Universität des Saarlandes

How to Train Your Renderer: **Optimized Methods for Learning Path Distributions in Monte Carlo Light Transport**

Vorgelegt von

Alexander Rath

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften (Dr.-Ing.) der Fakultät für Mathematik und Informatik der Universität des Saarlandes

Saarbrücken, 2024

Tag des Kolloquiums:
Dienstag, 06. Mai 2025

Dekan:
Prof. Dr. Roland Speicher

Vorsitzende:
Prof. Dr. Verena Wolf

Berichterstatter:
Prof. Dr. Philipp Slusallek
Prof. Dr. Hans-Peter Seidel
Prof. Dr. Tamy Boubekeur

Akademischer Mitarbeiter:
Dr. Pascal Grittmann

Abstract

Light transport simulation allows us to preview architectural marvels before they break ground, practice complex surgeries without a living subject, and explore alien worlds from the comfort of our homes. Fueled by the steady advancements in computer hardware, rendering virtual scenes is more accessible than ever, and is met by an unprecedented demand for such content. Light interacts with our world in various intricate ways, hence the challenge in realistic rendering lies in tracing all the possible paths that light could take within a given virtual scene. Contemporary approaches predominantly rely on Monte Carlo integration, for which countless sampling procedures have been proposed to handle certain families of effects robustly. Handling all effects holistically through specialized sampling routines, however, remains an unsolved problem.

A promising alternative is to use learning techniques that automatically adapt to the effects present in the scene. However, such approaches require many complex design choices to be made, which existing works commonly resort to heuristics for. In this work, we investigate what constitutes effective learning algorithms for rendering – from data representation and the quantities to be learned, to the fitting process itself. By strategically optimizing these components for desirable goals, such as overall render efficiency, we demonstrate significant improvements over existing approaches.

Kurzfassung

Die Simulation von Licht ermöglicht es uns, architektonische Meisterwerke noch vor deren Errichtung zu visualisieren, komplexe chirurgische Eingriffe ohne lebendes Subjekt zu üben, und immersive Fantasiewelten bequem vom Sofa aus zu erleben. Angefeuert durch den stetigen technologischen Forschritt sind solche Anwendungen nun weiter verbreitet und gefragter denn je. Licht interagiert auf vielschichtige Weise mit unserer Welt – eine Herausforderung, die bei der realistischen Simulation virtueller Szenen vollständig berücksichtigt werden muss. Moderne Algorithmen zur Bildsynthese bauen nahezu ausschließlich auf Monte Carlo Integration auf, wofür bereits zahlreiche Sampling Routinen publiziert wurden, die einzelne Lichteffekte robust händeln können. Eine ganzheitliche Simulation aller möglichen Effekte bleibt jedoch weiterhin ein ungelöstes Problem.

Eine vielversprechende Alternative ist der Einsatz von lernbasierten Verfahren, die das Sampling adaptiv an die spezifischen Eigenschaften der Szene anpassen. Solche Verfahren erfordern jedoch viele komplexe Entwurfsentscheidungen, für welche existierende Werke zumeist auf heuristische Annahmen zurückgreifen. In dieser Arbeit analysieren wir, was erfolgreiche Lernverfahren auszeichnet – von der Datenrepräsentation, über die zu lernenden Größen, bis hin zum Fitting der Modelle selbst. Mittels strategischer Optimierung der einzelnen Komponenten für relevante Metriken, wie beispielsweise die Konvergenzgeschwindigkeit der Simulation, demonstrieren wir signifikante Verbesserungen über existierenden Ansätzen.

Abstrait

Simuler le transport de la lumière permet d'admirer des merveilles architecturales avant leur construction, de s'entraîner aux chirurgies les plus complexes sans risques, et de visiter des mondes étrangers depuis le confort de nos maisons. Alimenté par la constante croissance des performances matérielles, le rendu de scènes virtuelles est plus accessible que jamais, et rencontre une demande sans précédent. La lumière interagit avec notre monde d'une myriades de façons, dès lors le défi du rendu photoréaliste consiste à considérer tous les chemins possibles que la lumière peut prendre pour un lieu virtuel donné. Les approches contemporaines sont basées de façon prédominante sur l'intégration Monte Carlo, pour laquelle d'innombrables approches d'échantillonnage ont été proposées pour capturer certaines familles d'effet de façon robuste. Prendre en charge chaque effet de façon holistique via des routines d'échantillonnage spécialisées reste cependant un problème non-résolu.

Une prometteuse alternative est d'utiliser des techniques apprentissantes qui s'adaptent automatiquement aux effets présents dans la scène. Néanmoins, de telles approches nécessitent de prendre de nombreux choix complexes, pour lesquelles les travaux existants on eu recourt à des heuristiques. Dans cette œuvre, nous étudions ce qui constitue des méthodes apprentissantes efficaces pour le rendu – depuis la représentation des données et des paramètres à apprendre, jusqu'au processus d'apprentissage lui-même. En optimisant stratégiquement ces composants pour les objectifs désirés, comme l'efficacité du rendu, nous démontrons des améliorations significatives par rapport aux approches existantes.

Acknowledgements

First and foremost, I would like to express my gratitude to my advisor, Philipp Slusallek, who sparked my fascination with computer graphics and gave me the opportunity to embark on this incredible journey. His unwavering support and the countless opportunities he provides at our chair have created an inspiring and nurturing environment to work and grow. I am also immensely thankful to Pascal Grittmann and Sebastian Herholz, who taught me the essential skills of our field – from scientific evaluations and paper writing to delivering impactful presentations. Their mentorship and *guidance* have been invaluable throughout this endeavor, and their feedback has significantly elevated the quality of my work.

Speaking of the chair, there is few places buzzing with such a wide variety of inspirations. Anything ranging from artificial intelligence, over compilers, to manifacturing autonomous whiteboards – there is always an expert at the chair. I will forever remember our insightful discussions and (not always work-related) projects. Thank you Pascal, Ömercan, Philippe, Qin, Misa, Hugo, Matthias, Puya, Richard, Manuela, and Stefan, for creating such a wonderful and fun environment to work and thrive in. But it is not just the people that work here that I would like to thank. Our students, be it in lectures, seminars, or thesises, have also been a constant source of inspiration. Their unusual ideas and unique perspectives have not only enriched my knowledge but also filled gaps I didn't know existed.

Over the course of my PhD, I've been fortunate to collaborate with brilliant minds across diverse fields and locations. It was great fun simulating Radar with Sandro and Thomas from Continental, which has also taught me many new surprising things about light transport. And working on real-time rendering with Sebastian and Tobias at Intel has taught me a lot about hardware architectures, on top of the many other joyful discussions we have had. It was also a great pleasure to get to work with Ilyian, Tamy, and Élie from Adobe, whose infectious enthusiam and creative discussions have been nothing but bliss. And finally, I will always treasure the amazing time I had in Switzerland – thank you Marco, Tiziano, Marios, and all the other amazing people I have met at Disney Research in Zürich.

I am also profoundly grateful to the people working behind the scenes who make all of this possible. A big thank you to the secretaries and managers – Sabine, Léa, Nadja, Jocelyn, and Laura – for your indispensable support. And, of course, gratitude is owed to the authors of test scenes, the literal people *behind the scenes*.

Last, but not least, I would like to thank my family for their unending support and encouragement. Most importantly, my parents Klaus and Erika for my fostering my fascination for computer science from an early age on – and driving more than a million kilometers between school and university while I was yet too young to drive. And of course I would also like to thank my sister Nathalie for fighting off the bullies in primary school! And to my wonderful wife, Anika: You have been my anchor, my source of motivation, and my greatest joy throughout this journey – I am endlessly grateful for your love and encouragement.

Contents

List of Figures

List of Tables

List of Symbols

Ω	Unit sphere. *(p. 7)*
ω_i	Incident direction. *(p. 7)*
ω_o	Outgoing direction. *(p. 8)*
$L_\mathrm{i}(\omega_\mathrm{i}, x)$	Incident Radiance. *(p. 7)*
$L_\mathrm{o}(\omega_\mathrm{o}, x)$	Outgoing Radiance. *(p. 7)*
$L_\mathrm{e}(\omega_\mathrm{o}, x)$	Emitted Radiance. *(p. 8)*
$\mathcal{P}$	Path space. *(p. 8)*
$\bar{\mathbf{x}}$	A path consisting of $n+1$ vertices $\mathbf{x}_0, \ldots, \mathbf{x}_n$. *(p. 8)*
$\bar{\mathbf{x}}_k$	The first $k+1$ vertices (prefix) of a path $\bar{\mathbf{x}}$. *(p. 31)*
$G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j)$	Geometry term. *(p. 9)*
px	A pixel of the rendered image. *(p. 7)*
I_px	Radiance arriving at pixel px. *(p. 7)*
$W_\mathrm{px}(x, \omega_\mathrm{i})$	Sensor response function. *(p. 7)*
$B(\omega_\mathrm{i}, x, \omega_\mathrm{o})$	Bidirectional scattering distribution function. *(p. 8)*
$\langle \cdot \rangle$	A primary estimator of an integral. *(p. 12)*
$\mathbb{E}\left[\cdot\right]$	Expected value of a random variable. *(p. 11)*
$\mathbb{V}\left[\cdot\right]$	Variance of a random variable. *(p. 11)*
$\mathbb{C}\left[\cdot\right]$	Expected computational cost of a random variable. *(p. 14)*
$\epsilon\left[\cdot\right]$	Efficiency of an estimator. *(p. 14)*

1 Introduction

"Show, don't tell." This principle underscores the significance of vision as the information highway to the human mind, which aids us in perceiving the world at a pace unparalleled by our other senses. It is why we gravitate towards visual storytelling, from cave paintings to movies, to enigmatic pictograms in furniture assembly instructions. Visual media allow us to efficiently convey complex ideas and emotions, and share moving experiences with others.

Light transport simulation is the science of faithfully simulating photographs of objects that may not physically exist. For example, it allows us to preview architectural marvels before breaking ground, practice complex surgeries without a living subject, and explore alien worlds from the comfort of our homes. And thanks to the steady advancements in computer hardware, rendering virtual scenes is more accessible than ever, and is met by an unprecedented demand. Even real-time applications such as computer games are beginning to embrace physically-based lighting simulation to pursue new levels of visual fidelity.

But simulating light is no easy feat, as there are various intricate ways in which light interacts with matter before it reaches our eyes. Surfaces reflect light depending on their chemical composition and microscopic structure, resulting in a colorful variety of effects that range from soft, diffuse glow, to sharp, mirror-like reflections or delicate glints. Additionally, light can penetrate some materials, scattering internally before emerging elsewhere, and when passing through transparent objects like glass, it can create captivating patterns where it lands. And all of this happens not only once, but inconceivably many times successively along the paths that light takes. The complexity of accurately simulating light lies in the need to account for every potential path it might take, especially when numerous of these intricate interactions happen along the path.

Contemporary methods for light transport simulation predominantly employ Monte Carlo integration, which randomly samples the set of possible light paths to reconstruct the image. Path tracing, the most prevalent technique, constructs paths incrementally through random walks that initiate at the camera and continue in random directions whenever objects are encountered along a ray. Due to its stochastic nature, the effectiveness of Monte Carlo methods hinges on how closely the distribution of sampled paths matches the true distribution of light. A significant discrepancy necessitates a higher number of samples to arrive at the same amount of noise in the resulting image, thereby increasing rendering times.

Specialized techniques exist to systematically sample paths for various kinds of light phenomena. For example, next event estimation can reliably explore direct illumination from small light sources by taking shortcuts to them. But once there is a transparent object in front of the light, a more sophisticated and harder-to-implement technique is required to sample the light thoroughly. Implementing all of these specialized techniques is a tedious undertaking, and falls short as soon as unforeseen effects emerge that are not handled by existing techniques. A promising, more general alternative is to build adaptive path distributions for these challenging effects by employing learning mechanisms during rendering.

Two important applications of using learning to shape path distributions are path guiding as well as Russian roulette and splitting. The former alters the distribution of directions in which to walk by predicting which direction light comes from, while the latter splits important paths into multiple samples and terminates unimportant paths to balance computational cost and stochastic noise. The effectiveness of these strategies hinges on a careful orchestration of design parameters, such as the knowledge representations employed, the quantities attempted to be learned, and the training scheme through which the knowledge is obtained. Ideally, these design parameters are jointly optimized towards an explicit goal, but their complex interplay makes this difficult. Hence, common design choices often rely on hand-tuned heuristics, which have suboptimal performance and only weak guarantees for robustness.

The goal of this work is to make learned path distributions more robust and efficient by moving away from reliance on heuristics towards a methodology grounded in explicit optimization goals. Within the prevalent path tracing framework, we analyze the two key techniques to shaping path distributions, path guiding as well as Russian roulette and splitting, and discuss the shortcomings of common heuristics used in learning. To address these shortcomings, we investigate relevant goals, such as minimizing noise or robustly handling challenging families of complex light phenomena, and demonstrate how individual components of learning algorithms can be optimized to achieve these goals.

1.1 Contributions

This thesis proposes three novel approaches to learning path distributions that are explicitly optimized towards relevant goals, such as noise reduction or robust handling of certain effects. While our work focuses on the popular path tracing framework, many of the insights can be applied to other rendering algorithms.

Variance-Aware Path Guiding Previous path guiding methods make sampling decisions assuming that all other decisions have been sampled perfectly, which due to limits of representations and the training process is not generally the case. With the goal of minimizing the noise of the rendered image, we derive an optimized target function that replaces the heuristic on what quantities should be learned. We additionally apply our technique to the related problems of optimizing mixing ratios of different sampling densities, and sampling light sources for next event estimation. This work has been previously published in our SIGGRAPH 2020 paper [Rath et al. 2020]. I was the main author of that paper, contributing the original idea, the implementation and evaluation, and much of the text in the paper. Parts that overlap with my master's thesis [Rath 2019] are covered as previous work in this thesis.

Focal Path Guiding Existing path guiding approaches focus mostly on directional representations to steer path sampling, with some more recent works also exploring distributions over surfaces. We identify an important family of effects that is challenging to capture using these representations, and classify the situations in which they occur. Some of these effects cannot be handled robustly by any existing guiding method, and there exists no specialized technique to sample them explicitly. With the goal of reliably sampling this family of phenomena, we derive a novel representation that directly exploits its defining properties. This work has been previously published in our SIGGRAPH 2023 paper [Rath et al. 2023]. I was the main author of that paper, contributing the original idea, the implementation and some evaluation, and the majority of text of the paper.

Efficiency-Aware Russian Roulette and Splitting The goal of Russian Roulette and splitting is to maximize the efficiency of the lighting simulation. Performing these techniques optimally poses a challenging optimization problem, which existing works have largely avoided through approximations and simplifications. We derive a simple training scheme, based on the theory of fixed-point functions, that avoids most of these simplifications and provably converges to the optimal solution. In principle, our theory can be applied to any random walk Monte Carlo method, and achieves consistent speed-ups over the previous state of the art. The work has been previously published in our SIGGRAPH 2022 paper [Rath et al. 2022a]. I was the main author of that paper, contributing the original idea, the implementation and evaluation, and much of the text in the paper.

Source code Implementations of all our methods as used in the papers are publicly available on GitHub, along with interactive playgrounds and additional demonstrations:

- Variance-Aware Guiding: https://github.com/iRath96/variance-aware-path-guiding
- Efficiency-Aware RRS: https://github.com/iRath96/ears
- Focal Path Guiding: https://github.com/iRath96/focal-guiding

The respective chapters that detail our contributions re-use most of the text and figures from their respective original publications.

1.2 Outline

Following this introduction, Chapter 2 begins by establishing the necessary theoretical background and conventions used throughout the thesis, with a particular focus on light transport and Monte Carlo integration.

Chapter 3 starts with an overview of the various ways how Monte Carlo integration can be used to solve light transport, and then sets the stage for our approaches by reviewing how previous works employ learning to shape path distributions in path tracing.

The following three chapters present and discuss our contributions, starting with our path guiding work in Chapter 4 ("Variance-Aware Path Guiding") and Chapter 5 ("Focal Path Guiding"), and followed by "Efficiency-Aware Russian Roulette and Splitting" in Chapter 6.

The thesis is then concluded by Chapter 7, which summarizes our findings and contributions.

2 Background

In this chapter, we provide the necessary theoretical background and conventions that the remainder of the thesis builds upon. We limit our discussion to a high-level overview, for a more in-depth discussion we refer the reader to the excellent book by Pharr et al. [2016]. We begin by defining the problem setting (Section 2.1), followed by the underlying physics and models for light transport (Section 2.2), and conclude with an introduction to Monte Carlo integration, the most prevalent framework used to simulate light transport (Section 2.3).

2.1 Problem setting

Our goal is to produce photo-realistic renders from virtual scenes, which are composed of mathematical models of geometry, materials, cameras, and light sources. To determine the image captured by a virtual camera, we need to simulate how light propagates through and interacts with virtual environments. While sophisticated models, such as quantum electrodynamics, accurately describe the behavior of light down to microscopic scales, simulating these models at our scale of interest is prohibitive. We must therefore trade off some accuracy to reduce complexity.

Luckily (or perhaps sadly?), many of the more intricate ways light interacts with matter go unnoticed by us, since we are too large to notice them and our visual systems too limited. For instance, wave effects like *diffraction* (light bending around corners) are dominant only when objects are small compared to the wavelength, which for visible light is on the order of hundreds of nanometers. Similarly, *interference* (the property that light can cancel itself) is only important for light of long coherence lengths (e.g., lasers), which is not the case for most natural light sources. Our perception is also too slow to notice that light does not propagate instantly. While such effects play an important role outside the visible spectrum (for example in acoustic or radar simulations), we can safely ignore them in the following and rely on *geometrical optics* instead, as is commonly done in light transport.

To keep our derivations simple (both for the reader and us), we ignore volumetric, spectral, and polarization[1] effects. While efficient light transport techniques exist to handle them, these extensions are mostly orthogonal to our work and therefore left as future work.

[1] Did you know that humans can learn to see polarization with the naked eye? [Temple et al. 2015]

2.2 Light transport

Predicting what a virtual scene looks like to a camera requires models of how light propagates through the scene and interacts with objects and the camera. In this section, we build towards the famous rendering equation. We begin with a review of physical units required to quantify light, discuss how light is measured by the camera, and then detail how it propagates and scatters through the environment. We conclude with a brief overview of the challenges that arise when computing these models.

