|\cos\gamma_1|\,|\cos\gamma_2|}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,W_{xy}(\mathbf{x}_1\to\mathbf{x}_2)\,\mathrm{d}A(\mathbf{x}_2)\,\mathrm{d}A(\mathbf{x}_1) \\ &= \int_{\mathcal{P}_2} L_{\mathrm{e}}(\mathbf{x}_1\to\mathbf{x}_2)\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\,\frac{|\cos\gamma_1|\,|\cos\gamma_2|}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,W_{xy}(\mathbf{x}_1\to\mathbf{x}_2)\,\mathrm{d}\Pi(\bar{\mathbf{x}})\,. \end{aligned} \tag{A.8}$$

Note, that the change of variables from $\omega_{1,2}$ to $\mathbf{x}_2$ allows for mutual occlusion of surfaces, which we handle by setting $\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)$ to zero in such cases, i.e. $r(\mathbf{x}_i,\omega_{ij})\neq\mathbf{x}_j \implies \tau(\mathbf{x}_i,\mathbf{x}_j)=0$.

By pattern-matching with the desired path integral (A.4), we find that

$$T(\bar{\mathbf{x}}) = \tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\,G_{\partial\mathcal{V}}(\mathbf{x}_1\leftrightarrow\mathbf{x}_2) \tag{A.9}$$

$$G_{\partial\mathcal{V}}(\mathbf{x}_i\leftrightarrow\mathbf{x}_j) = \frac{|\cos\gamma_i|\,|\cos\gamma_j|}{\|\mathbf{x}_i-\mathbf{x}_j\|^2} \tag{A.10}$$

where we refer to $G_{\partial\mathcal{V}}(\mathbf{x}_i\leftrightarrow\mathbf{x}_j)$ as the "surface geometry term".

Base case (volume vertices). Let us now consider the same base case ($k=2$) while constraining the path vertices to *non-surface* (i.e. *volume*) positions. Like above, we write the measurement equation for this case

$$I_{xy} = \int_{\mathcal{V}}\int_{\mathcal{S}^2}\int_0^d L_{\mathrm{e}}(\mathbf{x}_t\to\mathbf{x}_2)\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_t)W_{xy}(\mathbf{x}_1\to\mathbf{x}_2)\,\mathrm{d}t\,\mathrm{d}\Omega(\omega_{1,2})\,\mathrm{d}V(\mathbf{x}_1)\,. \tag{A.11}$$

Here, our goal is to perform a change of measure from $\mathrm{d}t\,\mathrm{d}\Omega(\omega_{1,2})\,\mathrm{d}V(\mathbf{x}_1)$ to $\mathrm{d}\Pi(\bar{\mathbf{x}}) = \mathrm{d}V(\mathbf{x}_1)\,\mathrm{d}V(\mathbf{x}_2)$. To this end, we use the identity

$$\mathrm{d}t\,\mathrm{d}\Omega(\omega_{ij}) = \mathrm{d}V(\mathbf{x}_j)\frac{1}{\|\mathbf{x}_j-\mathbf{x}_i\|^2}\,, \tag{A.12}$$

leading to

$$\begin{aligned} I_{xy} &= \int_{\mathcal{V}}\int_{\mathcal{V}} L_{\mathrm{e}}(\mathbf{x}_1\to\mathbf{x}_2)\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\,\frac{1}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,W_{xy}(\mathbf{x}_1\to\mathbf{x}_2)\,\mathrm{d}V(\mathbf{x}_1)\,\mathrm{d}V(\mathbf{x}_2) \\ &= \int_{\mathcal{P}_2} L_{\mathrm{e}}(\mathbf{x}_1\to\mathbf{x}_2)\tau(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\,\frac{1}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,W_{xy}(\mathbf{x}_1\to\mathbf{x}_2)\,\mathrm{d}\Pi(\bar{\mathbf{x}})\,, \end{aligned} \tag{A.13}$$

where, analogously to the surface case, we define a "volume geometry term" $G_{\mathcal{V}}(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$ to obtain

$$T(\bar{\mathbf{x}}) = \tau(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, G_{\mathcal{V}}(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \tag{A.14}$$

$$G_{\mathcal{V}}(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) = \frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}\,. \tag{A.15}$$

Base case (any vertices). Equivalent derivations can be performed for the base cases ($k = 2$) where $\mathbf{x}_1 \in \partial\mathcal{V} \wedge \mathbf{x}_2 \in \mathcal{V}$ as well as $\mathbf{x}_1 \in \mathcal{V} \wedge \mathbf{x}_2 \in \partial\mathcal{V}$, leading to the *volume-surface-agnostic* expression of $T(\bar{\mathbf{x}})$ as

$$T(\bar{\mathbf{x}}) = \tau(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\,, \tag{A.16}$$

$$G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) = \frac{D(\mathbf{x}_i, \omega_{ij})\, D(\mathbf{x}_j, \omega_{ji})}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}\,, \tag{A.17}$$

$$D(\mathbf{x}, \omega) = \begin{cases} |N(\mathbf{x}) \cdot \omega| & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ 1 & \text{otherwise.} \end{cases} \tag{A.18}$$

The shorthand $E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$, which we will henceforth call the "edge throughput", captures volumetric attenuation and geometric effects along the path edge.

Induction (surface vertex). Suppose that for paths with some number of vertices k the path throughput function $T(\bar{\mathbf{x}}_k)$ is known and that the first path vertex resides on a surface ($\mathbf{x}_1 \in \partial\mathcal{V}$). We will derive the path throughput function of $k+1$ vertex long paths by replacing surface emission at $\mathbf{x}_1$ with one-fold surface scattering (i.e. emission happens one vertex beforehand). To this end, we adjust the above measurement equation for paths of length k according to the scattering equation (4.15) by replacing $L_{\mathrm{e}}(\mathbf{x}_1 \to \mathbf{x}_2)$ with $L_{\mathrm{s}}(\mathbf{x}_1 \to \mathbf{x}_2)$ and simplifying:

$$\begin{aligned} I_{xy} &= \int_{\mathcal{P}_k} L_{\mathrm{s}}(\mathbf{x}_1 \to \mathbf{x}_2) T(\bar{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k)\, \mathrm{d}\Pi(\bar{\mathbf{x}}) \\ &= \int_{\mathcal{P}_k} \int_{\mathcal{S}^2} L_{\mathrm{i}}(\mathbf{x}_1, \omega_{\mathrm{i}}) f_{\mathrm{s}}(\mathbf{x}_1, \omega_{\mathrm{i}} \to \omega_{1,2})\, |\cos\gamma_{\mathrm{i}}|\, \mathrm{d}\Omega(\omega_{\mathrm{i}})\, T(\bar{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k)\, \mathrm{d}\Pi(\bar{\mathbf{x}}) \\ &= \int_{\mathcal{P}_{k+1}} L_{\mathrm{e}}(\mathbf{x}_1 \to \mathbf{x}_2) E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) f_{\mathrm{s}}(\mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3) T(\bar{\mathbf{x}}_{2:k+1}) W_{xy}(\mathbf{x}_k \to \mathbf{x}_{k+1})\, \mathrm{d}\Pi(\bar{\mathbf{x}})\,, \end{aligned} \tag{A.19}$$

where we denote $\bar{\mathbf{x}}_{i:j} = \mathbf{x}_i \ldots \mathbf{x}_j$. In the last step, we use the same identities as in the case $k = 2$, where the exact identities that are used depend on whether $\mathbf{x}_1$ is on a surface or not. It follows that $\forall \bar{\mathbf{x}} \in \{\mathcal{P}_{k+1} \,|\, \mathbf{x}_2 \in \partial\mathcal{V}\}$:

$$T(\bar{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, f_{\mathrm{s}}(\mathbf{x}_1 \leftarrow \mathbf{x}_2 \leftarrow \mathbf{x}_3)\, T(\bar{\mathbf{x}}_{2:k+1})\,. \tag{A.20}$$