Dynamic equilibrium When a light is turned on, it begins flooding the environment with *photons*, the particles that constitute light[2]. Not much later, seemingly instant to the human observer, the environment reaches a steady brightness: The light distribution has reached a dynamic equilibrium, in which the ongoing emission of photons is met by an equal absorption of photons by the environment. It is this steady state of light that we are interested in simulating, which reflects itself in the parametrization of our models as well as the units we will use to quantify light.

2.2.1 Physical units

Radiometry is the science of measuring electromagnetic radiation, such as visible light. As all models in the following are grounded in radiometry, we begin with a brief overview of the radiometric units that constitute these models which is then summarized in Fig. 2.1.

Radiant flux ϕ describes the energy per unit time (measured in *Watt*, W) that enters or exits a surface A as light from all directions Ω.

Irradiance E and **Radiosity** M describe the incoming (resp. outgoing) radiant flux per unit area (measured in *Watt per square meter*, $\mathrm{W} \cdot \mathrm{m}^{-2}$). When directionality is not important, for example for diffuse surfaces, this unit is commonly used to quantify light distributions.

Radiance L describes irradiance/radiosity per unit direction (measured in *Watt per square meter per steradian*, $\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}$). It quantifies light distributions with spatial and directional dependency and has convenient properties for simulation that we explore in the following.

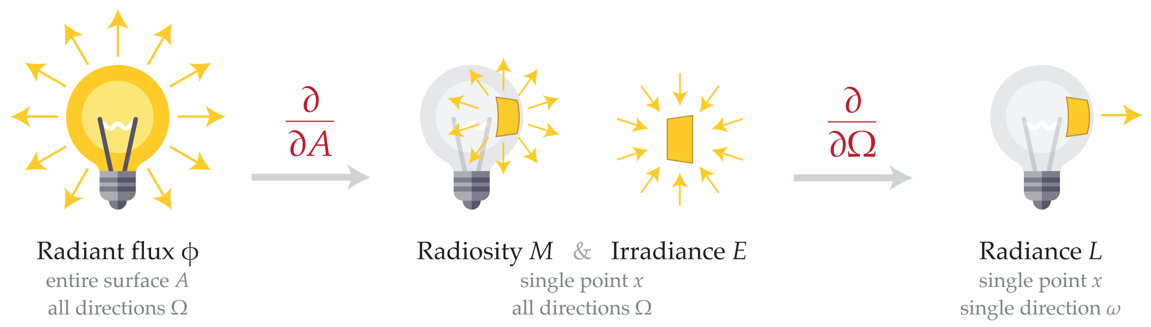
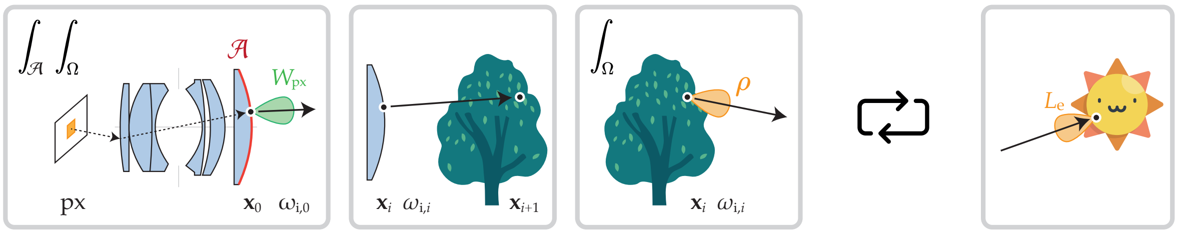


Figure 2.1: *An overview over radiometric units commonly used in light transport.*

[2] Velten et al. [2013] have captured the fascinating propagation of light with sophisticated slow-motion cameras, and Müller [2016] simulates the propagation of light directly in your browser.



(a) *Camera model* **(b)** *Propagation* **(c)** *Scattering* **(d)** *(repeat)* **(e)** *Emission*

Figure 2.2: *An overview of the components of light transport. (a) The* camera model *describes interactions of light within the camera. (b) Between scattering events, light propagates along straight lines. (c) On surfaces, light scatters in new directions. (d) Scattering can occur several times before (e) the path reaches a light source.*

2.2.2 Camera models

Cameras (and the human visual system) use lenses to project an image of the scene onto a photo-sensitive surface. In our eyes, this surface is the retina, which is covered by millions of photoreceptor cells that relay the brightness and color of incident light to our brain. In analog cameras, the photo-sensitive surface is the film, which can be developed into photographs through chemical processes. Our focus lies on simulating digital cameras, in which electronic sensors use rectangular grids of photodiodes to record digital images. In this section, we introduce the mathematical model we will use to simulate such cameras.

The input of a camera is a distribution of incoming light $L_{\text{i}}(\omega_{\text{i}}, x)$, which after passing through the lens and sensor electronics, results in an output image I. The image is a rectangular grid of *pixels* px (picture elements), for which we want to quantify the amount of light I_{px}

$$I_{\text{px}} = \int_{\mathcal{A}} \int_{\Omega} W_{\text{px}}(x, \omega_{\text{i}})\, L_{\text{i}}(\omega_{\text{i}}, x)\, \mathrm{d}\omega_{\text{i}}\, \mathrm{d}x. \tag{2.1}$$

Here, the *sensor response function* $W_{\text{px}}(x, \omega_{\text{i}})$ models the behavior of the lens, describing how sensitive a pixel px is to light arriving at a point x on the camera aperture $\mathcal{A}$ coming from direction $\omega_{\text{i}} \in \Omega$. For example, in the *pinhole camera* model (also known as *perspective camera*), the aperture is a single point and the sensor response function is a simple partition of the field of view. More realistic models can incorporate effects such as depth of field and lens aberrations, for instance by simulating the paths light takes through optical systems (Fig. 2.2).

2.2.3 Distribution of light

We will now investigate the distribution of light itself, and review how it is influenced by objects, materials, and light sources in our scene.

Propagation of light To determine how much light arrives at a point, we need to trace the light back to its origin. The fundamental assumption of geometrical optics is that light travels along straight lines, and as we only consider surface rendering, any incoming light at point x from direction ω_{i} must come from the surface visible from x in direction ω_{i}. Radiance has the convenient property that it stays constant along rays, hence the incoming light L_{i} is exactly equal to the outgoing light L_{o} at the visible point, but in opposing direction $-\omega_{\text{i}}$:

$$L_{\text{i}}(\omega_{\text{i}}, x) = L_{\text{o}}(-\omega_{\text{i}}, \mathrm{RT}(x, \omega_{\text{i}})). \tag{2.2}$$

Here, RT denotes the *ray tracing operator*, which for a given ray returns the closest intersection point with the scene. It is one of the most crucial operations in light transport, as it models the propagation of light. Unsurprisingly, a large body of research explores how to efficiently determine intersections with various types of geometrical models and with enormous amounts of objects (billions and upwards), and recent GPUs even feature hardware-accelerated ray tracing. While fascinating, the inner workings of the ray tracing operator are orthogonal to our work, and hence we refer the reader to Pharr et al. [2016] for a comprehensive discussion.

Surface scattering The outgoing radiance L_o at a surface is modeled by the famous rendering equation [Kajiya 1986], which describes it as sum of self-emission L_e and scattered radiance L_r. The self-emission, which is non-zero only for light sources, is typically described by analytical models, and thus easy to evaluate. The scattered radiance, however, requires integrating over all incoming light and weighting it by a model B of how strongly the material scatters light from the incoming direction ω_i into the outgoing direction ω_o

$$L_\mathrm{o}(\omega_\mathrm{o}, x) = L_\mathrm{e}(\omega_\mathrm{o}, x) + \underbrace{\int_\Omega L_\mathrm{i}(\omega_\mathrm{i}, x)\, B(\omega_\mathrm{i}, x, \omega_\mathrm{o})\, |\cos\theta_\mathrm{i}|\, \mathrm{d}\omega_\mathrm{i}}_{L_\mathrm{r}(\omega_\mathrm{o}, x)} . \tag{2.3}$$

The weighting by the cosine of θ_i, the angle between incoming direction ω_i and surface normal, takes into account that light spreads over a larger surface at grazing angles, and is commonly referred to as *foreshortening factor*. Since L_i is directly related to L_o, it becomes evident that the rendering equation is a recursive integral with an additive term (a Fredholm equation of second kind), which introduces challenges that will be discussed in later sections.

Material models The appearance of objects is a complex interplay of light with the material's chemical properties and its microscopic surface structure. Both are challenging to incorporate into the simulation directly. Instead, one uses phenomenological and statistical models of how the object scatters light, which are modeled by the $B(\omega_\mathrm{i}, x, \omega_\mathrm{o})$: the *bidirectional scattering distribution function* (BSDF). Similar to how the sensor response function describes interactions of light within the camera, the BSDF captures interactions of light and surfaces.

2.2.4 Path integral formulation

For mathematical analysis, it can be more convenient to unfold the recursion of the rendering equation and perform a change of variables to integrate over surface points instead of directions, leading us from an integral equation to a pure integration problem. This formulation is known as *path integral formulation* of light transport [Veach 1998]

$$I_\mathrm{px} = \int_\mathcal{P} f_\mathrm{px}(\bar{\mathbf{x}})\, \mathrm{d}\mu(\bar{\mathbf{x}}). \tag{2.4}$$

Path space The path space $\mathcal{P}$ is the union of all light paths of all possible lengths. A path $\bar{\mathbf{x}} = (\mathbf{x}_0, \ldots, \mathbf{x}_n)$ of length $n + 1$ is defined by its path vertices $\mathbf{x}_i$, where the first point $\mathbf{x}_0$ lies on the lens aperture and the last point $\mathbf{x}_n$ lies on a light source. The measure of the integral is given by μ, which is the area product measure over all vertices of a path.

Measurement function The *measurement contribution function* f_{px} models how much a light path contributes to a given pixel px in our image. It is the product of the sensor response, BSDFs encountered along the path, and emission of the light source at the end of the path

$$f_{\mathrm{px}} = W_{\mathrm{px}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) \left(\prod_{i=1}^{n-1} G(\mathbf{x}_{i-1} \leftrightarrow \mathbf{x}_i) B(\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i \leftarrow \mathbf{x}_{i+1}) \right) G(\mathbf{x}_{n-1} \leftrightarrow \mathbf{x}_n) L_{\mathrm{e}}(\mathbf{x}_{n-1} \leftarrow \mathbf{x}_n). \tag{2.5}$$

We use a different parametrization for W_{px}, B, and L_{e} to emphasize the change of integration domain, but the underlying models are unaffected by this as the direction vectors ω are uniquely determined by consecutive path vertices.

Geometry term Through the change of variables from directions (as in the rendering equation) to surface points (as in the path integral formulation), an additional term known as *geometry term* $G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j)$ is introduced in the integrand

$$G(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) = V(\mathbf{x}_i \leftrightarrow \mathbf{x}_j) \frac{|\cos \theta_{i\to j} \, \cos \theta_{j \to i}|}{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|}. \tag{2.6}$$

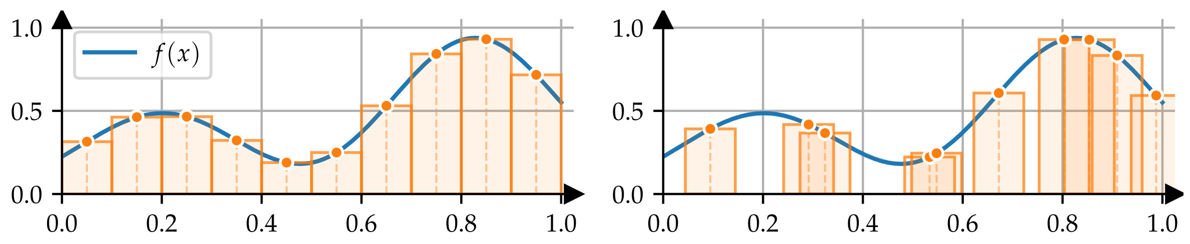
Here, $\theta_{i\to j}$ denotes the angle formed by the surface normal at vertex $\mathbf{x}_i$ and the direction vector from $\mathbf{x}_i$ to $\mathbf{x}_j$, and the *visibility function* V indicates whether the points $\mathbf{x}_i$ and $\mathbf{x}_j$ can see each other (= 1) or whether an occluder blocks their visibility (= 0).

2.2.5 Computational challenges

The equations that govern light transport may look harmless at first glance, but they are challenging enough to have occupied countless researchers for almost four decades. While each model we have discussed boils down to an integration problem, multiple circumstances make computing these integrals analytically infeasible and challenging for numerical methods. In the following, we review the main issues before we move on to the most prominent method to address them in the next section.

High dimensionality The rendering equation is a recursive integral: With each scattering along our path, we introduce another variable that needs to be integrated. Even just a few dozen scattering events, often necessary to achieve photo-realism, result in a dimensionality untractable for most quadrature methods. Ideally, we want to be able to simulate paths of all possible lengths, resulting in infinite dimensionality. The dimensionality is often further increased by considering additional effects, such as motion blur (integration over time) or spectral effects (integration over wavelengths).

Discontinuities and singularities The integrand of the rendering equation is highly irregular. For example, the ray tracing operator introduces complex discontinuities as different geometric primitives may be visible from different directions, which can create sharp shadow boundaries. It is not unusual for scenes to contain billions (and upwards) of such primitives, which makes analytical analysis challenging. Furthermore, small light sources or shiny materials introduce strong, localized peaks in the integrand. For some materials (such as glass) and certain kinds of light sources (such as point lights), these peaks can become singularities, which are particularly challenging to handle by many numerical integration techniques.



(a) *Riemann sum* **(b)** *Monte Carlo*

Figure 2.3: (a) *Deterministic quadrature methods, such as the Riemann sum, evaluate the integrand at a fixed grid of locations.* (b) *Monte Carlo integration samples random locations, which can create clumping but translates better to high dimensionalities.*

2.3 Monte Carlo integration

When integrals cannot be computed analytically, one resorts to *numerical integration*, in which evaluating the integrand is sufficient to estimate the integral value. Deterministic quadrature schemes evaluate the integrand at fixed locations in each dimension, which results in exponential growth of evaluations with dimensionality, making them unsuitable for high-dimensional problems like light transport. *Monte Carlo integration* avoids the curse of dimensionality by sampling the integrand at random positions (see Fig. 2.3), which – along with its flexibility and extensibility – has made it the predominant approach to computing light transport [Fascione et al. 2018b].

Since Monte Carlo integration relies on random processes, we briefly revisit probability theory. We then formally define integral estimators, explore their convergence properties, and introduce the two methods to speed up convergence that this thesis builds upon.

2.3.1 Probability theory

To better understand random processes, we first define random variables and explore their statistical properties, which are later useful to study the convergence of estimators.

Probability space Random processes are commonly described by a tuple $(\mathcal{S}, \mathcal{E}, Q)$. The *sample space* $\mathcal{S}$ describes all possible outcomes of an experiment, for example when rolling a dice, it could capture all orientations and locations the dice ends up in. The *event space* $\mathcal{E}$ contains subsets of $\mathcal{S}$, for example, the subset of all orientations in which "the dice reads 1". Finally, the *probability function* Q assigns each event a number in $[0, 1]$, indicating how likely the event is to happen. As the experiment is repeated over and over, the ratio of trials in which an event happens converges to its probability.

Discrete random variables are mappings from outcomes $\mathcal{S}$ to a countable measurable set $\mathcal{X}$. In our dice example, one such variable could be the parity of the number shown by the dice, with $\mathcal{X} = \{\text{even}, \text{odd}\}$. In light transport, we often map the outcomes of a random number generator to a discrete set of options, for example, which light source we try to connect to, or which distribution we want to sample from. The *probability mass function* P (pmf), which measures the probability of each value of a discrete random variable X, is given by

$$P(x) = Q(\{s \in \mathcal{S} \mid X(s) = x\}). \tag{2.7}$$

Continuous random variables are mappings onto an uncountable measureable set $\mathcal{X}$. In our dice example, we could ask how far away in meters the dice landed, in which case $\mathcal{X} = \mathbb{R}_{\geq 0}$. In light transport, we might ask how much light is transported along a random path we have picked. The *probability density function* p (pdf) measures the density of probability over an infinitesimal neighborhood around x, and – in a slight abuse of notation – obeys the property that for any subset $\mathcal{Y} \subseteq \mathcal{X}$ it integrates to the probability of the subset

$$\int_{\mathcal{Y}} p(x)\,\mathrm{d}x = Q(\{s \in \mathcal{S} \mid X(s) \in \mathcal{Y}\}). \tag{2.8}$$

Expected value According to the *law of large numbers* theorem, repeating a random experiment indefinitely and averaging the values of a variable X will converge against a value[3] known as *expected value* of X, which we denote $\mathbb{E}[\cdot]$

$$\mathbb{E}[X] = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} X_i. \tag{2.9}$$

This value can also be computed analytically. For discrete variables, we know that as the number of trials n goes towards infinity, the ratio with which each value x occurs converges to the probability $P(x)$ of the value, and hence the expected value is equal to

$$\mathbb{E}[X] = \sum_{x\in\mathcal{X}} x\,P(x). \tag{2.10}$$

In our dice example, the expected value of the number shown by the dice (assuming it is fair) is given by $\sum_{i=1}^{6} i \cdot 1/6 = 3.5$.