Induction (volume vertex). Analogously to the surface case, we begin with the k-vertex measurement equation but instead of assuming the first vertex is on a surface, we assume it lies in a volume instead. We therefore replace $L_\mathrm{e}(\mathbf{x}_1 \to \mathbf{x}_2)$ with $L_\mathrm{in}(\mathbf{x}_1 \to \mathbf{x}_2) = (\omega_{1,2} \cdot \nabla) L_\mathrm{in}(\mathbf{x}_1, \omega_2)$ and simplify as follows:

$$\begin{aligned} I_{xy} &= \int_{\mathcal{P}_k} L_\mathrm{in}(\mathbf{x}_1 \to \mathbf{x}_2) T(\overline{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \,\mathrm{d}\Pi(\overline{\mathbf{x}}) \\ &= \int_{\mathcal{P}_k} \int_{\mathcal{S}^2} L_\mathrm{i}(\mathbf{x}_k \to \omega_\mathrm{i}) \sigma_\mathrm{s}(\mathbf{x}_1) f_\mathrm{p}(\mathbf{x}_1, \omega_\mathrm{i} \to \omega_{1,2}) \,\mathrm{d}\Omega(\omega_\mathrm{i}) T(\overline{\mathbf{x}}) W_{xy}(\mathbf{x}_{k-1} \to \mathbf{x}_k) \,\mathrm{d}\Pi(\overline{\mathbf{x}}) \\ &= \int_{\mathcal{P}_{k+1}} L_\mathrm{e}(\mathbf{x}_1 \to \mathbf{x}_2) E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \sigma_\mathrm{s}(\mathbf{x}_2) f_\mathrm{p}(\mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3) T(\overline{\mathbf{x}}_{2:k+1}) W_{xy}(\mathbf{x}_k \to \mathbf{x}_{k+1}) \,\mathrm{d}\Pi(\overline{\mathbf{x}}) \,. \end{aligned} \tag{A.21}$$

Once again, in the last step, we use the same identities as in the case $k = 2$, where the exact identities that are used depend on whether $\mathbf{x}_1$ is on a surface or not. This time, we obtain the recurrence $\forall \overline{\mathbf{x}} \in \{\mathcal{P}_{k+1} \,|\, \mathbf{x}_2 \in \mathcal{V}\}$:

$$T(\overline{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \, \sigma_\mathrm{s}(\mathbf{x}_2) f_\mathrm{p}(\mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3) \, T(\overline{\mathbf{x}}_{2:k+1}) \,. \tag{A.22}$$

Induction (any vertex). We can now assemble both recurrences together, yielding $\forall \overline{\mathbf{x}} \in \mathcal{P}_{k+1}$

$$T(\overline{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \, f(\mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3) \, T(\overline{\mathbf{x}}_{2:k+1}) \,, \tag{A.23}$$

$$f(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_j) = \begin{cases} f_\mathrm{s}(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) & \text{if } \mathbf{x} \in \partial\mathcal{V} \\ \sigma_\mathrm{s}(\mathbf{x}_j) f_\mathrm{p}(\mathbf{x}_i \to \mathbf{x}_j \to \mathbf{x}_k) & \text{otherwise.} \end{cases} \tag{A.24}$$

Combining this recursion with the base case $T(\overline{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)$, we finally obtain the desired path throughput $T(\overline{\mathbf{x}})$, $\forall \overline{\mathbf{x}} \in \mathcal{P}$ by induction

$$T(\overline{\mathbf{x}}) = E(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \prod_{i=2}^{k-1} f(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}) E(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \,. \tag{A.25}$$

APPENDIX B

Multiple Scattering in Translucent Materials

B.1 Estimated GSDF Error

In Figure B.1 through Figure B.4, we visualize the estimated approximation error introduced by our GSDFs in the directional domain $e_{f_g}(\omega_i, \omega_o)$ for various grain types. The visualized grains are, in order, "glass sphere", "brown sand", "snowflake", and "cinnamon" from Figure 6.7.