Analogously, the expected value of a continuous variable can be found by integrating all possible values x weighted by the density $p(x)$ of their occurrence

$$\mathbb{E}[X] = \int_{\mathcal{X}} x\,p(x)\,\mathrm{d}x. \tag{2.11}$$

When a random variable can be expressed through another one, i.e., $Y = f(X)$, we can equivalently compute the expected value by summing or integrating over X

$$\mathbb{E}[Y] = \int_{\mathcal{Y}} y\,p(y)\,\mathrm{d}y = \int_{\mathcal{X}} f(x)\,p(x)\,\mathrm{d}x = \mathbb{E}[f(X)]. \tag{2.12}$$

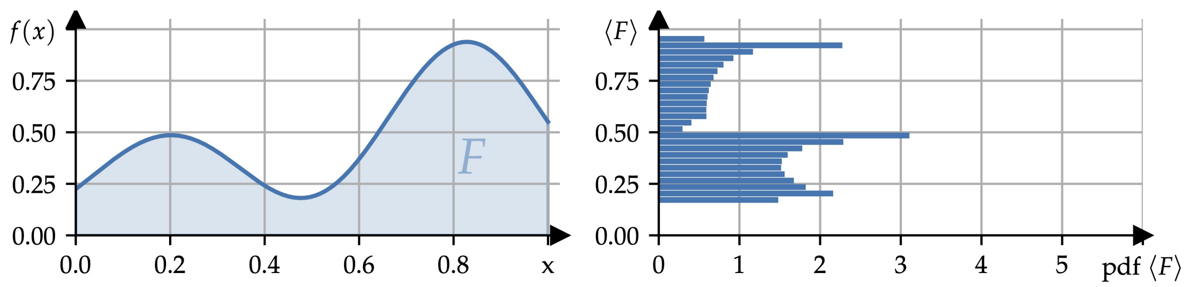
Note that the expected value is a linear operator, which means it obeys *additivity* and *multiplicity*

$$\mathbb{E}[\alpha X + \beta Y] = \alpha\mathbb{E}[X] + \beta\mathbb{E}[Y]. \tag{2.13}$$

Variance In addition to knowing what a random variable will be on average, it can be of great use to know how strongly it is expected to deviate from the mean in the form of a squared error. This is called the *variance* of a random variable, denoted $\mathbb{V}[\cdot]$ and given by

$$\mathbb{V}[X] = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (X_i - \mathbb{E}[X])^2. \tag{2.14}$$

[3] This does not hold for variables of infinite variance, but those will not be relevant to us.



(a) *Integrand* $f(x)$ **(b)** *Distribution of* $\langle F \rangle$

Figure 2.4: *The variance of a primary estimator is directly related to the fluctuations of the integrand* (a)*, as evident from the histogram of values* (b) *that the primary estimator takes on.*

The variance can also be expressed as an expected value

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\ &= \mathbb{E}\left[X^2 - 2\,X\,\mathbb{E}[X] + \mathbb{E}[X]^2\right] \\ &= \mathbb{E}\left[X^2\right] - 2\,\mathbb{E}[X]\;\mathbb{E}[X] + \mathbb{E}[X]^2 \\ &= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2\,.\end{aligned} \tag{2.15}$$

The expected values of powers of random variables $\mathbb{E}\left[X^k\right]$ are important enough that they have their own name: We call them the k-th *moment* of X. In other words, the variance is the second moment minus the first moment squared.

2.3.2 Integral estimators

Let us now see how we can use probability theory to solve integration problems. Our goal is to compute F, the definite integral of a function f over its domain $\mathcal{X}$

$$F = \int_{\mathcal{X}} f(x)\,\mathrm{d}x. \tag{2.16}$$

Primary estimator The key observation is that the expected value of a continuous random variable is itself an integral. To exploit this, we carefully craft a random variable $\langle F \rangle$ with its expected value equal to the integral F. Angular brackets denote that $\langle F \rangle$ is a *primary estimator* of F. For example, if we pick x uniformly, i.e., $p(x) = |\mathcal{X}|^{-1}$, this estimator is given by

$$\mathbb{E}[\langle F \rangle] = \int_{\mathcal{X}} \langle F \rangle\, |\mathcal{X}|^{-1}\,\mathrm{d}x = F \quad \Rightarrow \quad \langle F \rangle = |\mathcal{X}|\, f(x). \tag{2.17}$$

This observation is the foundation of *Monte Carlo integration*. By randomly picking x uniformly from $\mathcal{X}$ and evaluating the integrand $f(x)$ multiplied by the size of the domain $|\mathcal{X}|$, we arrive at an *unbiased* estimate of the integral value. Unbiased means that the result is correct on average, albeit with potential residual noise due to the random process involved.

To determine the amount of noise, we have to analyze the distribution of values that $\langle F \rangle$ takes on. It is the same distribution as $f(X)$, but scaled by the domain size, as shown in Fig. 2.4. An integrand with strong fluctuations will therefore result in a wider (and hence noisier) distribution of $\langle F \rangle$. On the other hand, if our integrand was constant, our estimate would always give the true value. We will later see techniques to reduce the fluctuations of the integrand without changing the expected value.

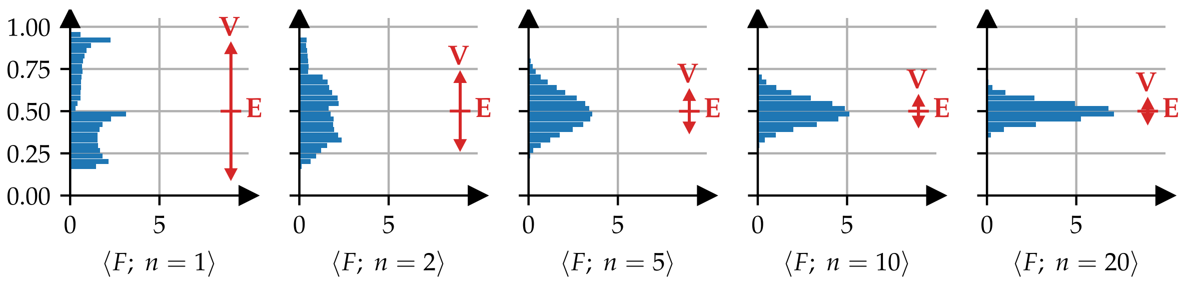


Figure 2.5: *As predicted by the central limit theorem, averaging multiple samples of the primary estimator* (a) *results in narrower histograms* (b-e) *for the secondary estimators, resulting in reduced variance.*

Secondary estimator A trivial way to improve the accuracy of our estimate is to average multiple trials. This forms the so-called *secondary estimator* $\langle F; n\rangle$, which converges towards the true value F with increasing sample count n

$$\lim_{n\to\infty} \underbrace{\frac{1}{n}\sum\nolimits_{i=1}^{n} \langle F\rangle_i}_{\text{secondary estimator}} = \lim_{n\to\infty} \langle F; n\rangle = F. \tag{2.18}$$

Since our individual estimates $\langle F\rangle_i$ are independent random variables with equal distribution to $\langle F\rangle$, it is simple to show that the expected value of the secondary estimator is still F

$$\mathbb{E}\left[\frac{1}{n}\sum\nolimits_{i=1}^{n} \langle F\rangle_i\right] = \frac{1}{n}\sum\nolimits_{i=1}^{n} \mathbb{E}\left[\langle F\rangle_i\right] = \frac{1}{n}\, n\, \mathbb{E}\left[\langle F\rangle\right] = F. \tag{2.19}$$

Similarly, one can show that the variance of the secondary estimator decreases in $\mathcal{O}(n^{-1})$

$$\mathbb{V}\left[\frac{1}{n}\sum\nolimits_{i=1}^{n} \langle F\rangle_i\right] = \frac{1}{n}\mathbb{V}\left[\langle F\rangle\right]. \tag{2.20}$$

This is illustrated in Fig. 2.5. For a sample size of $n = 1$, the distribution of the secondary estimator is equal to the primary estimator. With increasing sample size, we see that the distribution begins to become narrower. Moreover, it also converges to a Gaussian distribution in the process. This is known as the *central limit theorem* [Fischer 2011], and it holds for any distribution of the primary estimator except for those of infinite variance.

Advantages over deterministic quadrature Monte Carlo integration brings many benefits over deterministic quadrature methods. Most importantly, the required number of samples to arrive at a desired error threshold does not depend on the dimensionality of the integral, unlike deterministic quadrature methods haunted by the *curse of dimensionality* [Bellman 1961]. This makes it much more suitable for high-dimensional integration problems like light transport. Furthermore, one can trivially build *progressive* algorithms that do not need the sample count determined upfront but rather stop once some desired error threshold is reached. Unlike deterministic methods, which suffer from systematic error, Monte Carlo integration suffers from stochastic noise. This is indeed an advantage, as *denoising* can filter out noise in rendered images, whereas systematic artefacts like *aliasing* are much harder to handle [Huo and Yoon 2021]. Lastly, Monte Carlo integration offers ample flexibility for extensions, which we will explore in the following.

2.3.3 Efficiency Metric

We now know how we can compute challenging integration problems, but how do we compute them *fast*? To this end, we need a way to quantify how fast an estimator converges towards the true integral value. A common metric, that we will make extensive use of in this thesis, is the notion of *efficiency* $\epsilon\left[\cdot\right]$

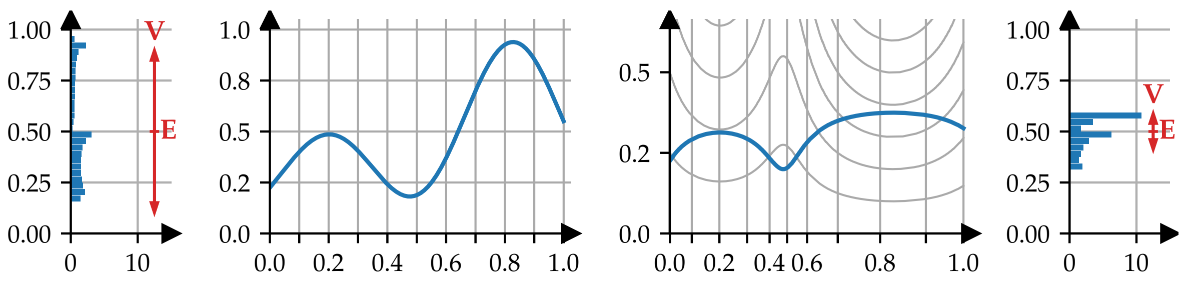
$$\epsilon\left[\langle I\rangle\right] = \frac{1}{\mathbb{V}\left[\langle I\rangle\right]\mathbb{C}\left[\langle I\rangle\right]}. \tag{2.21}$$

Cost The operator $\mathbb{C}\left[\cdot\right]$ denotes the expected computational cost of evaluating $\langle I\rangle$, e.g., how many seconds it takes to evaluate the estimator on average. In theoretical derivations, it is often sufficient to use a simple proxy for the cost – for example, the length of a light path usually correlates strongly with the computational cost of constructing and evaluating it. While many algorithms do not explicitly consider cost in their derivation, it is still reflected in their evaluation – for instance when reporting the time taken for a given sample count or by performing *equal time* comparisons. For the secondary estimator, the cost is a linear function of sample count $\mathcal{O}(n)$ – if we take twice as many samples, computing the estimate will take twice as long.

Variance We have already seen earlier how the variance $\mathbb{V}\left[\langle I\rangle\right]$ corresponds to the expected squared error of $\langle I\rangle$. While other error metrics exist to assess the noise of an estimator, variance has two convenient properties that set it apart from other metrics. First, its quadratic form lends itself nicely to mathematical analysis, as its derivatives needed for analytical optimization result in linear equations. Second, it falls off with $\mathcal{O}(n^{-1})$ in sample count, which combined with the $\mathcal{O}(n)$ dependency that cost has means that efficiency is independent of sample count. This makes sense: A secondary estimator does not become more efficient just by taking more samples. Variance plays an integral role in the theoretical derivations of many light transport techniques and is also reflected by the results of their evaluation.

Optimizing the efficiency of an estimator is the holy grail that most light transport techniques strive for. Some do so explicitly, by building their theory around this metric – for example, how our *Efficiency-aware Russian roulette and splitting* algorithm (Chapter 6) balances cost and variance would not be possible without explicitly optimizing for the efficiency. Many other approaches focus on minimizing the variance and assume that the cost is mostly unaffected by the changes made to the estimator (see *Variance-aware path guiding* in Chapter 4). And even in approaches that make changes that are too opaque for mathematical analysis, for example hand-tuning data structures as we do in *Focal path guiding* (Chapter 5), efficiency is still kept in mind and its empirical evaluation drives many decisions made in the process of designing algorithms.

In the following, we introduce two fundamental techniques to improve the efficiency of estimators. We begin with *importance sampling*, which strives to minimize the variance of the estimator. Afterward, we investigate *Russian roulette and splitting*, which trades off cost and variance to increase efficiency.



(a) *Uniform sampling* **(b)** *Importance sampling*

Figure 2.6: (a) *Fluctuations in the integrand result in noise in Monte Carlo integration.* (b) *We can reduce the fluctuations through a* change-of-variables, *in which we transfer the integrand into a new (distorted) coordinate system. The integral value is unaffected by this transformation, as any stretching in the integration domain (x-axis) is met by a reciprocal stretching in the integrand value (y-axis). The grid lines correspond to the same grid from (a), and illustrate the amount of stretching applied at each point.*

2.3.4 Importance sampling

So far, we have only considered sampling the integration domain uniformly, but it is possible to craft an unbiased estimator from any distribution $x \sim p$, as long as it can sample all non-zero regions of the integrand (i.e., $f(x) \neq 0 \Rightarrow p(x) \neq 0$).

$$\langle F \rangle = \frac{f(x)}{p(x)} \quad \Rightarrow \quad \mathbb{E}\left[\langle F \rangle\right] = \int_{\mathcal{X}} \frac{f(x)}{p(x)}\, p(x)\, \mathrm{d}x = F. \tag{2.22}$$

Intuitively, this can be understood as a change of variables as illustrated in Fig. 2.6. In uniform sampling (Fig. 2.6a), the estimator is directly proportional to the integrand, and hence its variance stems from the fluctuations of the integrand. If we can reduce the fluctuations of the integrand without changing the integral value, we can build unbiased estimators with lower variance. Imagine for a second that our integral was made out of jelly – we could just stretch the integration domain in some places to make the height come down, and squeeze it in other places where we want the height to go up. The mathematical tool that allows us to do this is known as *change of variables* (Fig. 2.6b). Importance sampling stretches the integration domain by placing more samples in some regions (proportional to $p(x)$) and accordingly weights these down in the estimator $\langle F \rangle$ (by $p^{-1}(x)$) to retain the original integral value.

Perfect importance sampling How far can we reduce the variance using importance sampling? Ideally, we cancel all fluctuations of $f(x)$ so that the estimator becomes a constant and its variance becomes zero. This is achieved when p is proportional to f, i.e. $p(x) = \lambda f(x)$

$$\langle F \rangle = \frac{f(x)}{p(x)} = \frac{1}{\lambda}. \tag{2.23}$$

Unfortunately, there is a catch. Since probability densities must integrate to one, it follows that $\lambda = F^{-1}$. In other words, to evaluate p, we must already know the value F we strive to compute. Even worse, sampling a distribution is typically much harder than just knowing its normalization constant. Instead, it is common to use simple-to-sample functions of a similar shape to the integrand or to employ learning strategies that adaptively fit distributions to match the integrand and cancel its fluctuations.

Multiple Importance Sampling Very often, there are multiple candidate distributions that we could sample from. As we will see later, integrands are often the product of multiple functions that are simple to sample on their own, but it may not be feasible to sample their product. In this case, we could use either of the factors to steer importance sampling. Unfortunately, it is not always known beforehand which one matches the integrand best and they might even complement each other in which regions they match well. Ideally, we want a unified strategy that can automatically combine the strengths of all candidates at our disposal. This is achieved by *multiple importance sampling* (MIS), which comes in two important variants:

One-sample MIS forms a *mixture density* p_m from the convex combination of N candidate distributions p_i weighted by α_i. Sampling p_m is achieved by randomly picking a distribution p_i according to the selection probabilities α_i. The resulting one-sample estimator is given by

$$\langle F \rangle_{\mathrm{OS}} = \frac{f(x)}{p_m(x)} = \frac{f(x)}{\sum_i^N \alpha_i p_i(x)}. \tag{2.24}$$

Multi-sample MIS combines samples from N different distributions by weighting them according to how well they work compared to the other techniques.

$$\langle F \rangle_{\mathrm{MS}} = \sum_i^N \frac{1}{n_i} \sum_j^{n_i} w_i(x_{i,j}) \frac{f(x_{i,j})}{p_i(x_{i,j})}. \tag{2.25}$$

At its core, this is a sum over N secondary estimates: Each distribution p_i produces n_i samples $x_{i,1}, \dots x_{i,n_i}$ and estimates the integrand. To make sure that all secondary estimates sum up to the integral value F rather than a multiple of it and to make sure we only use estimators where they perform well, the *weighting heuristic* $w_i(x_{i,j})$ weights the individual primary estimates. For all points where $f(x) \neq 0$, it must sum to one $\sum_i^N w_i(x) = 1$. Additionally, whenever a point x cannot be sampled by some technique $p_i(x) = 0$, its weight must be zero $w_i(x) = 0$.

A particularly popular choice for the weighting function is the *power heuristic* [Veach 1998]

$$w_i(x) = \frac{(n_i p_i(x))^\beta}{\sum_j^N \left(n_j p_j(x)\right)^\beta}. \tag{2.26}$$

Common choices are $\beta = 1$, in which case the heuristic is known as *balance heuristic*, or $\beta = 2$. It ensures that distributions with a higher chance of producing a sample x (either through a high probability density $p_i(x)$ or a high sample count n_i) are assigned higher weights, which helps reduce the variance of the overall estimate $\langle F \rangle_{\mathrm{MS}}$.

2.3.5 Russian roulette and splitting

Another fundamental technique to shape the sample distribution is given through *Russian roulette and splitting* (RRS). Instead of directly changing the distribution of samples, as done by importance sampling, RRS reshapes the distribution while evaluating the estimator.

Russian roulette (RR) terminates the evaluation of the estimator if it becomes foreseeable that the estimator will have an unusually low contribution. For instance, when our integrand is a product of many factors, we might consider termination after sampling and computing the first few factors if they are close to zero. To retain the unbiasedness of the estimator, Russian roulette terminates paths stochastically according to some *survival probability* q and corrects for this in the estimator

$$\langle F\rangle_{\text{RR}} = \begin{cases} \frac{1}{q}\langle F\rangle, & \text{if survived (probability } q) \\ 0, & \text{otherwise (probability } 1-q) \end{cases} \tag{2.27}$$

$$\mathbb{E}\left[\langle F\rangle_{\text{RR}}\right] = q\,\mathbb{E}\left[\frac{1}{q}\langle F\rangle\right] + (1-q)\,\mathbb{E}\left[0\right] = F. \tag{2.28}$$

While Russian roulette introduces additional noise, and therefore always increases the variance $\mathbb{V}\left[\langle F\rangle_{\text{RR}}\right] > \mathbb{V}\left[\langle F\rangle\right]$, a clever choice of survival probabilities q can contain the impact on the variance and greatly improve efficiency $\epsilon\left[\langle F\rangle_{\text{RR}}\right] > \epsilon\left[\langle F\rangle\right]$ by drastically reducing the expected cost of the estimator.