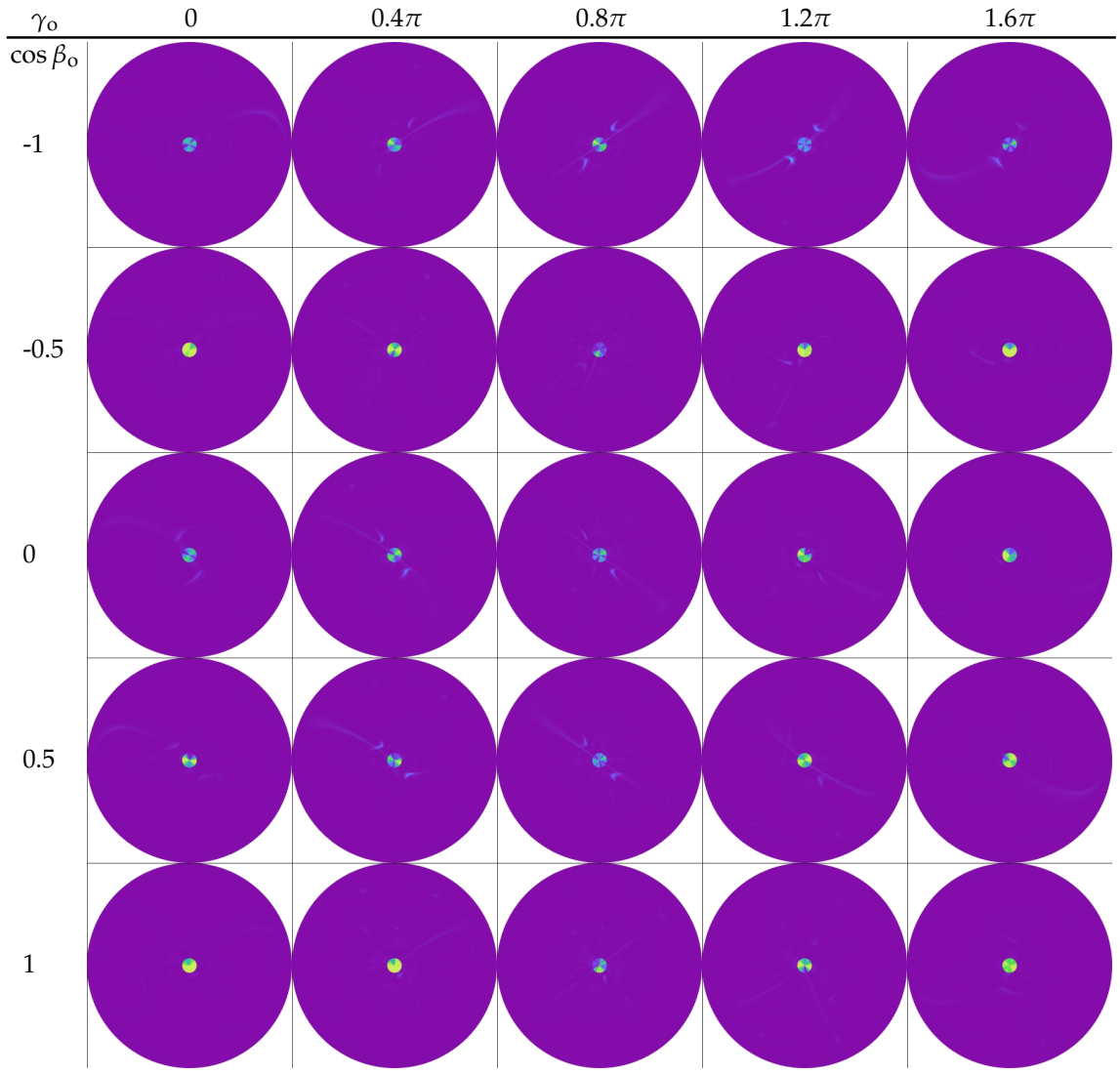


Figure B.1: *Front-facing directional GSDF approximation error $e_{f_g}(\omega_i, \omega_o)$ on snowflake grains. We visualize the 4-dimensional error function by plotting slices through $\cos\beta_o$ (vertical) and γ_o (horizontal). The individual circular heatmaps encode $\sin\beta_i$ as the distance from their center and γ_i as their rotational component.*

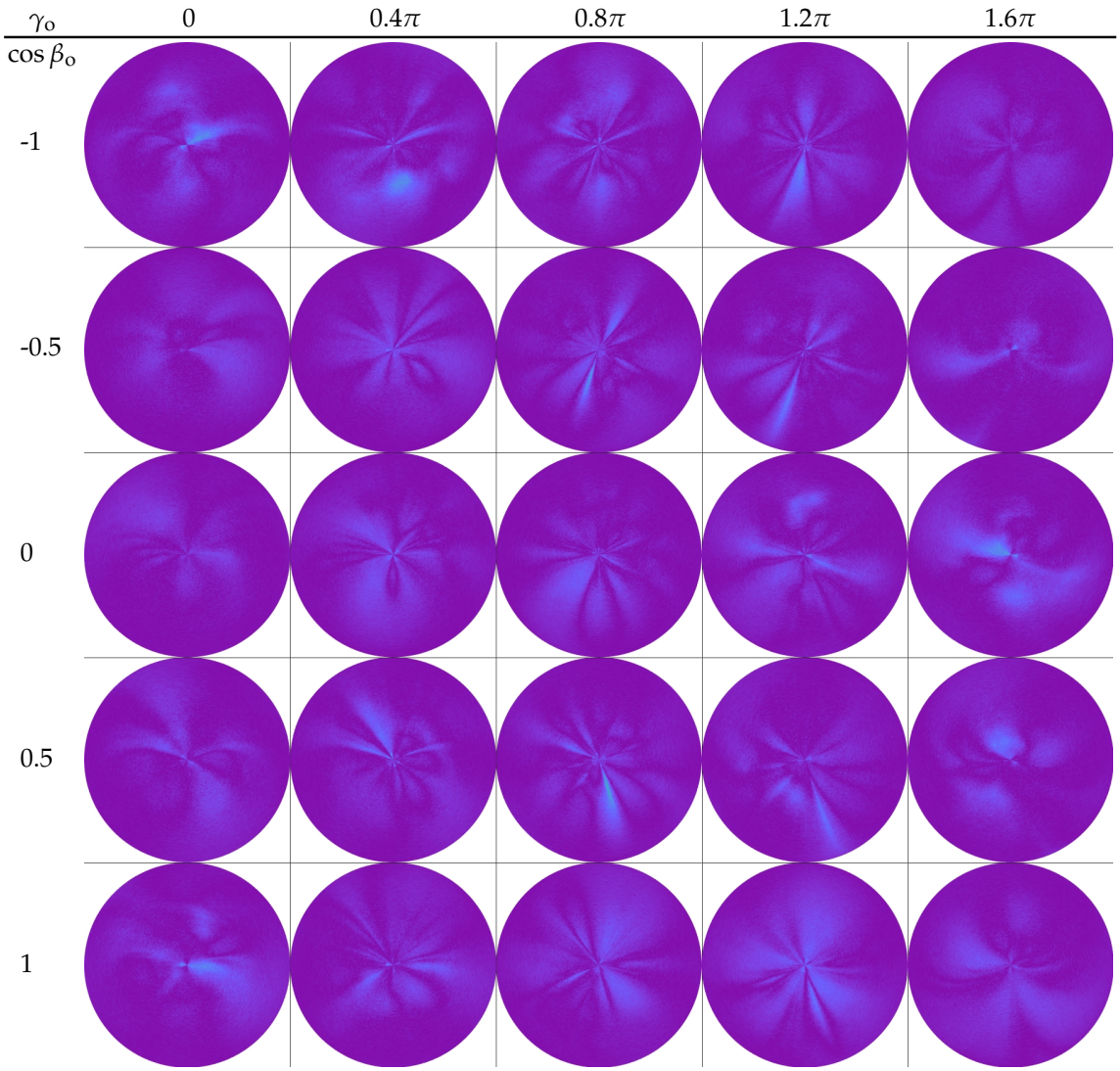


Figure B.2: *Front-facing directional GSDF approximation error $e_{f_g}(\omega_i, \omega_o)$ on brown sand grains. We visualize the 4-dimensional error function by plotting slices through $\cos\beta_o$ (vertical) and γ_o (horizontal). The individual circular heatmaps encode $\sin\beta_i$ as the distance from their center and γ_i as their rotational component.*

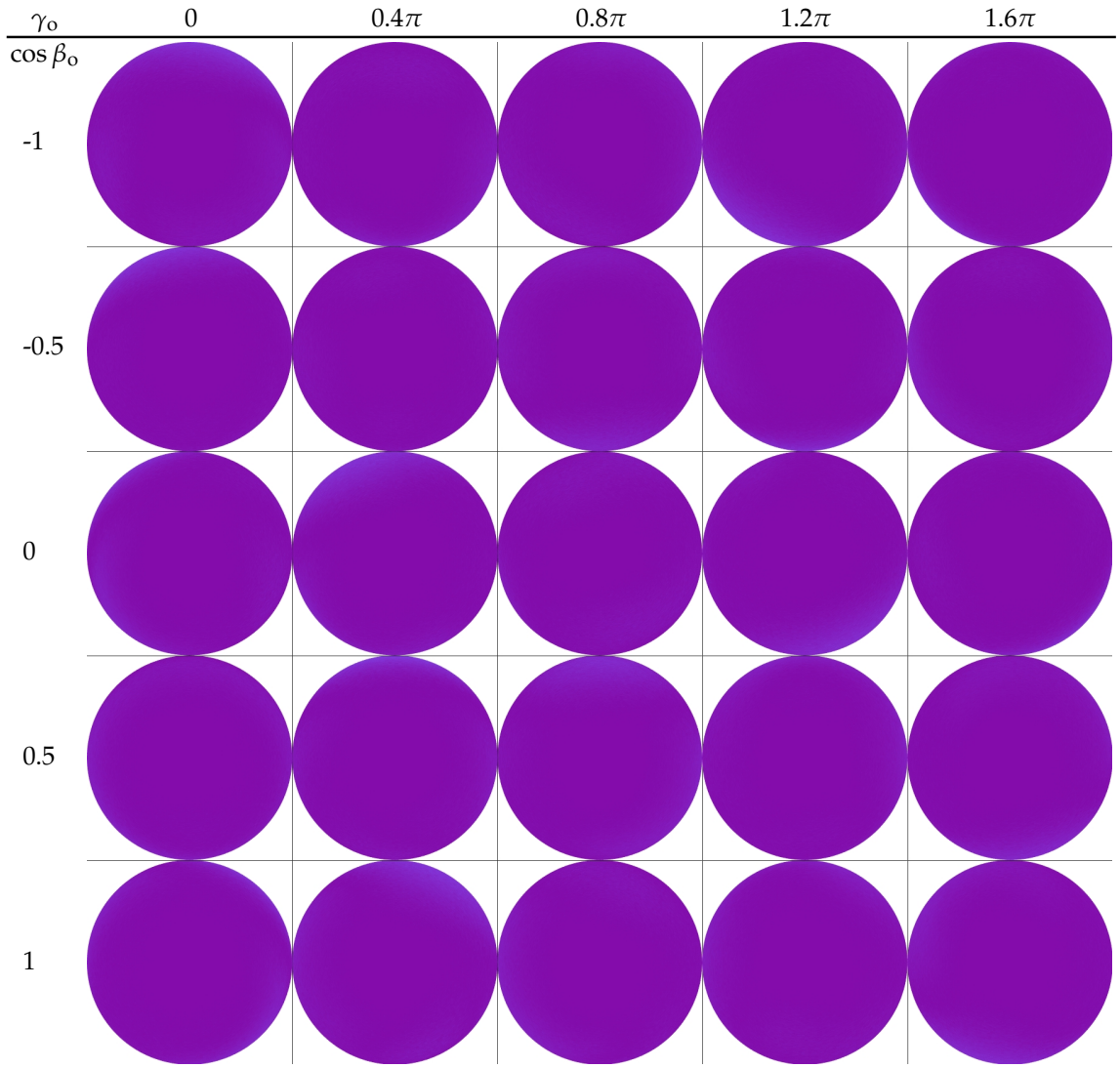


Figure B.3: *Front-facing directional GSDF approximation error $e_{f_g}(\omega_i, \omega_o)$ on cinnamon grains. We visualize the 4-dimensional error function by plotting slices through $\cos\beta_o$ (vertical) and γ_o (horizontal). The individual circular heatmaps encode $\sin\beta_i$ as the distance from their center and γ_i as their rotational component.*

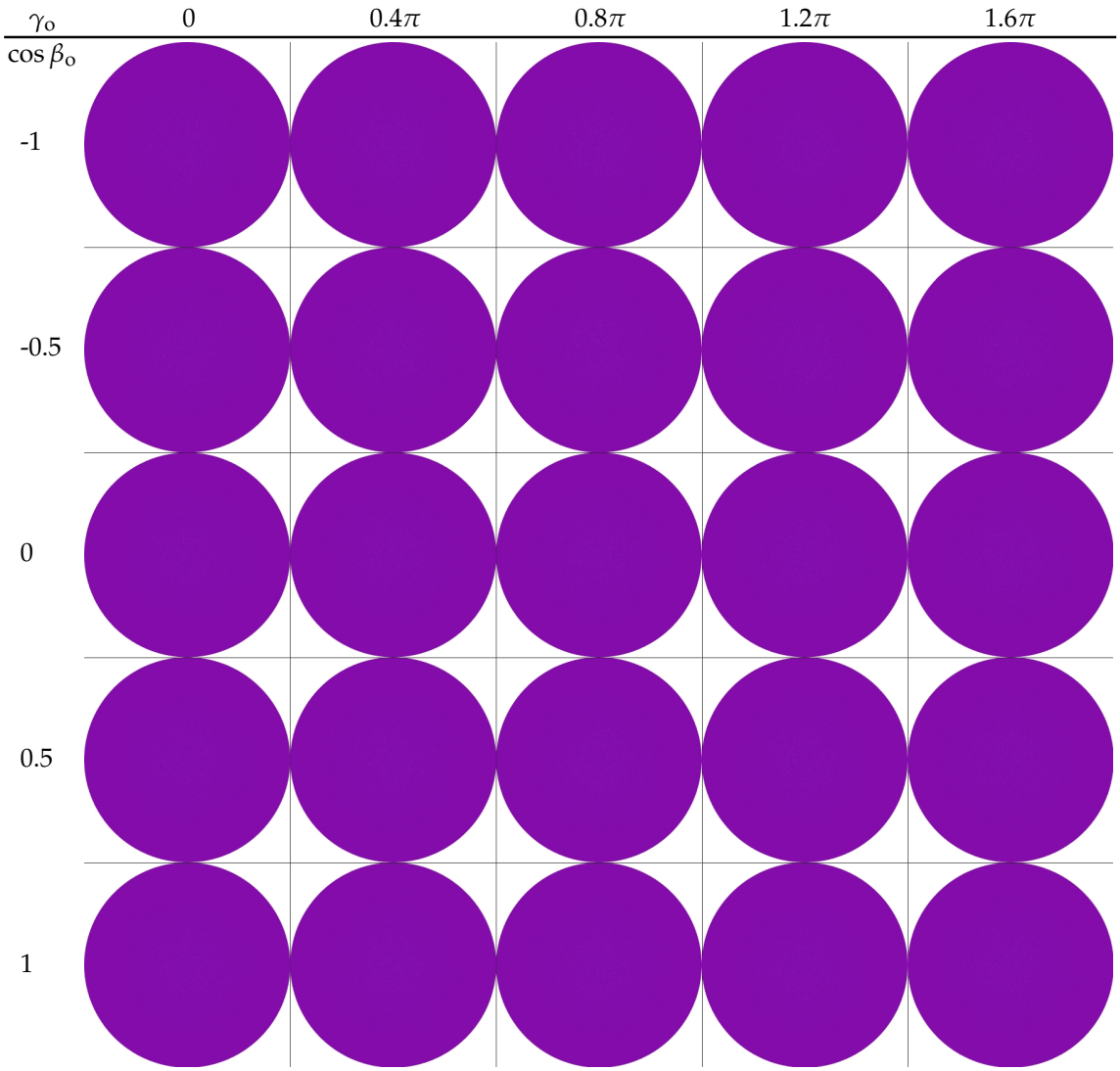


Figure B.4: *Front-facing directional GSDF approximation error $e_{f_g}(\omega_i, \omega_o)$ on dielectric sphere grains. We visualize the 4-dimensional error function by plotting slices through $\cos\beta_o$ (vertical) and γ_o (horizontal). The individual circular heatmaps encode $\sin\beta_i$ as the distance from their center and γ_i as their rotational component.*