Splitting (S) is the counterpiece to Russian roulette. As soon as it becomes foreseeable that the estimator will have an unusually high contribution, splitting turns the remainder of the estimator into a secondary estimator. In the example of evaluating a product of many terms, we might notice after having sampled and evaluated the first k terms that they are unusually high, and hence splitting will take multiple estimates s of the remaining terms

$$\langle F\rangle_{\text{S}} = \frac{1}{n}\sum_{i}^{s=1}\langle F(x_{0\ldots k-1}, x^{(i)}_{k\ldots n})\rangle \tag{2.29}$$

This will always reduce the variance $\mathbb{V}\left[\langle F\rangle_{\text{S}}\right] < \mathbb{V}\left[\langle F\rangle\right]$, but a careful choice of the splitting factor s is required to make sure that the increase in cost does not outweigh the reduction in variance, which allows for an overall net gain in efficiency $\epsilon\left[\langle F\rangle_{\text{S}}\right] > \epsilon\left[\langle F\rangle\right]$. Ideally, splitting is only performed selectively where the estimator struggles most – such as when the currently sampled factors are high, or the remaining factors are expected to introduce noise.

To predict good survival probabilities and splitting factors, which are commonly combined in a single RRS factor γ (where $\gamma < 1$ invokes Russian roulette and $\gamma > 1$ invokes splitting), one can rely on heuristics (such as considering the first terms if the integrand is a product) or employ learning techniques to build an oracle that determines the fate of estimates.

2.4 Summary

We have seen how the behavior of light can be described through surprisingly tame equations, most of which end up being integrals. Computing these integrals is not so tame, however. The predominant technique to compute light transport is Monte Carlo integration, for which we have laid the foundation in this chapter. We have also seen how the efficiency of integral estimators is defined and learned two important techniques to increase the efficiency. In the following, we will put all of these puzzle pieces together by investigating how previous works have applied them to simulate light transport in practice.

3

Previous Work

In this chapter, we discuss how simulating light can be achieved through Monte Carlo integration (Section 3.1), give an overview of different families of algorithms from previous works (Section 3.2), and conclude by analyzing existing learning based methods based on path guiding (Section 3.3) and Russian roulette and splitting (Section 3.4) in greater detail.

3.1 Introduction

Rendering an image requires us to determine the color of each pixel I_{px}. We know from the rendering equation that I_{px} can be written as an integral over all possible paths $\bar{\mathbf{x}}$ that connect the pixel px with a point on a light source, and that the contribution of each such path is given by the measurement contribution function $f_{\mathrm{px}}(\bar{\mathbf{x}})$ introduced in Eq. (2.5). The job of Monte Carlo integration is to construct and evaluate random path samples, from which the image can then be estimated. This is achieved through the primary estimator

$$\langle I_{\mathrm{px}} \rangle = \frac{f_{\mathrm{px}}(\bar{\mathbf{x}})}{p(\bar{\mathbf{x}})}. \tag{3.1}$$

Evaluating the contribution of a path f_{px} is straightforward: It is a product of terms that are known analytically, namely the camera response function, the BSDF values at each path vertex, and the emitted radiance of the light at the end of the path. Note that these terms can introduce high variation in the integrand. For example, glossy materials reflect some directions much more strongly than others or there can be sharp discontinuities at shadow boundaries. The challenge therefore lies in constructing paths with a suitable distribution $p(\bar{\mathbf{x}})$, the shape of which should be as close to f_{px} as possible to minimize variance.

A robust sampling distribution capturing all relevant light effects can oftentimes perform multiple orders of magnitude better than a naïve one, which drastically reduces the number of required samples and can turn otherwise prohibitive render times into acceptable ones. The dream of a light transport researcher is to find the one distribution to rule them all, but existing techniques so far always strike trade-offs between which effects they handle robustly and which they cannot. In the following, we will look at different families of algorithms for sampling paths and investigate various ways the distribution can be tuned.

3.2 Rendering algorithms

Over the past couple of decades, many different methods have been proposed to sample paths for light transport simulation: Paths can be constructed through random walks, mutations, explicit connections, manifold solvers, merging of nearby vertices, and many more techniques. In the following, we will give a brief overview of the different families of rendering algorithms, as well as their strengths and shortcomings. We limit our discussion to a high-level overview, for a more in-depth discussion we refer the reader to the excellent book by Pharr et al. [2016].

3.2.1 Uni-directional methods

Together with the rendering equation, Kajiya [1986] proposed a simple but effective sampling method to compute it. This method laid the foundation for the now famous *path tracing* algorithm, which has become the cornerstone of many production renderers due to its simplicity and extensibility [Fascione et al. 2018b]. Given its relevance for the remainder of this work, we will examine path tracing in greater detail.

Path tracing is founded on the idea of constructing paths incrementally, one vertex at a time, by performing a random walk through the virtual scene. Similarly to how the ancient Greeks assumed that light begins its journey at the eye, path tracing initiates paths at the camera. As we will later see, starting at the camera has a few benefits, and due to the Helmholtz reciprocity (which states that light paths retain the same contribution if you swap the light source and sensor) this does not affect the outcome of our simulation [Veach 1998].

An overview of the path tracing algorithm is given in Alg. 1. For each pixel, the beginning of the path is found by importance sampling the camera response function W_{px}, which produces a point $\mathbf{x}_0$ on the camera aperture along with a direction ω_{i} to shoot a ray in. Intersecting the ray with the scene yields the next path vertex $\mathbf{x}_1$, at which point we sample the BSDF B for a new random direction to continue. This process is repeated until a desired maximum number of bounces is reached or the ray becomes fully absorbed. Whenever an intersection $\mathbf{x}_i$ lands on a light source, we have found a complete light path $\bar{\mathbf{x}}$ of length $i+1$ and add its contribution to the pixel estimate $\langle I_{\mathrm{px}} \rangle$. The contribution is determined by keeping track of all terms that make up the measurement function in the *throughput weight* T, which also includes all the divisions by PDFs required by Monte Carlo estimation

$$T(\bar{\mathbf{x}}_k) = \frac{W_{\mathrm{px}}(\mathbf{x}_0, \omega_{\mathrm{i},0})}{p(\mathbf{x}_0, \omega_{\mathrm{i},0})} \prod_{j=1}^{k-1} \frac{B(\omega_{\mathrm{i},j}, \mathbf{x}_j, \omega_{\mathrm{o},j})\,|\cos\theta_{\mathrm{i},j}|}{p(\omega_{\mathrm{i},j} \mid \mathbf{x}_j, \omega_{\mathrm{o},j})}. \tag{3.2}$$

Starting at the camera has the benefit that we only construct visible paths that have a chance of contributing to the image and importance sampling the BSDF at each bounce helps eliminate fluctuations in the integrand caused by the glossiness of materials, but an important challenge remains: Path tracing is unaware of where light comes from. It does not systematically find illuminated regions of the scene or construct paths that hit light sources. Especially for small sources of light, like the filament in a light bulb, very few paths will find the light source, and this variance in the distribution of light directly translates to excessive noise in the rendered image. In the following, we will explore popular techniques to address this shortcoming.

Algorithm 1: *Overview of the path tracing algorithm. In practice, each pixel will average multiple paths to reduce noise to acceptable levels, which we leave out for brevity.*

```
1:  function PathTracing
2:      for px ∈ 1..$N_{px}$ do
3:          $\langle I_{px} \rangle = 0$                                   ← initialize pixel estimate to zero
4:          $\mathbf{x}_0, \omega_i = \textsc{SampleCamera}(px)$                  ← start a path from the camera
5:          $T = W_{px}(\mathbf{x}_0, \omega_i) \div p_{W_{px}}(\mathbf{x}_0, \omega_i)$          ← initialize throughput weight
6:          for $d \in 1..\text{MaxDepth}$ do
7:              if $T = 0$ then break              ← no need to continue if contribution will be zero
8:              $\mathbf{x}_d = \text{RT}(\mathbf{x}_{d-1}, \omega_i)$                 ← find next vertex through ray tracing
9:              $\omega_o = -\omega_i$                          ← outgoing direction is where we came from
10:             $\langle I_{px} \rangle \mathrel{+}= T * L_e(\mathbf{x}_d, \omega_o)$                     ← add sample to pixel estimate
11:             $\omega_i = \textsc{SampleBsdf}(\bar{\mathbf{x}}_d, \omega_o)$                ← sample direction to continue in
12:             $T \mathrel{*}= B(\omega_i, \bar{\mathbf{x}}_d, \omega_o) * |\cos\theta_i| \div p_B(\omega_i \mid \mathbf{x}_d, \omega_o)$    ← update throughput weight
```

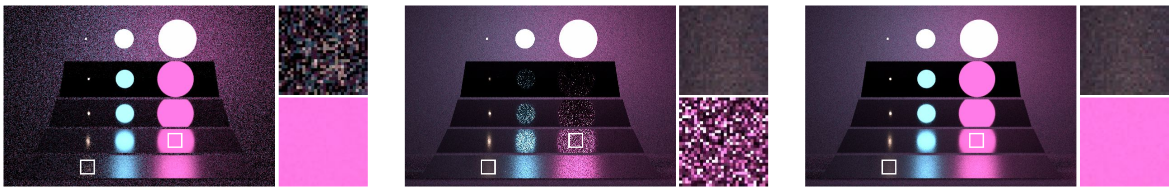
Light tracing (also known as *particle tracing*) also constructs paths through random walks, but starts at the light sources [Arvo 1986]. While this can result in many paths that never reach the camera, it can handle certain effects much more efficiently. Especially when the light sources are small and the camera is easy to find, this can perform much better than path tracing. In practice, however, this technique is rarely used standalone. Rather, it is commonly combined with path tracing to form *bidirectional* methods that we discuss later.

3.2.2 Next event estimation

A prevalent extension to reduce the impact that fluctuations in the incoming light have on path tracing is *next event estimation* (NEE), also known as *direct light sampling*. The idea is to systematically sample paths that end in a light source. For every path $\bar{\mathbf{x}}_i$ that we construct during path tracing, we sample a random point $\mathbf{y}$ on a light source and try to *connect* it with our path to form a new path $\bar{\mathbf{z}} = \{\mathbf{x}_0, \ldots, \mathbf{x}_i, \mathbf{y}\}$. This way, we no longer rely on the BSDF at $\mathbf{x}_i$ to find the light source and instead rely on our light sampling scheme to pick an appropriate point $\mathbf{y}$ that may be visible at $\mathbf{x}_i$. From the viewpoint of the estimator, we are exchanging the importance sampling distribution p_B at the last bounce with p_{NEE}.

Next event estimation works well when the primary source of noise is fluctuations in the incoming light (e.g., small light sources casting light onto diffuse surfaces, which would be rarely found by BSDF sampling), but performs poorly when the material itself introduces variance (e.g., large light source and glossy surface, where light samples would rarely be reflected strongly). It is therefore common to use both techniques and combine their respective strengths through *multiple importance sampling* (see Section 2.3.4) as shown in Fig. 3.1.

It is not uncommon for scenes to feature tens of thousands of light sources – choosing the right point $\mathbf{y}$ on a light source visible at $\mathbf{x}_i$ and strongly reflected by the BSDF is a challenging problem. Many methods have been proposed to construct good sampling distributions p_{NEE} [Walter et al. 2005; Cline et al. 2006; Guo et al. 2020; Yuksel 2021], some of which also employ learning techniques [Georgiev et al. 2012b; Pantaleoni 2019]. In Chapter 4 we show a practical extension to the learning technique of Vévoda et al. [2018] to reduce variance.



(a) *BSDF sampling* **(b)** *Next event estimation* **(c)** *Multiple importance sampling*

Figure 3.1: (a) *BSDF sampling performs nicely for glossy surfaces or large light sources (lower inset).* (b) *NEE, on the other hand, performs great for rougher surfaces and smaller light sources (upper inset).* (c) *We can combine the respective strengths of BSDF and NEE sampling through* multiple importance sampling.

On its own, next event estimation can only handle directly visible light sources. Manifold next event estimation (MNEE) [Hanika et al. 2015; Zeltner et al. 2020] introduces methods to systematically sample specular chains that lead to a light source, such as the reflection of light sources in mirrors or the refraction through glass objects, which are hard to find through BSDF sampling alone. Similar extensions exist for volumetric rendering [Georgiev et al. 2013; Hanika et al. 2022], which can sample additional scattering vertices along the path to the light.

3.2.3 Bidirectional methods

Next event estimation can help when the last link of a path is challenging to find through BSDF sampling, but what can we do if the challenge lies somewhere else along the path? For example, adding a lampshade shifts the problem from finding the last path segment (connecting the lampshade and light source) to the prior path segment (constructing paths that arrive at the lampshade in the first place). A powerful family of methods that can solve this problem are *bidirectional methods*, which combine random walks from the camera (path tracing) and random walks from the light source (light tracing).

Bidirectional path tracing (BDPT) [Lafortune and Willems 1993; Veach and Guibas 1995a] traces a camera path $\bar{\mathbf{x}}$, a light path $\bar{\mathbf{y}}$, and then connects all possible combinations of their prefixes to arrive at complete paths $\bar{\mathbf{z}} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_{s-1}, \ldots, \mathbf{y}_0\}$. It generalizes all methods we have seen so far (see Fig. 3.2): path tracing (paths where $s = 0$), path tracing with next event estimation ($s = 1$), and light tracing ($t = 0$), but also entails many strategies beyond these methods. In our lampshade example, light paths of length $s = 2$ will reliably find path vertices on the lampshade, which we can then easily connect our camera paths to.

An important ingredient in making BDPT successful is combining the different strengths of strategies (s, t) through multiple importance sampling [Veach 1998]. While bidirectional methods perform well for some hard cases, they tend to be less efficient overall, especially for large scenes with many light sources. It is possible to reduce the overhead by limiting the bidirectional techniques to those effects that benefit from them the most, like caustics [Grittmann et al. 2018]. Still, the additional implementation complexity and strict requirement of physically accurate models often make bidirectional methods less appealing in practice [Fascione et al. 2018a; Georgiev et al. 2018].

Photon mapping (PM) handles certain effects that cannot be constructed through connecting camera and light paths. Consider a dewdrop on a flower on a sunny day: We need to find paths starting at the camera $\mathbf{z}_0$, refracting through the dewdrop $\mathbf{z}_1$, scattering on the flower $\mathbf{z}_2$, refracting again $\mathbf{z}_3$, and finally reaching the sun $\mathbf{z}_4$. Path tracing reliably finds $\mathbf{z}_{0,1,2}$, but

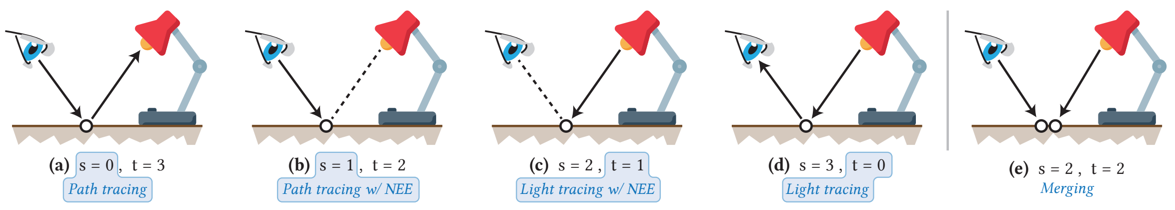


Figure 3.2: *Overview over bidirectional path construction methods for paths of two segment length.* (a) *Path tracing starts at the camera and chooses a random direction at each intersection.* (b) *Path tracing with NEE attempts direct connections to lights at each intersection.* (c-d) *Light tracing chooses to start at the light sources instead.* (e) *Vertex merging considers vertices identical if they are sufficiently close.*

struggles to find a $\mathbf{z}_3$ so that refraction directs us towards the sunlight. For light tracing, the challenge lies in refracting towards the camera. This *specular-diffuse-specular* (SDS, [Heckbert 1990]) interaction breaks BDPT since connections involving specular vertices never contribute as the two subpaths never align just right to obey the law of refraction or reflection.

A solution to our problem is to permit some leeway at the cost of introducing a small bias: If path tracing and light tracing find paths that end at sufficiently closeby points, we can just pretend that their endpoints are the same path vertex. In our example, we would merge paths where path tracing and light tracing land at similar points on the flower $\bar{\mathbf{z}} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2 \approx \mathbf{y}_2, \mathbf{y}_1, \mathbf{y}_0\}$. This technique is known as *merging* and is the basis of algorithms such as photon mapping [Shirley et al. 1995; Jensen 1996; Walter et al. 1997].

Vertex connection and mergin (VCM) combines the respective strengths of connections (BDPT) and merging (PM) through MIS [Georgiev et al. 2012a; Hachisuka et al. 2012b]. Due to the wide array of light effects that it handles, it has become a popular alternative to path tracing in scenes that feature challenging light effects.

3.2.4 Metropolis light transport

As an alternative to sampling paths from scratch each time, it is also possible to employ *Markov chain Monte Carlo* integration (MCMC) to mutate previous paths [Metropolis et al. 1953; Veach and Guibas 1997; Kelemen et al. 2002; Šik and Křivánek 2018]. Through carefully designed mutations and acceptance probabilities, specialized solutions can be found that handle almost any problem. For instance, caustics can be captured through manifold exploration [Jakob and Marschner 2012], small gaps through geometry-aware mutations [Otsu et al. 2018], and so on. An important aspect of these methods is their notion of *target functions*, which steers the distribution of constructed paths. Instead of sampling according to the path contribution, these functions are usually designed to be easier to explore by the Markov chain [Hachisuka and Jensen 2011], to adaptively sample regions of high error [Gruson et al. 2017], or to perform better in combination with a regular path tracer [Šik et al. 2016].

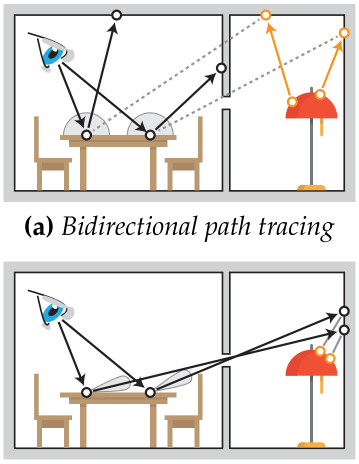
3.2.5 Adaptive sampling

Another common method to control rendering efficiency is via adaptive sampling in image space [Zwicker et al. 2015]. There is a plethora of such methods with very different goals and approaches. What they have in common is that the number of samples per pixel are controlled automatically, based on, e.g., variance estimates. Russian roulette and, in particular, splitting can be seen as a path space extension of such adaptive sampling methods.