APPENDIX C

Practical Path Guiding

C.1 Proof of $\hat{V}_k$ Being Convex

We want to show, that $\hat{V}_k$ is convex, i.e.

$$2\hat{V}_k \leq \hat{V}_{k+1} + \hat{V}_{k-1}\,. \tag{C.1}$$

Proof. Let $\forall k : B_k > 0,\ \hat{B}_k > 0,\ \tau_k > 0$. We write Eq. C.1 terms of τ as

$$\frac{2\tau_k}{\hat{B}_k} \leq \frac{\tau_{k+1}}{\hat{B}_{k+1}} + \frac{\tau_{k-1}}{\hat{B}_{k-1}}\,, \tag{C.2}$$

$$2\tau_k \leq \tau_{k+1}\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\frac{\hat{B}_k}{\hat{B}_{k-1}}\,. \tag{C.3}$$

We use the convexity of τ_k to obtain the tighter inequality

$$2\tau_k \leq (2\tau_k - \tau_{k-1})\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\frac{\hat{B}_k}{\hat{B}_{k-1}}\,, \tag{C.4}$$

$$2\tau_k \leq 2\tau_k\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\left(\frac{\hat{B}_k}{\hat{B}_{k-1}} - \frac{\hat{B}_k}{\hat{B}_{k+1}}\right). \tag{C.5}$$

We further tighten the inequality by using $\tau_k < \tau_{k-1}$ as

$$2\tau_k \leq 2\tau_k\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_k\left(\frac{\hat{B}_k}{\hat{B}_{k-1}} - \frac{\hat{B}_k}{\hat{B}_{k+1}}\right), \tag{C.6}$$

$$2 \leq \frac{\hat{B}_k}{\hat{B}_{k+1}} + \frac{\hat{B}_k}{\hat{B}_{k-1}} = \frac{\hat{B}_k}{\hat{B}_k - B_k} + \frac{\hat{B}_k}{\hat{B}_k + B_{k-1}}\,. \tag{C.7}$$

Through re-arrangement and simplification we obtain

$$B_{k-1} - B_k - \frac{B_{k-1} B_k}{\hat{B}_k} \leq 0\,, \tag{C.8}$$

which holds, because the sequence B_k is monotonically increasing and always positive (see Section 7.3). □

APPENDIX D

Neural Importance Sampling

D.1 Determinant of Coupling Layers

Here we include the derivation of the Jacobian determinant akin to Dinh et al. [2016]. The Jacobian of a single coupling layer, where $A = [\![1, d]\!]$ and $B = [\![d+1, D]\!]$, is a block matrix:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} I_d & 0 \\ \frac{\partial C\left(x^B; m(x^A)\right)}{\partial (x^A)^T} & \frac{\partial C\left(x^B; m(x^A)\right)}{\partial (x^B)^T} \end{bmatrix}, \tag{D.1}$$

where I_d is a $d \times d$ identity matrix. The determinant of the Jacobian matrix reduces to the determinant of the lower right block. Note that the Jacobian $\frac{\partial C\left(x^B; m(x^A)\right)}{\partial (x^A)^T}$ (lower left block) does not appear in the determinant, hence m can be arbitrarily complex.

For the multiply-add coupling transform Dinh et al. [2016] we get

$$\frac{\partial C\left(x^B; m(x^A)\right)}{\partial (x^B)^T} = \begin{bmatrix} e^{s_1} & & 0 \\ & \ddots & \\ 0 & & e^{s_{D-d}} \end{bmatrix}. \tag{D.2}$$

The diagonal nature stems from the separability of the coupling transform. The determinant of the coupling layer in the forward and the inverse pass therefore reduce to $e^{\sum s_i}$ and $e^{-\sum s_i}$, respectively.

In our piecewise-polynomial coupling transforms, we maintain separability to preserve the diagonal Jacobian, i.e.

$$C\left(x^B; m(x^A)\right) = \left(C_1\left(x_1^B; m(x^A)\right), \cdots, C_{D-d}\left(x_{D-d}^B; m(x^A)\right)\right)^T,$$

and therefore, using $\frac{\partial C_i\left(x_i^B; m(x^A)\right)}{\partial x_i^B} = q_i(x_i^B)$, we get

$$\frac{\partial C\left(x^B; m(x^A)\right)}{\partial \left(x^B\right)^T} = \begin{bmatrix} q_1(x_1^B) & & 0 \\ & \ddots & \\ 0 & & q_{D-d}(x_{D-d}^B) \end{bmatrix}. \tag{D.3}$$

The determinant thus is the product of the marginal PDFs defining the piecewise-polynomial warp along each dimension $\prod_{i=1}^{D-d} q_i(x_i^B)$.

D.2 Adaptive Bin Sizes in Piecewise-Linear Coupling Functions

Without loss of generality, we investigate the simplified scenario of a one-dimensional input $A = \emptyset$ and $B = \{1\}$, a single coupling layer $L = 1$ and the KL-divergence loss function. Further, let the coupling layer admit a piecewise-linear coupling transform—i.e. it predicts a piecewise-constant PDF—with $K = 2$ bins. Let the width W of the 2 bins be controlled by traininable parameter $\theta \in \mathbb{R}$ such that $W_1 = \theta$ and $W_2 = 1 - \theta$ and $S = Q_1\theta + Q_2(1-\theta)$, then

$$q(x;\theta) = \begin{cases} Q_1/S & \text{if } x < \theta \\ Q_2/S & \text{otherwise.} \end{cases} \tag{D.4}$$

Using Equation 8.17, the gradient of the KL divergence w.r.t. θ is

$$\nabla_\theta D_{\mathrm{KL}}(p \,\|\, q;\theta) = \nabla_\theta \int_0^1 \begin{cases} p(x)\log(Q_1/S) & \text{if } x < \theta \\ p(x)\log(Q_2/S) & \text{otherwise} \end{cases} \mathrm{d}x\,, \tag{D.5}$$

where—in contrast to our piecewise-quadratic coupling function—the gradient can *not* be moved into the integral (see Equation 8.18) due to the discontinuity of q at θ. This prevents us from expressing the stochastic gradient of Monte Carlo samples with respect to θ in closed form and therefore optimizing with it.

We further investigate ignoring this limitation and performing the simplification of Equation 8.18 regardlessly, resulting in

$$\nabla_\theta D_{\mathrm{KL}}(p \,\|\, q;\theta) \approx \mathbb{E}\left[\begin{cases} p(X)\left(1 - \frac{Q_2}{Q_1}\right) & \text{if } X < \theta \\ p(X)\left(\frac{Q_1}{Q_2} - 1\right) & \text{otherwise} \end{cases}\right], \tag{D.6}$$

which has the same sign *regardless of the value of* θ, resulting in divergent behavior.

A similarly undesirable (albeit different) behavior emerges when normalizing q in a slightly different way by interpreting Q as probability masses rather than unnormalized densities:

$$q(x;\theta) = \begin{cases} Q_1/\theta & \text{if } x < \theta \\ Q_2/(1-\theta) & \text{otherwise.} \end{cases} \tag{D.7}$$

The KL divergence gradient is then

$$\begin{aligned} \nabla_\theta D_{\mathrm{KL}}(p \,\|\, q;\theta) &\approx \int_0^1 \begin{cases} p(x)/\theta & \text{if } x < \theta \\ p(x)/(\theta-1) & \text{otherwise,} \end{cases} \mathrm{d}x \\ &= \frac{1}{\theta}\int_0^\theta p(x)\,\mathrm{d}x - \frac{1}{1-\theta}\int_\theta^1 p(x)\,\mathrm{d}x\,. \end{aligned} \tag{D.8}$$

To illustrate the flawed nature of this gradient, consider the simple scenario of $p(x) = 1$, in which the RHS *always* equals to zero, suggesting *any* θ being a local minimum. However, θ clearly influences $D_{\mathrm{KL}}(p \,\|\, q;\theta)$ in this example, and therefore can not be optimal everywhere. Empirical investigations with other shapes of p, e.g. the examples from Figure 8.4, also suffer from a broken optimization and do not converge to a meaningful result.

While we only discuss a simplified setting here, the simplification in Equation 8.18 is also invalid in the *general* case of piecewise-linear coupling functions, likewise leading to a broken optimization.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Erik Alerstam, Stefan Andersson-Engels, and Tomas Svensson. White monte carlo for time-resolved photon migration. *Journal of Biomedical Optics*, 13:13 – 13 – 10, 2008.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv:1606.04474*, June 2016.

Peter Apian-Bennewitz. Bme bsdf, brdf data, 2013. `http://www.pab.eu/gonio-photometer/demodata/bme/` Accessed: 2019-02-23.

Mahdi M. Bagher, Cyril Soler, and Nicolas Holzschuch. Accurate fitting of measured reflectances using a shifted gamma micro-facet distribution. *Computer Graphics Forum*, 31(4):1509–1518, June 2012.

Mahdi M. Bagher, John Snyder, and Derek Nowrouzezahrai. A non-parametric factor microfacet model for isotropic brdfs. *ACM Trans. Graph.*, 35(5):159:1–159:16, July 2016.

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4), 2017.

David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of The 34th International Conference on Machine Learning*, 2017.

Laurent Belcour and Pascal Barla. A practical extension to microfacet theory for the modeling of varying iridescence. *ACM Trans. Graph.*, 36(4):65:1–65:14, July 2017.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Benedikt Bitterli, Jan Novák, and Wojciech Jarosz. Portal-masked environment map sampling. *Computer Graphics Forum*, 34(4), June 2015.

Benedikt Bitterli, Wenzel Jakob, Jan Novák, and Wojciech Jarosz. Reversible jump metropolis light transport using inverse mappings. *ACM Trans. Graph.*, 37(1):1:1–1:12, October 2017.

Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. A radiative transfer framework for non-exponential media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 37(6):225:1–225:17, nov 2018.

Benedikt Bitterli. Rendering resources, 2016. `https://benedikt-bitterli.me/resources`.

James F. Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics*, 11(2):192–198, July 1977.

Tamy Boubekeur, Wolfgang Heidrich, Xavier Granier, and Christophe Schlick. Volume-Surface Trees. *Computer Graphics Forum*, 25(3):399–406, 2006.

G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, July 1958.

B. C. Budge, J. C. Anderson, and K. I. Joy. Caustic Forecasting: Unbiased Estimation of Caustic Lighting for Global Illumination. *Computer Graphics Forum*, 2008.

Andrew Burnes. NVIDIA RTX technology: Making real-time ray tracing a reality for games. `https://www.geforce.com/whats-new/articles/nvidia-rtx-real-time-game-ray-tracing`, March 2018. Accessed: 2019-02-23.

Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. *Computer Graphics*, 21(4):273–281, August 1987.

Olivier Cappé, Arnaud Guillin, Jean-Michel Marin, and Christian P. Robert. Population monte carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929, 2004.

Olivier Cappé, Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian P. Robert. Adaptive importance sampling in general mixture classes. *Statistics and Computing*, 18(4):447–459, December 2008.

Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017.

S. Chandrasekar. *Radiative Transfer*. Dover Publications, 1960.

Yutian Chen, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 748–756, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Matt Jen-Yuan Chiang, Peter Kutz, and Brent Burley. Practical and controllable subsurface scattering for production path tracing. In *ACM SIGGRAPH Talks*, pages 49:1–49:2, New York, NY, USA, 2016. ACM.

Nicolas Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539–552, August 2002.

Per H. Christensen and Wojciech Jarosz. The path to path-traced movies. *Found. Trends. Comput. Graph. Vis.*, 10(2):103–175, October 2016.

Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Trans. Graph.*, 36(4), 2017.

David Cline, Justin Talbot, and Parris Egbert. Energy redistribution path tracing. *ACM Trans. Graph.*, 24(3):1186–1195, July 2005.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

Ken Dahm and Alexander Keller. Machine learning and integral equations. *CoRR*, abs/1712.06115, 2017.

Ken Dahm and Alexander Keller. Learning light transport the reinforced way. In Art B. Owen and Peter W. Glynn, editors, *Monte Carlo and Quasi-Monte Carlo Methods*, pages 181–195. Springer International Publishing, 2018.

Pierre Del Moral. Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2:555–580, March 1996.

Eugene d'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. *ACM Trans. Graph.*, 30(4):56:1–56:14, July 2011.

Eugene d'Eon. Rigorous asymptotic and moment-preserving diffusion approximations for generalized linear boltzmann transport in arbitrary dimension. *Transport Theory and Statistical Physics*, 42(6-7):237–297, 2013.

Eugene d'Eon. A reciprocal formulation of non-exponential radiative transfer. 1: Sketch and motivation. *arXiv preprint arXiv:1803.03259*, 2018.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Jonathan Dupuy and Wenzel Jakob. An adaptive parameterization for efficient material acquisition and rendering. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 37(6):274:1–274:18, November 2018.

Jonathan Dupuy, Eric Heitz, and Eugene d'Eon. Additional progress towards the unification of microfacet and microflake theories. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*. The Eurographics Association, 2016.

Luca Fascione, Johannes Hanika, Rob Pieké, Ryusuke Villemin, Christophe Hery, Manuel Gamito, Luke Emrose, and André Mazzone. Path tracing in production. In *ACM SIGGRAPH Courses*, pages 15:1–15:79, New York, NY, USA, 2018. ACM.

Jiří Filip and Radomír Vávra. Template-based sampling of anisotropic brdfs. *Computer Graphics Forum*, 33(7):91–99, 2014.

Jiří Filip, Radomír Vávra, and Michal Havlíček. Effective acquisition of dense anisotropic BRDF. In *Proceedings of the 22th International Conference on Pattern Recognition, ICPR 2014*, pages 2047–2052, August 2014.

Václav Gassenbauer, Jaroslav Křivánek, and Kadi Bouatouch. Spatial directional radiance caching. *Computer Graphics Forum*, 28(4):1189–1198, 2009. Eurographics Symposium on rendering, EGSR '09.

Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192:1–192:10, November 2012.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

Abhijeet Ghosh, Shruthi Achutha, Wolfgang Heidrich, and Matthew O'Toole. Brdf acquisition with basis illumination. pages 1–8, November 2007.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR.org, 2010.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, January 1984.