3.3 Path guiding

The strategies for path sampling we have discussed so far, except for MLT, are static – they rely on prior information such as the reflection lobes of materials or placement of light sources to make predictions about how the light might be distributed in the scene and sample paths according to those assumptions. While there are many specialized techniques to predict and sample certain effects, implementing all of them is a tedious undertaking, and some effects are not covered by any existing explicit sampling strategy. An interesting alternative is to employ learning mechanisms that continuously refine their assumptions about light distribution throughout the render process and hence can generalize better to unforeseen effects. One of the fundamental methods to learn path distributions is *path guiding*, which is founded on adaptive importance sampling.

Motivating example Imagine a camera in a room where the only light arrives through a keyhole in a closed door (Fig. 3.3). The problem lies in constructing sufficiently many paths that connect the camera and the light in the neighboring room, all of which need to pass through the minuscule keyhole. BSDF sampling is unaware of where light comes from and will rarely construct paths that pass through the keyhole. Connecting path vertices from one room with the other (e.g., BDPT) also rarely succeeds, as nearly all connections are occluded by the door or walls. Path guiding can solve this problem: After finding a few paths that pass the keyhole, path guiding begins to steer more of the rays directly toward the keyhole, resulting in more successful paths and hence significant reduction in variance.


(a) *Bidirectional path tracing*

(b) *Unidirectional path guiding*

Figure 3.3: *Motivating example*

Metropolis light transport Path guiding is not the only approach to utilize information gathered from previous samples. A very related family of techniques, which we have briefly discussed in Section 3.2.4, is Metropolis light transport (MLT). The key difference between the two is that MLT re-uses information immediately by using the last generated path as a template for the next path, whereas path guiding caches information for later use. A key benefit of path guiding is that it avoids correlating paths, which is the reason for uneven convergence in MLT methods. A drawback, however, is that guiding may need many training samples to reliably pinpoint challenging light effects in its representation, whereas MLT methods can start exploring such effects after a single successful path has been constructed.

Overview Various forms of path guiding have been proposed by previous works. They all share the same goal of learning the distribution of light in the scene, but they differ in parametrization, along with the data structure used as representation and the training scheme used to fit it. While path guiding can be performed in bidirectional path tracing [Vorba et al. 2014; Schüßler et al. 2022; Li et al. 2022], most previous works have focused on forward path tracing, which is also what we will be focusing on in this work.

3.3.1 Representations

In conventional path tracing, the BSDF is used to sample which direction to continue the path in at each vertex. Path guiding augments this sampling process – most commonly through an additional distribution that is then combined with the BSDF through one-sample MIS

$$\langle L_\mathrm{r}(\omega_\mathrm{o}, x)\rangle = \frac{\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle\, B(\omega_\mathrm{i}, x, \omega_\mathrm{o})\, |\cos\theta_\mathrm{i}|}{\lambda_B\, p_B(\omega_\mathrm{i} \mid x, \omega_\mathrm{o}) + (1-\lambda_B)\, p_\mathrm{g}(\omega_\mathrm{i} \mid x, \omega_\mathrm{o})}. \tag{3.3}$$

Here, λ_B is the probability of BSDF sampling $\omega_\mathrm{i} \sim p_B$ and $1-\lambda_B$ is the probability of using guiding to pick the direction $\omega_\mathrm{i} \sim p_\mathrm{g}$. Many works choose an equal mixture of the two ($\lambda_B = 0.5$), but we will later see that this parameter can be learned as well to reduce variance.

Radiance sampling Given that the integrand of reflected light L_r is a product of BSDF B with incident light L_i and one of our sampling strategies takes care of sampling the BSDF p_B, a natural choice for the guiding distribution p_g is sampling the incident radiance

$$L_r(\omega_\mathrm{o}, x) = \int_\Omega \overbrace{L_\mathrm{i}(\omega_\mathrm{i}, x)}^{p_\mathrm{g}(\omega_\mathrm{i} \mid x) \propto}\ \overbrace{B(\omega_\mathrm{i}, x, \omega_\mathrm{o})\, |\cos\theta_\mathrm{i}|}^{p_B(\omega_\mathrm{i} \mid x, \omega_\mathrm{o}) \propto}\ \mathrm{d}\omega_\mathrm{i} \tag{3.4}$$

A convenient property of this choice is that p_g has no dependency on ω_o, and hence is "only" five-dimensional: Three parameters for the position x plus two parameters for ω_i given its unit length. This reduced dimensionality results in smaller data structures and faster fitting compared to seven-dimensional distributions that include ω_o. Some works only consider indirect light for p_g and leave direct light to be sampled by NEE only.

Learning arbitrary functions p_g is not feasible in practice as there would be infinite variables, which we can neither fit with the finite amount of information we obtain while rendering nor can we store them in our finite amount of memory. Instead, the guiding distribution is represented as a composition of a finite set of basis functions. This can be achieved in various ways, for instance through photon maps [Jensen 1995], tree structures [Lafortune and Willems 1995; Pegoraro et al. 2008; Bus and Boubekeur 2017], particle footprints [Hey and Purgathofer 2002], or even neural networks [Müller et al. 2017; Bako et al. 2019].

We will focus on the popular approach of partitioning the scene into spatial regions, each individually learning a directional distribution [Müller et al. 2017; Ruppert et al. 2020] as visualized in Fig. 3.4. Partitioning the scene is achieved through a spatial data structure, such as a binary tree or octree, which is refined throughout the rendering process to adapt to the geometry and lighting of the scene. A common strategy is to subdivide regions to achieve an equal number of samples (i.e., path vertices that land in the region) across all of them. The directional distributions are commonly represented by quadtrees [Müller et al. 2017] or mixture models [Ruppert et al. 2020] that similarly adapt to the local lighting conditions.

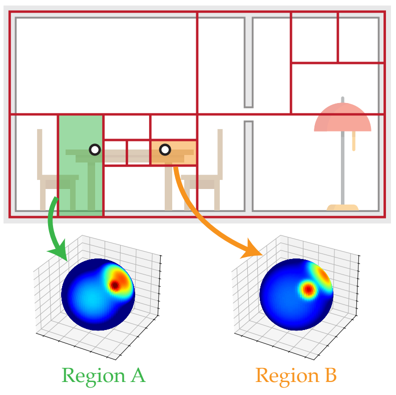
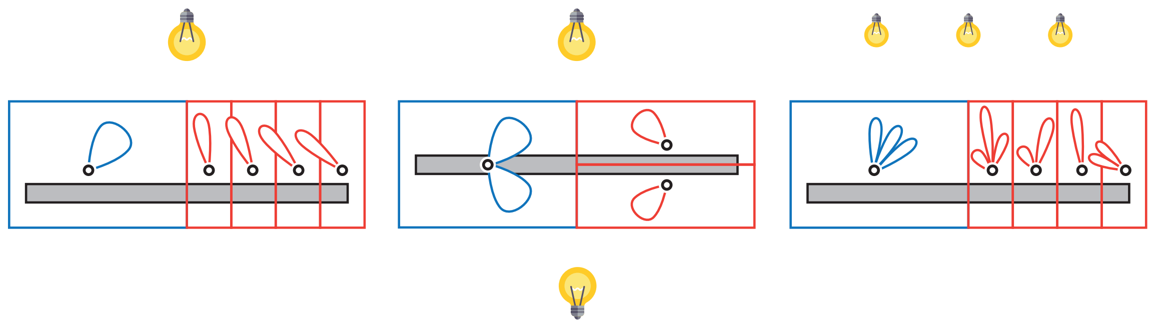


Figure 3.4: *Each region of the spatial subdivision tree contains its own directional distribution.*



(a) *Parallax* **(b)** *Conflicting information* **(c)** *Training stability*

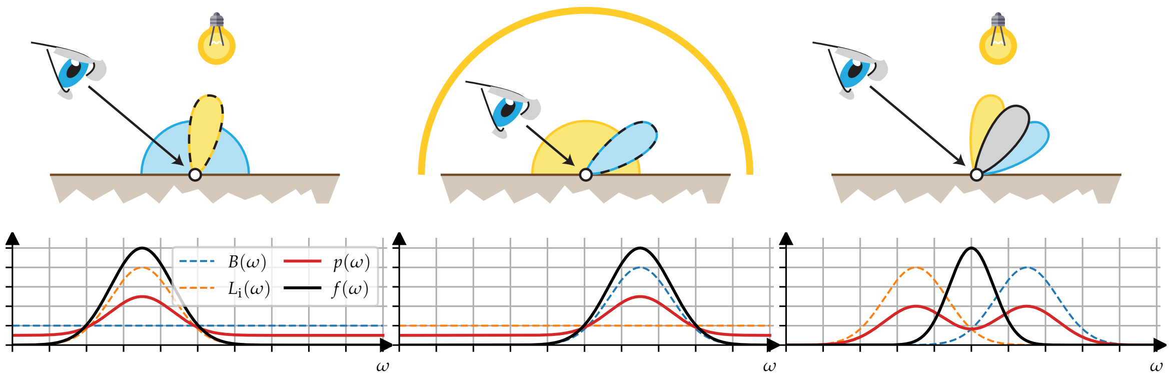
Figure 3.5: *Trade-offs between choosing many or few spatial regions. While choosing higher resolutions can help with spatial variations such as parallax* (a) *and avoid conflicting information in guiding caches* (b)*, they also require more memory and more training samples to avoid noise or missing contributions* (c)*.*

A curious issue with spatial regions is that they are both always too large and yet also too small at the same time. Ideally, we would like them to be infinitesimally small so that they can capture all variations of the lighting conditions, but that would require an infinite amount of memory and training samples. At the same time, we would like them to be as large as possible, possibly even encapsulating the entire scene, to arrive at a directional distribution that fully harnesses all information from our training samples.

Parallax is an excellent example of lighting variations within regions: The exact direction towards a light source can vary greatly within the region, especially with nearby lights (Fig. 3.5a). Parallax compensation [Ruppert et al. 2020] addresses this by reprojecting the directional distribution, but requires the distance to the source of light to be known. In simple cases, such as reflections from a diffuse wall or smooth mirrors, this information is readily available. In more complex cases, such as our motivating example, this is no longer the case. Our focal path guiding approach (Chapter 5) solves this by learning the positions where light is concentrated, and can therefore compensate parallax for a much wider variety of effects.

Another example of variations within regions is conflicting information, for example when two differently illuminated sides of a wall end up in the same spatial region and must therefore share the same directional distribution (Fig. 3.5b). This can cause many paths to be guided in the wrong direction, which can lead to worse performance than BSDF sampling within affected regions. Our variance-aware target density (Chapter 4) mitigates this issue by optimizing the image variance: The influence on the distribution is no longer determined by how much light each side of the wall receives, but rather by its impact on the final image.

On the flip side, it can also be beneficial to have larger regions. When rare events (such as challenging caustics) are present, not all regions might cover the effect in their training samples (Fig. 3.5c). This not only results in excessive noise in the image but can also produce artefacts as some parts of the image sample the effect reliably and others are covered in outliers. Sharing information with neighboring regions [Ruppert et al. 2020] or denoising across spatial regions [Zhu et al. 2021; Nilsson 2023] can partially mitigate this issue. In Chapter 5 we propose the radically different approach of using a single shared distribution over the entire scene and demonstrate that this global sharing of information results can handle extremely rare event situations better than prior approaches.



(a) *Diffuse & small light* **(b)** *Glossy & large light* **(c)** *Glossy & small light*

Figure 3.6: *Sampling the MIS combination p composed of BSDF B and incident light L_i covers many cases well.* (a) *If the variation in the L_r integrand f stems from the glossiness of the material, BSDF sampling handles it well.* (b) *On the other hand, if the incoming light strongly varies under ω, guiding takes care.* (c) *However, a problem arises when both of these vary strongly, as the shape of the product f no longer matches the shape of the sum p.*

Product sampling Mathematically, the MIS combination we have considered so far is a *sum* of two densities. As long as one of them remains mostly constant over ω_i, the shape of this sum resembles the shape of the product sufficiently well to be a decent choice for importance sampling, as the peaks of the sum will align with the peaks of the product (see Fig. 3.6). If both factors contain peaks, then the peaks of the sum may no longer be aligned with the peaks of the product, and sampling can perform poorly. Ideally, we would like to sample the *product* of p_B and p_g rather than their sum.

Various methods have been proposed to construct the product of the two densities, for example by fitting parametric mixture models to the BSDF which can be analytically multiplied if the guiding distribution is also represented by mixtures models [Herholz et al. 2016, 2018; Ruppert et al. 2020]. Similar techniques exist for other representations [Bashford-Rogers et al. 2012; Diolatzis et al. 2020]. Through resampling it is also possible to (approximately) sample the product without constructing its distribution [Talbot 2005], but this has only been applied to volumetric path guiding so far [Deng et al. 2020]. Both of these approaches make sampling more expensive, which can offset the benefits of product sampling.

Higher-dimensional representations Instead of learning directional distributions for various regions in space, one can also construct higher-dimensional distributions that automatically capture effects such as parallax or perform product sampling, for example through neural networks [Müller et al. 2019]. Another popular alternative is to predict the next path vertex based on the preceding one [Dodik et al. 2022; Schüßler et al. 2022], which elegantly captures the correlations (e.g., parallax) between vertices. By conditioning the next vertex on the previous two vertices, these path vertex distributions can also learn product sampling. Some works go further by conditioning on all preceding vertices [Reibold et al. 2018; Li et al. 2022]. This shares similarities with primary sample space guiding [Guo et al. 2018; Müller et al. 2019; Zheng and Zwicker 2019], which alters the distribution of paths by biasing the random numbers they are constructed from. The main drawback of many of these approaches is that their higher dimensionality makes them more costly to sample and require more samples to be fitted reliably, which is especially problematic in the presence of rare events.

3.3.2 Training schemes

While there is a lot of variation when it comes to representations of guiding distributions, previous works mostly follow the same schemes when it comes to training, i.e., fitting the distribution models to the scene. Earlier works performed fitting in a separate preprocessing step before starting the render process [Jensen 1995]. Nowadays, algorithms interweave training and rendering by re-using the paths constructed throughout the render process to continuously fit the guiding distributions in an on-line fashion [Vorba et al. 2014; Müller et al. 2017].

On-line learning proceeds in iterations, for example each one rendering the image with four samples per pixel. After each iteration, the paths that were constructed during the iteration are used to refine the guiding distribution that will be used in the next iteration. For example, regions of the spatial structure that contain many path vertices might be split to increase the resolution. The directional distributions are fit to the sample data depending on the choice of representation. In a histogram approach (such as quadtrees), each direction bin would sum up the estimates of incident light $\langle L_\mathrm{i} \rangle$ from paths that cross it. Parametric mixture models, on the other hand, use algorithms such as *Expectation-Maximization* (EM) for fitting [Vorba et al. 2014; Ruppert et al. 2020].

While preprocesses incur a delay until the rendering can start, on-line learning allows rendering to start immediately, which reduces the "time to first pixel" and thereby helps the user iterate quicker on changes to the scene or algorithm parameters. Proceeding in iterations also lets guiding refine itself, since improved importance sampling does not only reduce the noise in the image, but also in the estimates used to fit the guiding distribution. The final rendered image can be combined by averaging the images from all iterations, which can also be weighted to account for the reduction in noise over time as the guiding distribution improves [Vorba et al. 2019].

The exact parameters for training schemes have received relatively little attention and are mostly founded on heuristics. Müller et al. [2017] for example only use estimates from the last iteration for fitting and hence let the iteration duration grow exponentially. Vorba et al. [2019] propose a heuristic to decide when the distribution has mostly converged and training can be stopped to save computational cost. More recently, Ruppert et al. [2020] propose learning in very short iterations of constant length, which is made possible by retaining information from earlier iterations and efficient implementations of data structure updates.

Mixture optimization An important parameter in combining BSDF sampling and guiding is the mixture ratio λ_B that determines which fraction of samples is produced through BSDF sampling (λ_B) and from guiding ($1 - \lambda_B$). While constant fractions (such as 0.5 for radiance sampling or 0.25 for product sampling) perform well in practice, performance can be increased even further by optimizing the mixture ratio. Many works have investigated optimizing mixture ratios in the general MIS context [Lu et al. 2013; Sbert et al. 2019; Murray et al. 2020]. A popular approach in guiding is to assign each region one mixture ratio and train it using gradient descent [Müller et al. 2019].

3.3.3 Target densities

Research in path guiding has mostly focused on crafting efficient and accurate representations but paid little attention to what to learn with these representations. It is widely assumed that the product of incident radiance and BSDF is the ideal *target density* and that a simple approximation thereof is to learn incident light alone and combine it with the BSDF through MIS. Intuitively, this makes sense, as the ideal importance sampling density should be proportional to the integrand:

$$\langle L_r(\omega_\mathrm{o}, x)\rangle = \int_\Omega \overbrace{L_\mathrm{i}(\omega_\mathrm{i}, x)\, B(\omega_\mathrm{i}, x, \omega_\mathrm{o})\, |\cos\theta_\mathrm{i}|}^{p_\mathrm{g}(\omega_\mathrm{i} \mid x) \propto} \,\mathrm{d}\omega_\mathrm{i}. \tag{3.5}$$

Optimizing local variance Rath [2019] show that this intuition can be misleading: Because the function L_i is not known analytically and must therefore be estimated, the integral contains many random decisions (namely the ones of future bounces) that are *marginalized over*. They show that when variance is present in $\langle L_\mathrm{i}\rangle$, the optimal importance sampling density that minimizes the variance of $\langle L_\mathrm{r}\rangle$ is no longer proportional to the integrand. They also demonstrate how to marginalize over the outgoing direction ω_o in a provably good manner for a common five-dimensional representation. However, their proposed density only minimizes variance locally, which is prone to suffer from conflicting information in spatial regions. In Chapter 4 we address this issue by extending this approach to global optimization of (relative) image variance and provide additional insight into the resulting target density that was absent from the original work.

MIS compensation When one of the densities in an MIS combination can be chosen freely, it can be beneficial to avoid redundancy by focusing only on the regions that are not yet sufficiently explored by the other techniques in the combination. Karlík et al. [2019] propose a practical technique on how this can be applied to environment map sampling and demonstrate that their method can also improve radiance-based guiding algorithms by increasing the contrast of the guiding distribution through subtraction of a constant value. Rath [2019] implicitly use an alternative method for MIS compensation, which we analyze and compare to the approach by Karlík et al. in Chapter 4.

Mixture optimization Vorba et al. [2019] propose a gradient descent scheme to optimize the mixture ratio λ_B which can either minimize the KL divergence [Kullback and Leibler 1951] or $\mathcal{X}^2$ divergence [Rényi 1961] of the one-sample MIS distribution and the integrand of L_r. Minimizing the $\mathcal{X}^2$ divergence has the desirable property that it minimizes the variance of $\langle L_\mathrm{r}\rangle$. However, since the mixture ratio is independent of the outgoing direction ω_o, this can introduce conflicts similar to conflicting illumination in spatial regions. In the worst case, despite minimizing the variance of $\langle L_\mathrm{r}\rangle$, the overall variance in the final image can increase. Similar to how we address the issue of conflicting information in spatial regions, we show how to solve this issue and avoid its accompanying artefacts by optimizing variance globally in image space (see Chapter 4).

3.4 Russian roulette and splitting

As paths grow longer, they lose more and more energy due to absorption at each intersection. At some point, it becomes more efficient to terminate the random walk and invest the time in tracing new paths instead. By terminating paths *stochastically*, fittingly named *Russian roulette* (RR), it is possible to avoid introducing bias in the process [Arvo and Kirk 1990]. The counterpiece to Russian roulette is *splitting*, which continues the path with more than one sample. Splitting is beneficial when the variance stems from a later bounce, as the time saved on sampling variations of the beginning of the path is utilized to sample more variations of later portions of the path [Arvo and Kirk 1990]. Since both of these techniques lie on a spectrum, they are commonly combined under the name *Russian roulette and splitting* (RRS).

The RRS Estimator replaces the primary estimator $\langle L_\mathrm{r} \rangle$ with a secondary estimator

$$\langle L_\mathrm{r}(\omega_\mathrm{o}, x); \gamma \rangle = \frac{1}{\gamma} \sum_{s=1}^{r(\gamma)} \frac{\langle L_\mathrm{i}(\omega_{\mathrm{i},s}, x) \rangle \, B(\omega_{\mathrm{i},s}, x, \omega_\mathrm{o}) \, |\cos \theta_\mathrm{i}|}{p(\omega_{\mathrm{i},s} \mid x, \omega_\mathrm{o})}. \tag{3.6}$$

Here, γ is the *RRS factor* and determines with how many samples the path should be continued. Values in the range $[0, 1)$ indicate that Russian roulette is to be performed and values in $(1, \infty)$ correspond to splitting. To arrive at an integer number of samples, stochastic rounding is commonly employed [Vorba and Křivánek 2016]

$$r(\gamma) = \begin{cases} \lfloor \gamma \rfloor + 1 & \text{with probability } \gamma - \lfloor \gamma \rfloor \\ \lfloor \gamma \rfloor & \text{otherwise.} \end{cases} \tag{3.7}$$

The goal is to pick RRS factors γ that avoid unjustified costs (e.g., terminate paths that contribute little) and prevent sources of excessive variance (e.g., split paths that have found a rare source of light), which leads to overall faster convergence of the rendered image.

Relations to importance sampling Both RRS and importance sampling are powerful techniques to shape the sampling distribution of paths, but they differ in their strengths and shortcomings. While importance sampling has little control over the cost of the estimator, RRS can actively cut costs through termination or amortize the cost of path prefixes by splitting them. On the other hand, RRS is more reactive: Instead of guiding paths to sources of light, as importance sampling does, it can only split (respectively terminate) them after they have found the right (respectively wrong) direction by chance. Hence, it is generally a good idea to simultaneously use RRS and importance sampling.

Overview over existing approaches RRS is one of the fundamental techniques implemented by renderers [Pharr et al. 2016; Fascione et al. 2018b]. Somewhat surprisingly, only a few works have attempted to derive optimal RRS factors. Instead, common implementations are driven by simple heuristics such as surface albedo or estimates of expected path contribution, often in conjunction with hand-tuned parameters such as minimum path length. In the following, we give an overview of the most important previous works and discuss their strengths as well as their shortcomings.

3.4.1 Throughput-based methods

Throughput-based Russian roulette is the most widely adopted technique in practice [Pharr et al. 2016]. It owes much of its popularity to its simplicity: The survival probability of a given path prefix $\bar{\mathbf{x}}_k = (\mathbf{x}_0, \ldots, \mathbf{x}_k)$ depends solely on readily available factors

$$\gamma(\bar{\mathbf{x}}_k) = \underbrace{\frac{W_{\mathrm{px}}(\mathbf{x}_0, \omega_{\mathrm{i},0})}{p(\mathbf{x}_0, \omega_{\mathrm{i},0})} \prod_{j=1}^{k-1} \frac{B(\omega_{\mathrm{i},j}, \mathbf{x}_j, \omega_{\mathrm{o},j})\,|\cos\theta_{\mathrm{i},j}|}{\gamma(\bar{\mathbf{x}}_j)\,p(\omega_{\mathrm{i},j} \mid \mathbf{x}_j, \omega_{\mathrm{o},j})}}_{T(\bar{\mathbf{x}}_k)} . \tag{3.8}$$

Throughput-based RR relies on everything that is known so far about the contribution of the path, namely all factors of the measurement function that are determined by the vertices $\mathbf{x}_{0\ldots k}$. These factors are known as prefix weight (or *throughput*) $T(\bar{\mathbf{x}}_k)$ and are commonly computed iteratively throughout the random walk process (see Alg. 1).

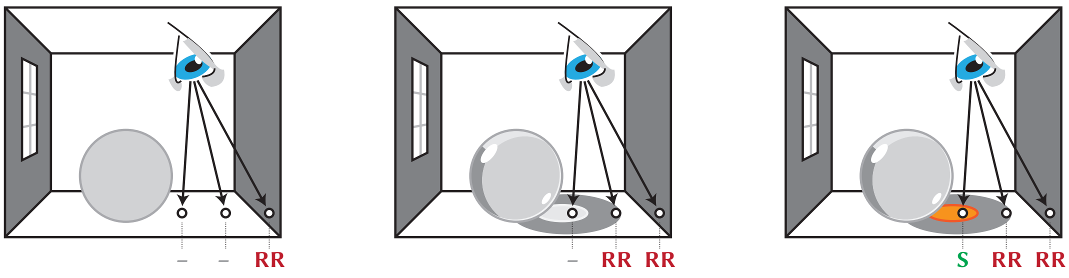
Relations to albedo Bringing $\gamma(\bar{\mathbf{x}}_k)$ into a more explicit form by expanding the dependency on previous survival probabilities $\gamma(\bar{\mathbf{x}}_{0\ldots k-1})$ reveals that all but the last term cancel

$$\gamma(\bar{\mathbf{x}}_k) = \begin{cases} \dfrac{W_{\mathrm{px}}(\mathbf{x}_0, \omega_{\mathrm{i},0})}{p(\mathbf{x}_0, \omega_{\mathrm{i},0})} & \text{if } k = 0 \\ \dfrac{B(\omega_{\mathrm{i},k}, \mathbf{x}_k, \omega_{\mathrm{o},k})\,|\cos\theta_{\mathrm{i},k}|}{p(\omega_{\mathrm{i},k} \mid \mathbf{x}_k, \omega_{\mathrm{o},k})} & \text{if } k \geq 1. \end{cases} \tag{3.9}$$

Especially the latter case is interesting, as perfect BSDF sampling results in a density p proportional to the product of BSDF and cosine, where the proportionality constant is the albedo of the material. For example, a dark surface that only reflects 10% of the incident light has a 10% chance of continuing the path. It is also possible to use the albedo directly [Arvo and Kirk 1990], but for complex shading networks it may not always be known analytically.

Extensions It is also possible to take spectral properties of the albedo into account or reduce variance by returning an estimate of reflected radiance instead of terminating the path with zero contribution [Szésci et al. 2003]. Szirmay-Kalos [2005] propose a heuristic that takes surface roughness into account and can also perform splitting. Common implementations [Pharr et al. 2016] perform RR only beyond a minimum path length (e.g., 5), subject to manual control by the user. It is also common to enforce a chance of termination by clamping the survival probability to some maximum (e.g., 0.95), which prevents infinitely long paths (e.g., when getting stuck between perfectly reflecting mirrors).

Strengths and weaknesses Despite its simplicity, throughput-based methods very effectively cull paths that have little chance of contributing much to the image – for example when they have bounced several times over darker surfaces. However, these methods underlie the assumption that all path prefixes have the same chance of finding light, which is often violated in practice: Even paths with low prefix weight can eventually find an exceptionally bright light source. Neglecting this property can lead to premature termination of important paths, which increases variance.



(a) *Throughput-based* **(b)** *Adjoint-driven* **(c)** *Efficiency-aware*

Figure 3.7: (a) *Throughput RR only reacts to the albedo, e.g., to terminate at dark surfaces.* (b) *Adjoint-driven RRS additionally considers the illumination, e.g., to terminate paths in shadowed regions.* (c) *Efficiency-aware RRS also considers variance (highlighted in orange) and cost, e.g., to perform splitting inside of caustics.*

3.4.2 Approximated contributions

The main issue behind throughput-based RRS is that it only sees the past of the path under construction, but not the future ahead of it (Fig. 3.7). To avoid terminating paths prematurely, it is desirable to consider how much light a given path prefix is expected to find so that the survival probability can be based on the contribution of the completed path. In some cases, such as when connecting camera- and light subpaths, the contribution is known upfront and Russian roulette can be used to randomly skip shadow rays [Veach 1998]. When performing random walks, as in path tracing and hence the focus of our work, we would need to know the incident radiance, which is yet to be estimated.

Adjoint-driven Russian roulette and splitting (ADRRS, [Vorba and Křivánek 2016]) addresses this problem by building an estimate of the incident radiance (the "adjoint"), which is multiplied by the throughput to give an estimate of the expected contribution of the path once completed. The RRS factor is chosen as the ratio of the expected contribution to an estimate of the corresponding pixel's value. An expected contribution higher than the pixel brightness leads to splitting, whereas a lower one leads to termination. The approach of learning incident radiance bears a resemblance to path guiding, in which similar estimates are used to steer importance sampling of directions. Consequently, path guiding and ADRRS are usually applied jointly and can share many of their data structures.

Strengths and weaknesses Basing RRS factors on the expected path contribution improves upon the uniform lighting assumptions of albedo-based methods. For example, a path that bounced over bright surfaces can still be terminated if the adjoint tells us we are in a shadowed region, and paths that bounce over dark surfaces are not terminated prematurely if they are likely to find a bright light source. Beyond this, ADRRS has been shown to perform optimally under zero-variance assumptions [Vorba and Křivánek 2016].

The main shortcoming of ADRRS, however, is that it only considers the expected contributions, not the variances or computational costs thereof. The former means ADRRS cannot directly reduce the variance due to poor sampling decisions, for example inside a caustic: Paths are only split after the poor decision has already been made, effectively bounding the subsequent variance. The latter means ADRRS will equally likely terminate paths that are just about to reach a high contribution as those that still need dozens of bounces to get there. Both of these issues can lead to suboptimal convergence speeds as we will later show in Chapter 6.

3.4.3 Efficiency analysis

The goal of any RRS method is to maximize the speed with which the render converges. A good measure for convergence speed is the efficiency ϵ (see Section 2.3.3), which we want to maximize through the choice of our RRS factors $\gamma(\bar{\mathbf{x}})$ for all path prefixes $\bar{\mathbf{x}}$. Equivalently, one can *minimize* the *inverse efficiency* ϵ^{-1}, which leads to a more convenient equation

$$\gamma^* = \arg\min_{\gamma} \left(\frac{1}{N_{\text{px}}} \sum_{\text{px}}^{N_{\text{px}}} \mathbb{V}\left[\langle I_{\text{px}}; \gamma \rangle\right] \right) \left(\frac{1}{N_{\text{px}}} \sum_{\text{px}}^{N_{\text{px}}} \mathbb{C}\left[\langle I_{\text{px}}; \gamma \rangle\right] \right). \tag{3.10}$$

Here, the overall efficiency of the image is given by the average variance and cost over all pixels. By optimizing all pixels jointly, rather than optimizing their individual efficiencies, pixels can trade off variance and cost between each other in a process similar to adaptive sampling [Zwicker et al. 2015]. Note that γ^* is a function with uncountable infinite degrees of freedom, which makes finding the optimal solution challenging.

In a simplified setting, in which the RRS factors only depend on the originating pixel px and current path length k, Bolin and Meyer [1997] analytically derive optimal factors $\gamma^*_{\text{px},k}$. By operating in image space, they forego the potential of controlling the RRS factor based on the actual prefix path at hand. If a pixel receives contributions from an unimportant and an important region at the same depth, the RRS factor is shared for both, which is suboptimal. Additionally, their method relies on quantities that are costly to estimate with sufficient accuracy, which makes their method less suitable for practical use.

In chapter Chapter 6 we extend this method to RRS factors that use all information of the path prefix. Through a novel fixed-point scheme, we retain the simplicity of practical methods such as ADRRS combined with the benefits of explicitly optimizing efficiency.

3.5 Summary

The reader is now armed with an overview of how Monte Carlo integration can be applied to compute light transport, most notably through the mighty *path tracing* algorithm. We have seen that shaping path distributions to match the relevant effects of the scene is crucial to achieving good performance. A popular approach to robustly handle a wide range of effects is to learn path distributions adaptively, through *path guiding* and *Russian roulette and splitting*.

In the following three chapters, we detail our work on learning robust path distributions. Our overarching theme is to optimize learning for explicit goals – such as minimizing variance, maximizing efficiency, and robustly handling effects that no previous technique handles reliably. Each of our works focuses on the respective learning constituent that is most relevant to achieving the given goal: Target densities, representations, and training schemes. We begin by optimizing variance through target densities in path guiding (Chapter 4), followed by guiding representations to learn challenging effects (Chapter 5), and concluding with training schemes that lead to efficiency maximizing RRS factors (Chapter 6).

4

Variance-Aware Path Guiding

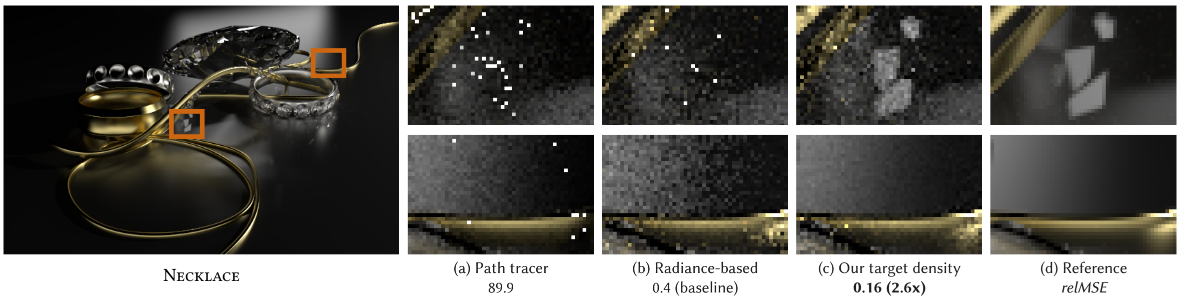
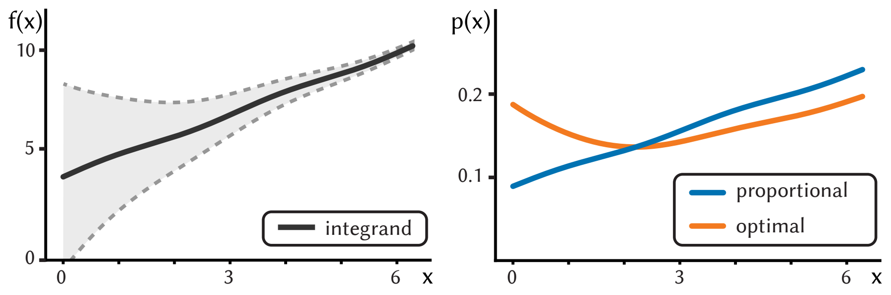


Figure 4.1: *The guiding approach of Müller et al. [2017] benefits greatly from our target densities, e.g., for caustics on glossy surfaces, as shown here. Our method consists of a trivial modification applicable to a variety of path guiding algorithms without additional parameters or computational overhead.*

The majority of rendering systems today rely on unidirectional path tracers [Keller et al. 2015; Burley et al. 2018; Fascione et al. 2018a; Georgiev et al. 2018]. The simplicity, flexibility, and extensibility of the algorithm is what makes it so appealing. The performance, however, depends heavily on the employed importance sampling strategy. Ideally, paths should be sampled proportionally to their pixel contribution. Unfortunately, computing that ideal distribution is a harder problem than rendering the image, because it would require knowledge of the full light transport in the scene. Hence, many implementations construct paths by local sampling from coarse approximations, like BSDF importance sampling.

Path guiding methods learn better importance sampling densities, either locally or for full paths, based on information gathered from previous rendering iterations [Vorba et al. 2019]. The learned densities are then used to importance sample paths in future iterations.

Learning the optimal sampling density for a complete path is often infeasible, due to the curse of dimensionality [Zheng and Zwicker 2019; Müller et al. 2019]. Alternatively, it is theoretically possible to construct an optimal path with only local decisions. To achieve that goal, however, every single local decision needs to be guided perfectly. The optimal local decision for zero-variance sampling is proportional to the product of the BSDF and the incident radiance. Hence, this product distribution would need to be either learned, which is expensive, or computed on-the-fly at every intersection point, which is also expensive. In practice, that usually means that zero-variance sampling cannot be achieved.



(a) Integrand with variance (b) Sampling densities

Figure 4.2: *(a) An integrand (black line) and the variance of its nested estimator (shaded region). (b) Instead of importance sampling the ground truth shape of the integrand $f(x)$, it is better to invest more samples where the variance of the nested $\langle f(x) \rangle$ is high, resulting in provably better performance.*

Previously it was shown that local guiding distributions can marginalize provably well over subsequent sampling decisions by taking their variance into account [Rath 2019]. In effect, more samples are invested towards directions that cause high variance, to adaptively reduce that variance. However, their target density underlies the assumption that each path prefix gets an individual guiding distribution. In practice, distributions are shared across regions, which can introduce conflicting information that reduces the robustness of this method.