Jerry Jinfeng Guo, Pablo Bauszat, Jacco Bikker, and Elmar Eisemann. Primary Sample Space Path Guiding. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*. The Eurographics Association, 2018.

Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Trans. Graph.*, 28(5):141:1–141:8, December 2009.

Toshiya Hachisuka and Henrik Wann Jensen. Robust adaptive photon tracing using photon path visibility. *ACM Trans. Graph.*, 30(5):114:1–114:11, October 2011.

Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. Multiplexed metropolis light transport. *ACM Trans. Graph.*, 33(4):100:1–100:10, July 2014.

Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691v3*, December 2017.

David Money Harris and Sarah L. Harris. 3.4.2 - state encodings. In *Digital Design and Computer Architecture*, pages 129–131. Morgan Kaufmann, Boston, second edition, 2013.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The sggx microflake distribution. *ACM Trans. Graph.*, 34(4):48:1–48:11, July 2015.

Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. Real-time polygonal-light shading with linearly transformed cosines. *ACM Trans. Graph.*, 35(4):41:1–41:8, July 2016.

Eric Heitz, Johannes Hanika, Eugene d'Eon, and Carsten Dachsbacher. Multiple-scattering microfacet bsdfs with the smith model. *ACM Trans. Graph.*, 35(4):58:1–58:14, July 2016.

Eric Heitz. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques (JCGT)*, 3(2):48–107, June 2014.

L. G. Henyey and J. L. Greenstein. Diffuse radiation in the galaxy. *The Astrophysical Journal*, 93:70–83, 1941.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum*, 2016.

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik P. A. Lensch. A Unified Manifold Framework for Efficient BRDF Sampling based on Parametric Mixture Models. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*. The Eurographics Association, 2018.

Heinrich Hey and Werner Purgathofer. Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics*, SCCG '02, pages 107–114, New York, NY, USA, 2002. ACM.

Nicolas Holzschuch and Romain Pacanowski. A two-scale microfacet reflectance model combining reflection and diffraction. *ACM Trans. Graph.*, 36(4):66:1–66:12, July 2017.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, March 1991.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron C. Courville. Neural autoregressive flows. *CoRR*, abs/1804.00779, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Wenzel Jakob and Steve Marschner. Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Trans. Graph.*, 31(4):58:1–58:13, July 2012.

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.*, 29(4):53:1–53:13, July 2010.

Wenzel Jakob, Eugene d'Eon, Otto Jakob, and Steve Marschner. A comprehensive framework for rendering layered materials. *ACM Trans. Graph.*, 33(4):118:1–118:14, July 2014.

Wenzel Jakob, Miloš Hašan, Ling-Qi Yan, Jason Lawrence, Ravi Ramamoorthi, and Steve Marschner. Discrete stochastic microfacet models. *ACM Trans. Graph.*, 33(4):115:1–115:10, July 2014.

Wenzel Jakob. Mitsuba renderer, 2010. `http://mitsuba-renderer.org`.

Adrian Jarabo, Julio Marco, Adolfo Muñoz, Raul Buisan, Wojciech Jarosz, and Diego Gutierrez. A framework for transient rendering. *ACM Trans. Graph.*, 33(6):177:1–177:10, November 2014.

Wojciech Jarosz, Nathan A. Carr, and Henrik Wann Jensen. Importance sampling spherical harmonics. *Computer Graphics Forum*, 28(2):577–586, April 2009.

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proc. SIGGRAPH*, pages 511–518, 2001.

Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Rendering Techniques*, pages 326–335, Vienna, 1995. Springer Vienna.

Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In *Computer Graphics*, pages 271–280, July 1989.

James T. Kajiya. The rendering equation. *Computer Graphics*, 20:143–150, 1986.

Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans. Graph. (Proc. of Siggraph Asia)*, 36(6), November 2017.

Csaba Kelemen and László Szirmay-Kalos. A microfacet based coupled specular-matte brdf model with importance sampling. *Eurographics Short Presentations*, 25, April 2001.

Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the Metropolis light transport algorithm. *Computer Graphics Forum*, 21(3):531–540, 2002.

A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols. The path tracing revolution in the movie industry. In *ACM SIGGRAPH Courses*, pages 24:1–24:7, New York, NY, USA, 2015. ACM.

Alexander Keller. Quasi-monte carlo image synthesis in a nutshell. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 213–249, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *CoRR*, July 2018.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

Shinya Kitaoka, Yoshifumi Kitamura, and Fumio Kishino. Replica exchange light transport. *Computer Graphics Forum*, 28(8):2330–2342, December 2009.

Krzysztof C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical Programming*, 90(1):1–25, March 2001.

Jan J. Koenderink, Andrea J. van Doorn, and Marigo Stavridi. Bidirectional reflection distribution function expressed in terms of surface scattering modes. ECCV '96, pages 28–39, Berlin, Heidelberg, 1996. Springer-Verlag.

Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Compugraphics '93*, pages 145–153, 1993.

Eric P. Lafortune and Yves D. Willems. A 5d tree to reduce the variance of monte carlo ray tracing. In *Rendering Techniques '95 (Proc. of the 6th Eurographics Workshop on Rendering)*, pages 11–20, 1995.

Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In *Proc. EGWR*, pages 91–100, 1996.

Philip Laven. Mieplot, 2017. http://www.philiplaven.com/mieplot.htm.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

Richard Lee and Carol O'Sullivan. Accelerated light propagation through participating media. In *Proc. Eurographics / Ieee VGTC Conference on Volume Graphics*, pages 17–23, 2007.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning image restoration without clean data. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2965–2974, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR.

Hongsong Li, Fabio Pellacini, and Kenneth E. Torrance. A hybrid Monte Carlo method for accurate and efficient subsurface scattering. In *Proc. EGSR*, pages 283–290, 2005.

Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. Anisotropic gaussian mutations for metropolis light transport through hessian-hamiltonian dynamics. *ACM Trans. Graph.*, 34(6):209:1–209:13, October 2015.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

Jun S. Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 3730–3738, Washington, DC, USA, 2015. IEEE Computer Society.

Ludwig Lorenz. Lysbevægelsen i og uden for en af plane lysbølger belyst kugle. In *Det Kongelige Danske Videnskabernes Selskabs Skrifter (trykt utg.): Naturvidenskabelig og Mathematisk Afdeling*, 1890.

Joakim Löw, Joel Kronander, Anders Ynnerman, and Jonas Unger. Brdf models for accurate and efficient rendering of glossy surfaces. *ACM Trans. Graph.*, 31(1):9:1–9:14, February 2012.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6232–6240, USA, 2017. Curran Associates Inc.

Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, and Donald P. Greenberg. Image-based brdf measurement including human skin. In *Eurographics Workshop on Rendering*. The Eurographics Association, 1999.

Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, and Kenneth E. Torrance. Image-based bidirectional reflectance distribution function measurement. *Appl. Opt.*, 39(16):2592–2600, June 2000.

Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. *ACM Trans. Graph.*, 22(3):780–791, July 2003.

Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

Phillip R. Mattison, Mark S. Dombrowski, James M. Lorenz, Keith J. Davis, Harley C. Mann, Philip Johnson, and Bryan Foos. Handheld directional reflectometer: an angular imaging device to measure brdf and hdr in real time. *Proceedings of SPIE - The International Society for Optical Engineering*, October 1998.

Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Trans. Graph.*, 22(3):759–769, July 2003.

Michael D. McCool and Peter K. Harwood. Probability trees. In *Proceedings of the Graphics Interface 1997 Conference, May 21-23, 1997, Kelowna, BC, Canada*, pages 37–46, May 1997.

Morgan McGuire. Computer graphics archive, August 2011. `http://graphics.cs.williams.edu/data`.

Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. Multi-scale modeling and rendering of granular materials. *ACM Trans. Graph.*, 34(4):49:1–49:13, July 2015.

Nicholas Metropolis and Stanisław Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

Nick Metropolis. The beginning of the monte carlo method. *Los Alamos Science Special Issue*, 15:125–130, 1987.

Gustav Mie. Beiträge zur optik trüber medien, speziell kolloidaler metallösungen. *Annalen der Physik*, 330:377–445, 1908.

Jonathan T. Moon, Bruce Walter, and Stephen R. Marschner. Rendering discrete random media using precomputed scattering solutions. In *Proc. EGSR*, pages 231–242, June 2007.

Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. Efficient rendering of heterogenous polydisperse granular media. *ACM Trans. Graph.*, 35(6):168:1–168:14, 2016.

Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. *Computer Graphics Forum*, 36(4):91–100, June 2017.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *CoRR*, abs/1808.03856, 2018.

Thomas Müller. Modeling and rendering heterogeneous granular materials. Master's thesis, March 2016.

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. Deep shading: Convolutional neural networks for screen-space shading. 36(4), 2017.

Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental analysis of brdf models. In *Proceedings of the Eurographics Symposium on Rendering*, pages 117–226. Eurographics Association, 2005.

F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometrical considerations and nomenclature for reflectance. In *Radiometry*, pages 94–145. 1992.

Jan Novák and Carsten Dachsbacher. Rasterized bounding volume hierarchies. *Computer Graphics Forum*, 31(2):403–412, 2012.

Jan Novák, Iliyan Georgiev, Johannes Hanika, Jaroslav Křivánek, and Wojciech Jarosz. Monte carlo methods for physically based volume rendering. In *ACM SIGGRAPH Courses*, August 2018.

Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

Michael Oren and Shree K. Nayar. Generalization of lambert's reflectance model. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 239–246, New York, NY, USA, 1994. ACM.

Hisanari Otsu, Anton S. Kaplanyan, Johannes Hanika, Carsten Dachsbacher, and Toshiya Hachisuka. Fusing state spaces for markov chain monte carlo rendering. *ACM Trans. Graph.*, 36(4):74:1–74:10, July 2017.

Jacopo Pantaleoni and Eric Heitz. Notes on optimal approximations for importance sampling. *CoRR*, abs/1707.08358, 2017.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

Vincent Pegoraro. *Handbook of Digital Image Synthesis: Scientific Foundations of Rendering*. CRC Press, 2016.

Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ravi Ramamoorthi. Precomputation-based rendering. *Found. Trends. Comput. Graph. Vis.*, 3(4):281–369, April 2009.

Lord Rayleigh. Xxxiv. on the transmission of light through an atmosphere containing small particles in suspension, and on the origin of the blue of the sky. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 47(287):375–384, 1899.

Ju Ren and Jianlin Zhao. Measurement of a bidirectional reflectance distribution and system achievement based on a hemi-parabolic mirror. *Opt. Lett.*, 35(9):1458–1460, May 2010.

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Trans. Graph.*, 32(4):130:1–130:12, July 2013.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

Iman Sadeghi, Adolfo Muñoz, Philip Laven, Wojciech Jarosz, Francisco Seron, Diego Gutierrez, and Henrik Wann Jensen. Physically-based simulation of rainbows. *ACM Trans. Graph.*, 31(1):3:1–3:12, February 2012.

Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. Proc. SIGGRAPH, pages 161–172, New York, NY, USA, 1995. ACM.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. *arXiv preprint arXiv:1603.05201*, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jos Stam. Multiple scattering as a diffusion process. *Proc. EGWR*, pages 41–50, 1995.

J. Steinhurst and A. Lastra. Global Importance Sampling of Glossy Surfaces Using the Photon Map. *Symposium on Interactive Ray Tracing*, 0:133–138, 2006.

Justin F. Talbot, David Cline, and Parris Egbert. Importance resampling for global illumination. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, EGSR '05, pages 139–146, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of the Optical Society of America*, 57(9):1105–1112, September 1967.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756, 2016.

Eric Veach and Leonidas J. Guibas. Bidirectional estimators for light transport. In *EG Rendering Workshop*, 1994.

Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proc. SIGGRAPH*, pages 419–428, 1995.

Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *97*, Annual Conference Series, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

Eric Veach. *Robust Monte Carlo methods for light transport simulation*. PhD thesis, Stanford, CA, USA, December 1997.

Petr Vévoda, Ivo Kondapaneni, and Jaroslav Křivánek. Bayesian online regression for adaptive direct illumination sampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4), 2018.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4), 2018.

Jiří Vorba and Jaroslav Křivánek. Adjoint-driven russian roulette and splitting in light transport simulation. *ACM Trans. Graph.*, 35(4), July 2016.

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. Online learning of parametric mixture models for light transport simulation. *ACM Trans. Graph.*, 33(4), August 2014.

Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. Path tracing in production. In *ACM SIGGRAPH Courses*, pages 18:1–18:77, New York, NY, USA, 2019. ACM.

Martin Šik, Hisanari Otsu, Toshiya Hachisuka, and Jaroslav Křivánek. Robust light transport simulation via metropolised bidirectional estimators. *ACM Trans. Graph.*, 35(6):245:1–245:12, November 2016.

Walt Disney Animation Studios. Moana island scene, 2016. `https://www.technology.disneyanimation.com/islandscene` Accessed: 2019-04-04.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proc. EGSR*, pages 195–206, Aire-la-Ville, Switzerland, 2007. Eurographics Association.

Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. All-frequency rendering of dynamic, spatially-varying reflectance. *ACM Trans. Graph.*, 28(5):133:1–133:10, December 2009.

Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, June 1988.

Gregory J. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics*, 26(2):265–272, July 1992.

Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. In *Computer Graphics*, pages 255–264, July 1992.

David White, Peter Saunders, Stuart J. Bonsey, John van de Ven, and Hamish Edgar. Reflectometer for measuring the bidirectional reflectance of rough surfaces. *Applied optics*, 37:3450–4, July 1998.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. Anisotropic spherical gaussians. *ACM Trans. Graph.*, 32(6):209:1–209:11, November 2013.

Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph.*, 33(4):116:1–116:9, July 2014.

Shuang Zhao, Miloš Hašan, Ravi Ramamoorthi, and Kavita Bala. Modular flux transfer: efficient rendering of high-resolution volumes with repeated structures. *ACM Trans. Graph.*, 32(4):131:1–131:12, July 2013.

Quan Zheng and Matthias Zwicker. Learning to importance sample in primary sample space. *arXiv:1808.07840*, September 2018.