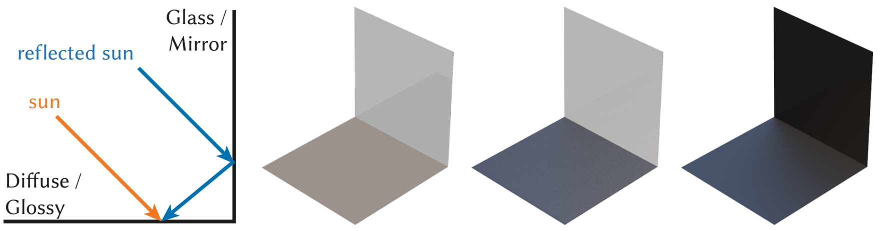
In this chapter, we show how to derive optimal target densities that minimize the image error. To this end, we extend the locally optimal work of Rath [2019] to globally optimal target densities (Section 4.1) and show how our density reduces artefacts in mixture optimization present in previous works [Vorba et al. 2019] (Section 4.2). We apply our theory to two state-of-the-art rendering applications: path guiding [Müller et al. 2017] (Section 4.3) and guided light selection for the many lights problem [Vévoda et al. 2018] (Section 4.4). The implementations for both applications and our mixture optimization consist of trivial modifications to the original code base, without introducing noticeable additional overhead.

The method has been previously published [Rath et al. 2020], I was the main author of that paper. Parts that overlap with my master's thesis [Rath 2019] are cited as previous work. The source code of our implementation can be found on GitHub[1].

4.1 Target densities for local path guiding

In this section, we present a generic approach to derive optimal target densities. At first, we assume that only a single decision along the random walk can be guided. We start with optimal target densities to guide that one local decision at an exact surface point x. We show how to account for variance in nested estimators and marginalization over the outgoing direction in that setting. These steps lead to the same target density derived by Rath [2019], but performing them individually reveals novel insights and intuition that were lacking in previous works. Then, we derive the local target density that minimizes the average error in the rendered image. Lastly, we show how to extend the results to the typical case where densities are learned for regions of space S, rather than exact points x.

[1] https://github.com/iRath96/variance-aware-path-guiding



(a) Cross-section (b) Diffuse / Glass (c) Glossy / Glass (d) Glossy / Mirror

Figure 4.3: *We evaluate the theory on simple test scenes: two perpendicular quads illuminated by a strong sun and uniform sky. The materials are varied for the three different stages of our theory (b-d).*

4.1.1 Adaptive densities: The irradiance integral

A common quantity in rendering is the irradiance $E(x)$ at a point x (e.g., to evaluate diffuse reflection at that point).

$$E(x) = \int_\Omega L_\mathrm{i}(\omega_\mathrm{i}, x)\, |\cos\theta_\mathrm{i}|\, \mathrm{d}\omega_\mathrm{i}. \tag{4.1}$$

The corresponding estimator typically is:

$$\langle E(x)\rangle = \frac{\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle\, |\cos\theta_\mathrm{i}|}{p(\omega_\mathrm{i} \mid x)}. \tag{4.2}$$

Here, for brevity, we assume that only a single sample for ω_i is taken. The incident radiance is computed via a nested MC estimator $\langle L_\mathrm{i}\rangle$.

Previous work usually sets the target function to the ground truth value of incoming radiance:

$$p(\omega_\mathrm{i} \mid x) \propto L_\mathrm{i}(\omega_\mathrm{i}, x)\, |\cos\theta_\mathrm{i}| = \mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle\right] |\cos\theta_\mathrm{i}|. \tag{4.3}$$

This, however, neglects variance in the nested $\langle L_\mathrm{i}\rangle$ estimation. Consider the illustrated example in Fig. 4.2. The shaded region around the black line visualizes the variance of the nested estimator. In the extreme case plotted here, the variance is highest where the integrand is lowest. Hence, sampling proportionally to the ground truth value performs poorly: The region of highest variance would receive the fewest samples.

We can find a better suited target density by minimizing the variance of the estimator,

$$\mathbb{V}\left[\langle E(x)\rangle\right] = \mathbb{E}\left[\langle E(x)\rangle^2\right] - E^2(x). \tag{4.4}$$

The free variable is the PDF $p(\omega_\mathrm{i} \mid x)$: For path guiding, we would like to find the best such PDF and approximate it based on training samples. Looking at (4.4), we can see that only the first term, $\mathbb{E}\left[\langle E(x)\rangle^2\right]$, the second moment, depends on the PDF. The squared ground truth value $E^2(x)$ is constant. The second moment is a convex functional in the PDF, so the minimizer can be found via Lagrange multipliers [Rockafellar 1993]:

$$p_E(\omega_\mathrm{i} \mid x) = \underset{p(\omega_\mathrm{i} \mid x)}{\arg\min}\, \mathbb{E}\left[\langle E(x)\rangle^2\right] + \lambda \left(\int p(\omega_\mathrm{i}' \mid x)\, \mathrm{d}\omega_\mathrm{i}' - 1 \right), \tag{4.5}$$

where λ is the Lagrange multiplier and ensures that $p(\omega_\mathrm{i} \mid x)$ integrates to one, i.e., is a valid PDF. The full derivation can be found in Appendix A. The resulting target density is:

$$p_E(\omega_\mathrm{i} \mid x) \propto \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta_\mathrm{i}|. \tag{4.6}$$

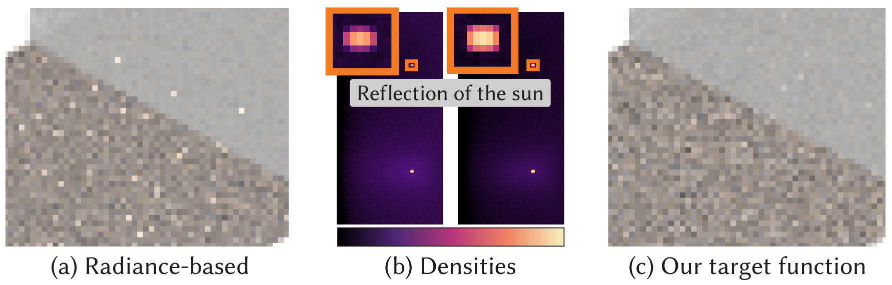


Figure 4.4: *Previous work (a) samples the reflection of the sun proportional to its radiance. Our method (c) compensates for the unguided 'reflect vs refract' decision on the glass with a higher density towards the reflected sun (b).*

That is, sampling should be proportional to the square root of the second moment of the nested radiance estimator, multiplied by the cosine term. If the nested estimator has no variance, the second moment is equal to the squared ground truth: $\mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] = L_\mathrm{i}^2$. Then, our target function is equal to the one used in previous work (4.3). Otherwise, the nested estimator's variance is accounted for.

To test our target density, we apply it to a simple rendering problem: We guide the irradiance estimation on a perfectly diffuse, planar surface. The scene layout is illustrated in Fig. 4.3. In all cases, two perpendicular quads are illuminated by an environment map with a small, strong sun and a uniform sky. Since the direction of the direct illumination is invariant with the position, we build a single, high resolution histogram with a large number of samples, to closely approximate our target density. Then, a small number of samples is taken from the approximated target density.

In the first example, depicted in Fig. 4.4, the floor is diffuse, the wall made of perfectly smooth glass. Hence, the light on the diffuse surface comes mostly from two directions: directly from the sun, and from the sun's reflection in the glass. The latter requires an additional sampling decision of whether to reflect or refract on the glass. In practice, guiding on Dirac delta surfaces (e.g., glass or mirrors) is usually infeasible, since it requires an exact representation of the local, potentially high-frequent 4D light field. Therefore, the decision on the glass is not guided, relying on Fresnel term importance sampling instead. Unfortunately, that results in a large number of rays being refracted through the glass, never finding the bright sun. Our method invests more samples towards the sun's reflection in the glass, compensating for the nested variance.

4.1.2 Marginalized product sampling

The reflected light off a glossy surface is given by the rendering equation integral (2.3). For that case, the zero-variance distribution would be proportional to the product of incident radiance, BSDF, and cosine. Such a product distribution can be computed on-the-fly, given a distribution proportional to the BSDF and one for irradiance, represented in a suitable model (e.g., parametric mixtures) [Herholz et al. 2016]. Then, we would only need to learn our irradiance target function (4.6). Unfortunately, computing suitable representations for all types of BSDF models and their variations is not always feasible [Herholz et al. 2018, 2016].

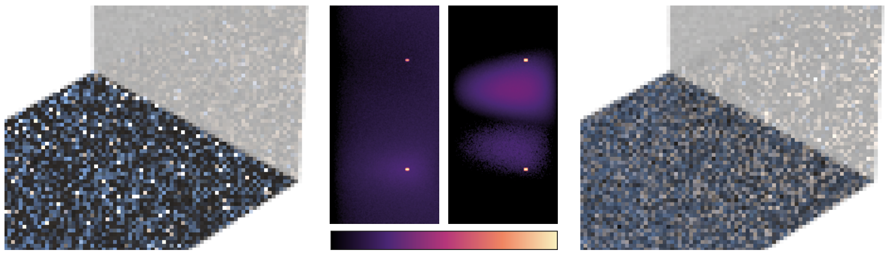


Figure 4.5: *A glossy floor is illuminated by the sun and the sun's reflection in a glass pane. Rather than ignoring the BSDF, we use a provably good marginalized distribution.*

We could still try to learn the zero-variance density $p(\omega_\mathrm{i} \mid x, \omega_\mathrm{o})$. That, however, would be a 7D density, possibly containing high frequencies. The accuracy of the fit, the required number of samples, and the overhead of the implementation can be greatly reduced by simplifying to a 5D density, conditional only on the position x and marginalized over the outgoing direction: $p(\omega_\mathrm{i} \mid x)$. Most previous works perform that simplification by ignoring the BSDF term [Vorba et al. 2019]. A notable exception is the work of Rath [2019], which derives an optimal marginalized target density for the reflected radiance estimator. In the following, we summarize how they arrived at their target density and illustrate the benefits of marginalizing the BSDF.

The goal is to guide an estimator for the reflected radiance, with a PDF independent of ω_o:

$$\langle L_\mathrm{o}(x, \omega_\mathrm{o}) \rangle = \frac{B(\omega_\mathrm{i}, x, \omega_\mathrm{o}) \, \langle L_\mathrm{i}(\omega_\mathrm{i}, x) \rangle \, |\cos \theta_\mathrm{i}|}{p(\omega_\mathrm{i} \mid x)}. \tag{4.7}$$

To find a suitable target distribution, we first need to define our optimization goal. Rath follow the approach of optimizing the expected error under a given distribution of outgoing directions ω_o:

$$p_{L_\mathrm{o}}(\omega_\mathrm{i} \mid x) = \underset{p(\omega_\mathrm{i} \mid x)}{\arg\min} \, \mathbb{E}_{\omega_\mathrm{o}} \left[\mathbb{E} \left[\langle L_\mathrm{o}(x, \omega_\mathrm{o}) \rangle^2 \right] \right] + \lambda \, (\dots). \tag{4.8}$$

Similar to our derivation for the irradiance estimator, they arrive at the target density:

$$p_{L_\mathrm{o}}(\omega_\mathrm{i} \mid x) \propto \sqrt{\mathbb{E} \left[B^2(\omega_\mathrm{i}, x, \omega_\mathrm{o}) \, \langle L_\mathrm{i}(\omega_\mathrm{i}, x) \rangle^2 \right]} \, |\cos \theta_\mathrm{i}|. \tag{4.9}$$

The key differences to the irradiance target density (4.6) are that they average over all outgoing directions and include the squared BSDF.

Figure 4.5 shows the same simple scene as before, with the floor made glossy. Ignoring the BSDF, as done by most previous works results in the exact same density as if the floor was diffuse. The density (4.9), instead, allocates a significant amount of samples to the glossy reflection of the sky. Note, however, that the error in the reflection increases. To reduce the overall error, the density trades a slight increase of noise in the glass for a significant improvement on the glossy surface.

While this target density is optimal regarding the chosen optimization goal (4.8), it is not optimal in a global sense. On the one hand, some glossy effects will still be better handled by BSDF importance sampling, e.g., almost specular reflections. On the other hand, the target density above optimizes for the *most frequent* directions ω_o, which might not be the *most important* ones. In Section 4.2 we summarize how to adapt the target density in an MIS combination, to avoid oversampling glossy effects that are better handled by BSDF importance sampling. But first, the next section will derive a density that accounts for the importance of different outgoing directions.

4.1.3 Minimizing the image error

In our discussion so far, we have neglected the image contribution of the local estimator, which leads to the same target density derived by prior work [Rath 2019]. However, this is only optimal in a local sense. The target densities should ideally minimize the variance of every pixel. In the following, we first derive the optimal local target density if we were able to learn one density per pixel. Then, we show how to extend the derivation to arrive at a marginalized density, shared for all pixels.

Pixel contribution To form our pixel-wise optimal target densities, we need to consider the contribution of a point x to some pixel px. To compute that contribution, one has to consider every path $\bar{x}$ that leads from the pixel to the point x. The contribution is then the outgoing radiance at x multiplied by the throughput of the path $\bar{x}$, integrated over all such paths:

$$C_{\mathrm{px}}(x) = \int_{P_x} f_{\mathrm{px}}(\bar{x}) \, \langle L_{\mathrm{o}}(x) \rangle \, \mathrm{d}\bar{x}. \tag{4.10}$$

Here, $\bar{x}$ is a path starting in the pixel and eventually arriving at the point x. We denote the space of all such paths as P_x. The contribution of the path to the pixel, $f_{\mathrm{px}}(\bar{x})$, is the product of the sensor response and the path throughput. This notation allows us to minimize the image error by minimizing the error due to each individual point x.

Minimizing the pixel error Our goal is to minimize the pixel variance due to local random sampling at the point x in the scene. Consider a pixel estimator that starts by sampling a path $\bar{x}$, starting in a pixel px and eventually arriving at point x (4.10). Its second moment is

$$\mathbb{E}\left[\langle C_{\mathrm{px}}(x)\rangle^2\right] = \underbrace{\int_{P_x} \frac{f_{\mathrm{px}}^2(\bar{x})}{p(\bar{x})} \, \mathrm{d}\bar{x}}_{\text{pixel contribution}} \underbrace{\int_{\Omega} \frac{B^2 \cos^2\theta_{\mathrm{i}}}{p(\omega_{\mathrm{i}} \mid x)} \mathbb{E}\left[\langle L_{\mathrm{i}}\rangle^2\right] \, \mathrm{d}\omega_{\mathrm{i}}}_{\text{local estimator}}, \tag{4.11}$$

where $p(\bar{x})$ is the joint probability of the random walk that leads to the point x. Minimizing this second moment yields:

$$p_{\mathrm{px}}(\omega_{\mathrm{i}} \mid x, \mathrm{px}) \propto \sqrt{\int_{P_x} \frac{f_{\mathrm{px}}^2(\bar{x})}{p(\bar{x})} B^2 \cos^2\theta_{\mathrm{i}} \, \mathbb{E}\left[\langle L_{\mathrm{i}}\rangle^2\right] \mathrm{d}\bar{x}}. \tag{4.12}$$

Note the dependency on the pixel: To use this target density, we would have to learn one density for every pixel in the image. Next, we show how to marginalize over the pixels.

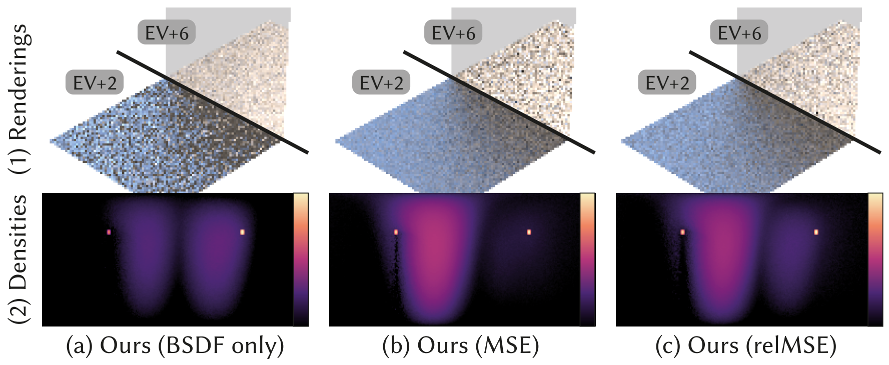


Figure 4.6: *Reflection of a glossy quad in a dark mirror (c.f., Fig. 4.3). The exposure value (EV) is unevenly adjusted over the image. (a) Considering only the frequency of ω_o assigns equal weight to both glossy lobes. (b) Minimizing the MSE favors the (originally) brighter pixels, i.e., the directly visible portion. (c) Minimizing the relMSE balances the noise between bright and dark regions, which is usually more desirable.*

Minimizing the MSE of the image It is usually infeasible to learn one density per pixel. One alternative is to minimize the mean variance over all pixels of the image. That is, instead of minimizing (4.11) for an individual pixel, we minimize the average over all pixels. Hence, the resulting local target density, derived as before, sums over all pixels

$$p_{\mathrm{MSE}}(\omega_\mathrm{i} \mid x) \propto \sqrt{\sum_{\mathrm{px}} \int_{P_x} \frac{f_{\mathrm{px}}^2(\bar{x})}{p(\bar{x})} B^2 \cos^2\theta_\mathrm{i}\, \mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] \mathrm{d}\bar{x}}. \tag{4.13}$$

The result is an estimator that produces the lowest possible mean-squared error (MSE). However, the MSE is not always the best error metric for an image, because it scales quadratically with the pixel luminance. The target density would neglect darker pixels in favor of brighter ones, as the comparison in Fig. 4.6 shows. When replacing the glass pane in our simple example by a dark mirror, the glossy floor is visible from two sets of pixels: the ones that see it directly and the ones that see the dark reflection. Minimizing the MSE over the complete image focuses on the brighter pixels and neglects the reflection almost completely.

Minimizing the relMSE of the image Instead of minimizing the MSE, we propose to minimize the relative MSE (relMSE): the MSE divided by the squared ground truth value of the pixel. The relMSE is independent of the pixel luminance, hence minimizing it achieves a more balanced level of noise. The only modification to our optimization is a division by a constant, the ground truth value I_{px}, which propagates into the target density, yielding:

$$p_{\mathrm{relMSE}}(\omega_\mathrm{i} \mid x) \propto \sqrt{\sum_{\mathrm{px}} \int_{P_x} \frac{f_{\mathrm{px}}^2(\bar{x})}{\tilde{I}_{\mathrm{px}}^2\, p(\bar{x})} B^2 \cos^2\theta_\mathrm{i}\, \mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] \mathrm{d}\bar{x}}. \tag{4.14}$$

The ground truth is, of course, unknown. It can, however, be approximated by denoising or aggressively filtering the image from previous training iterations, as done by Vorba and Křivánek [2016]. We denote the approximated pixel value as $\tilde{I}_{\mathrm{px}}$, which also includes a small offset ϵ to avoid division by zero: $\tilde{I}_{\mathrm{px}} \approx I_{\mathrm{px}} + \epsilon$.

4.1.4 Spatial caches

In practice, guiding distributions are learned not for a specific point x but for a spatial cache cell S (e.g., from a grid or tree structure [Vorba et al. 2019]). Typically, the learned densities are averaged over all points $x \in S$. This averaging, however, results in low densities for directions that matter only to a small number of points.

A simple example is depicted in Fig. 4.7. A density $p(\omega_\mathrm{i} \mid S)$ is learned for each of four spatial cells S. There is no variance in the nested incident radiance estimate, and the target density is approximated with a high resolution histogram. Still, the rendering with previous work shows outliers along the boundaries of S.

The set of points x to which a direction ω_i contributes vanishes at the boundaries. In the limiting case, for a point on the boundary itself, there is a direction ω_i that needs to be sampled for that point, but for no other $x \in S$. When averaging across the whole spatial cell, the resulting density for such an ω_i is almost zero.

A better distribution can be found by minimizing the average error due to all points $x \in S$. For our simplest target density, the optimal one for the irradiance estimator, we change the minimization objective from (4.5) to:

$$p_S(\omega_\mathrm{i} \mid x \in S) = \underset{p(\omega_\mathrm{i} \mid x \in S)}{\arg\min}\, \mathbb{E}_{x \in S}\left[\mathbb{E}\left[\langle E(x)\rangle^2\right]\right] + \lambda\,(\ldots)\,. \tag{4.15}$$

This is analogous to the marginalization over ω_o in (4.8). The resulting target density is:

$$p_S(\omega_\mathrm{i} \mid x \in S) \propto \sqrt{\mathbb{E}_{x \in S}\left[\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]\cos^2\theta_\mathrm{i}\right]}. \tag{4.16}$$

Note that the square root operation is now done *after* averaging over S. Intuitively, this square root 'steepens' the fall-off for directions that contribute only to the boundary, preventing vanishing densities.

The derivation for the other two target densities is analogous, the resulting target density for the simple BSDF marginalization is:

$$\begin{aligned} &p_\mathrm{simple}(\omega_\mathrm{i} \mid x \in S) \\ &\propto \sqrt{\mathbb{E}_{\omega_\mathrm{o}, x \in S}\left[B^2(\omega_\mathrm{i}, x, \omega_\mathrm{o})\,\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]\cos^2\theta_\mathrm{i}\right]}. \end{aligned} \tag{4.17}$$

We refer to this as the locally optimal target density in the following sections. Note that this variant is equivalent to the previous work of Rath [2019]. The target density to minimize the relative error is:

$$\begin{aligned} &p_\mathrm{full}(\omega_\mathrm{i} \mid x \in S) \propto \\ &\sqrt{\sum_\mathrm{px} \mathbb{E}_{x \in S}\left[\int_{P_x} \frac{f_\mathrm{px}^2(\bar{x})}{\tilde{I}_\mathrm{px}^2\, p(\bar{x})} B^2 \cos^2\theta_\mathrm{i}\,\mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right]\,\mathrm{d}\bar{x}\right]}. \end{aligned} \tag{4.18}$$

We refer to this one as our 'full' target density in the following.

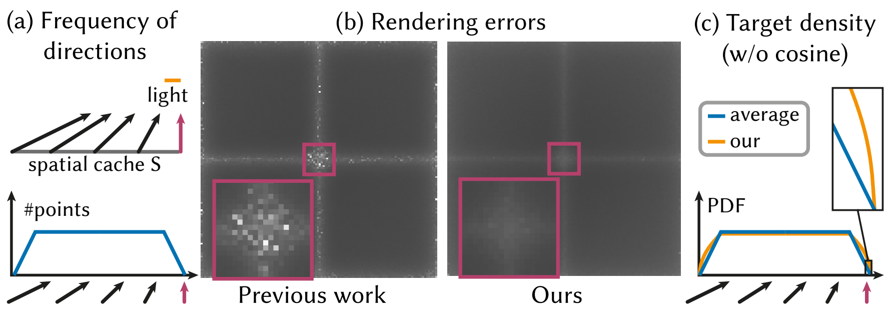


Figure 4.7: *Four directional distributions are learned on a diffuse plane illuminated by a small light source. (a) The points $x \in S$ see the light from different directions. We plot the number of points $x \in S$ (i.e., the surface area) to which each direction contributes. Directions at the boundaries of S are only relevant to a vanishingly small set of points. (b) Visualization of the rendering error, showing outliers at the boundaries. (c) Comparison of the PDFs when averaging (as in previous work) to our result. Our distribution steepens the fall-off at the boundary and eliminates the outliers.*

4.2 Multiple importance sampling

So far, we have discussed the variance of an estimator that uses only the learned sampling strategy. That, however, is insufficient in practice. Relying solely on learned densities can cause excessive variance, or bias [Owen and Zhou 2000]. The reasons include simplification of the integrand (e.g., no BSDF product), marginalization over important terms (e.g., outgoing direction or surface normal), and fitting a possibly inappropriate representation to noisy data. Therefore, it is common practice to combine the learned density with a conservative one, like BSDF importance sampling [Vorba et al. 2014; Hey and Purgathofer 2002].

The combination is usually done via one-sample MIS with the balance heuristic [Veach and Guibas 1995b], resulting in an estimator of the following form:

$$\langle L_\mathrm{o} \rangle_\mathrm{MIS} = \frac{\langle L_\mathrm{i} \rangle B |\cos \theta_\mathrm{i}|}{(1-\alpha) p_\mathrm{g}(\omega_\mathrm{i}) + \alpha p_B(\omega_\mathrm{i})}. \tag{4.19}$$

Here, p_g and p_B are the guiding and BSDF importance sampling distributions, respectively. First, one of the PDFs is chosen at random, where α is the probability of choosing BSDF importance sampling. Then, a direction ω_i is sampled according to the chosen PDF. In the more general case, if we allow not just the balance heuristic but arbitrary MIS weights w_g and w_B, the estimator is:

$$\langle L_\mathrm{o} \rangle_\mathrm{MIS} = \begin{cases} \frac{w_B(\omega_\mathrm{i}, x, \omega_\mathrm{o}) \langle L_\mathrm{i} \rangle B |\cos \theta_\mathrm{i}|}{p_B(\omega_\mathrm{i}|x, \omega_\mathrm{o}) \alpha(x, \omega_\mathrm{o})} & \text{, with prob. } \alpha(x, \omega_\mathrm{o}) \\ \frac{w_\mathrm{g}(\omega_\mathrm{i}, x, \omega_\mathrm{o}) \langle L_\mathrm{i} \rangle B |\cos \theta_\mathrm{i}|}{p_\mathrm{g}(\omega_\mathrm{i}|x)(1-\alpha(x, \omega_\mathrm{o}))} & \text{, else.} \end{cases} \tag{4.20}$$

As shown here, the optimal selection probability $\alpha(x, \omega_\mathrm{o})$ generally depends on the position and outgoing direction.

In the following sections, we first show how to tune our guiding density to perform best in an MIS combination. Then, we revisit previous work on optimizing the selection probability α and demonstrate how insights from our theory can benefit that problem, too.

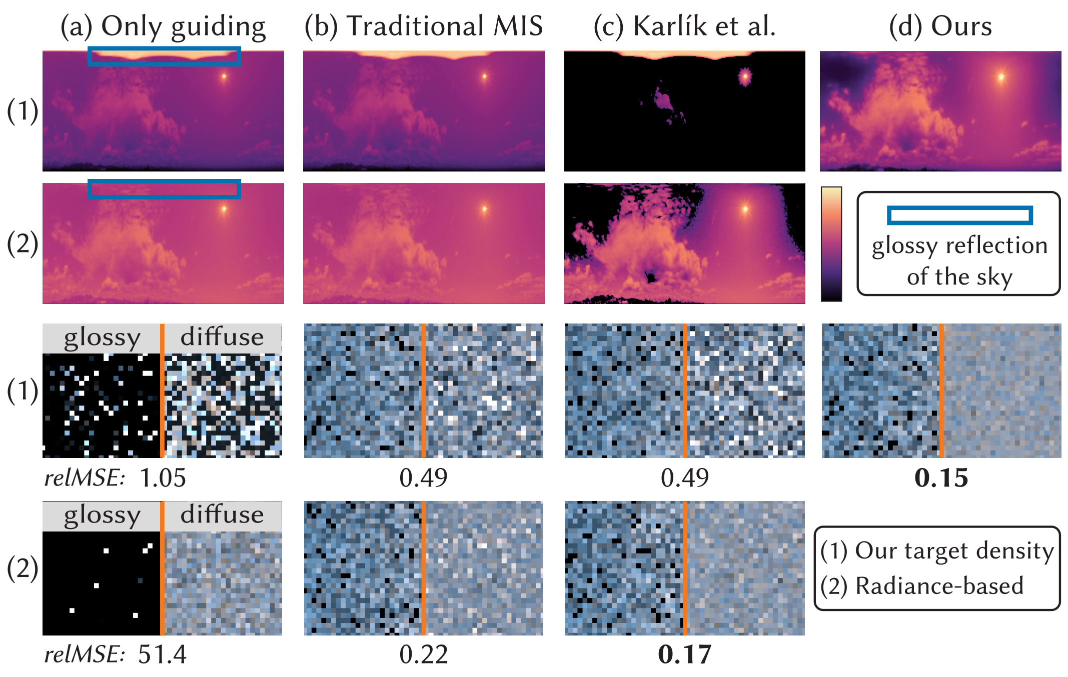


Figure 4.8: *We compare the the MIS compensation strategies of Karlík et al. [2019] and Rath [2019] for radiance-based densities and our target density. A single density is learned for a half glossy, half diffuse quad, illuminated by an environment map. The top rows compare the densities, the bottom rows the rendered images. For more details, see Section 4.2.1. (a) uses only guiding, (b) uses one-sample MIS with BSDF importance sampling, (c) additionally applies the method of Karlík et al. [2019], and (d) uses our MIS compensation. Here, BSDF importance sampling handles the reflection of the sky in the glossy surface well. Our method successfully removes the corresponding directions from the target density, resulting in a lower error than the other approaches.*

4.2.1 MIS compensation

Our target densities from Section 4.1 attempt to capture the full integrand. When combined with BSDF importance sampling via MIS, that is not always the best approach. Consider a case where a guiding cache spans a glossy and a diffuse surface, as shown in Fig. 4.8. Here, our target density (4.18) strikes a trade-off that minimizes the average error across both, increasing the noise on the diffuse surface to avoid outliers on the glossy one. In this example, the glossy reflection of the almost uniform sky is well handled by BSDF importance sampling. Hence, there is no need for our distribution to also cover that portion of the domain.

Instead of a target density that minimizes the error when used alone, we ideally want to learn the density that minimizes the error within an MIS combination. Finding the best such density has been recently proposed under the name of *MIS compensation* [Karlík et al. 2019]. The target density proposed by Rath [2019] already contains a form of MIS compensation, namely by including MIS weights in the target density. In the following, we analyze this approach in greater depth and compare it against previous work.

The approach of Karlík et al. is to subtract a constant from a tabulated PDF, which effectively enhances contrast. While this works well for the radiance-based target density, it does not always perform well when the BSDF is included. In the example from Fig. 4.8, their method cannot remove the strong glossy reflection of the sky.

Rath [2019] mitigate this issue by pretending that the MIS weights are independent of the PDFs. Then, instead of learning to sample the full integral, we only need to learn how to sample the MIS weighted portion of the guided technique:

$$L_\mathrm{o} = \underbrace{\int_\Omega w_g \left\langle L_\mathrm{i} \right\rangle B \left|\cos\theta_\mathrm{i}\right| \mathrm{d}\omega_\mathrm{i}}_{\text{guided portion}} + \int_\Omega w_B \left\langle L_\mathrm{i} \right\rangle B \left|\cos\theta_\mathrm{i}\right| \mathrm{d}\omega_\mathrm{i}. \tag{4.21}$$

The corresponding target density would simply contain the MIS weight as well. For the locally optimal target density (4.9), Rath [2019] arrive at

$$p(\omega_\mathrm{i} \mid x) \propto \sqrt{\mathbb{E}_{\omega_\mathrm{o}}\left[w_\mathrm{g}^2(\omega_\mathrm{i}, x, \omega_\mathrm{o})\, B^2\, \mathbb{E}\left[\left\langle L_\mathrm{i}\right\rangle^2\right]\right]}\, \left|\cos\theta_\mathrm{i}\right|. \tag{4.22}$$

The balance heuristic, of course, is not constant with respect to the PDF. In an iterative learning scheme, however, the densities, and hence the balance heuristic weight, tend to change smoothly between iterations. Therefore, they multiply the current balance heuristic weight on the sample weights, starting with the first guided training iteration. In our simple example (see Fig. 4.8), this approach successfully eliminates the glossy reflection of the sky, which is already captured well by BSDF importance sampling.

4.2.2 Selection probability

Sometimes, the best guiding decision might be not to learn anything and rely solely on BSDF importance sampling. One example could be almost specular surfaces, where the incident radiance is insignificant compared to the BSDF. Using guiding on such surfaces, even in an MIS combination, can increase variance unless the selection probability α is chosen carefully. Some care has to be taken, however, since a poorly chosen α can be far worse than a uniform probability.[2]

Finding the optimal α has been investigated in previous work [Havran and Sbert 2014; Sbert et al. 2016]. A solution in the path guiding context was proposed by Müller [Vorba et al. 2019]. Their motivation is that, ideally, the effective density should correspond to the zero-variance density. Thus, they propose to minimize the divergence between the effective density $p_\mathrm{eff} = (1-\alpha)\, p_\mathrm{g} + \alpha\, p_B$ and the zero-variance density p_zv, using stochastic gradient descent.

Unfortunately, the optimal selection probability is a function of the outgoing direction: $\alpha(\omega_\mathrm{o}, x)$. In practice, however, only a single value per cache cell S is learned, i.e., $\alpha(x \in S)$. Therefore, the outcome of the gradient descent will not be the optimal α. Instead, the expected divergence over all ω_o is minimized:

$$\alpha(x \in S) = \operatorname*{arg\,min}_\alpha \mathbb{E}_{\omega_\mathrm{o}}\left[\mathcal{D}(p_\mathrm{eff} \,||\, p_\mathrm{zv})\right]. \tag{4.23}$$

Here, $\mathcal{D}$ denotes some divergence function, e.g., KL divergence [Kullback and Leibler 1951].

[2] A safer alternative to optimizing the selection probability is the use of control variates [Owen and Zhou 2000; Kondapaneni et al. 2019].

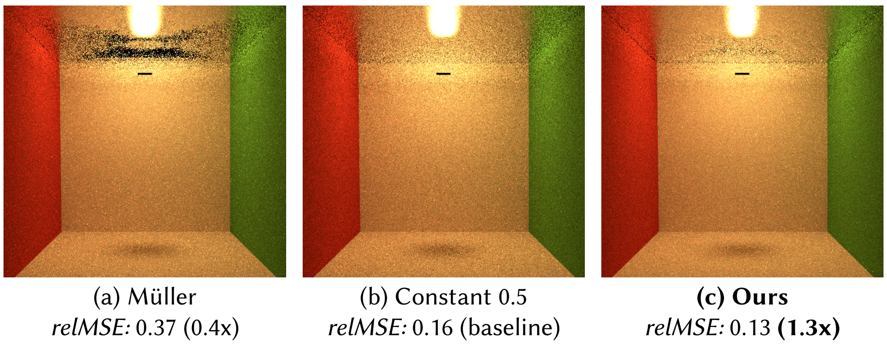


Figure 4.9: *A variant of the Cornell box with a glossy ceiling illuminated by a small light source shining upwards, rendered with different BSDF selection probabilities. The method proposed by Müller (a) performs worse than the uniform selection baseline (b). Our modification (c) increases both robustness and efficiency.*

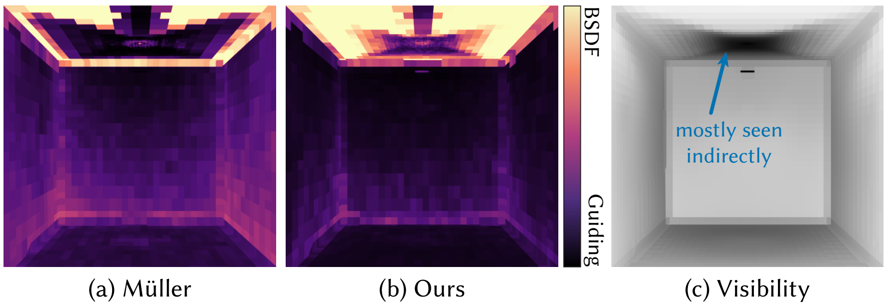


Figure 4.10: *Comparison of the learned selection probabilities (a-b) in the modified Cornell box. The gray-scale image (c) shows the ratio of primary to secondary rays in each guiding cache. White cells are only seen directly by the camera, black ones contribute strongly to indirect illumination. Guiding on the ceiling is only beneficial for indirect illumination, so it should only occur if the ratio (c) is close to zero, as is the case for our method.*

Minimizing the expected divergence over a given distribution of ω_o is certainly a reasonable approach. However, it is not optimal. When comparing this to our target density (4.9), the potential problem becomes apparent: Only the *distribution* of ω_o is considered, not the *contributions* of the different directions.

An extreme case scenario is shown in Fig. 4.9. A small light is turned towards a glossy ceiling, indirectly illuminating a box. The scene is rendered with the original guiding approach of Müller et al. [2017]. Whether guiding is beneficial on the ceiling greatly depends on ω_o: For the light's reflection seen by the camera, BSDF importance sampling is far superior and guiding performs poorly. For the indirect illumination caused by the light reflecting onto the walls, however, the light is at a grazing angle of the glossy lobe, hence guiding is the better choice there. Unfortunately, as visualized in Fig. 4.10, the majority of outgoing directions on the ceiling are due to the indirect reflection. Hence, the optimization for α neglects the directly visible component, producing a severe variance artefact.

To correct this issue, we apply a similar approach as with our target densities: Instead of minimizing the expected divergence, we minimize the expected divergence weighted by the relative contribution:

$$\alpha(x \in S) = \arg\min_{\alpha} \sum_{\text{px}} \mathbb{E}_{\bar{x}} \left[\mathcal{D}(p_{\text{eff}} \,||\, p_{\text{zv}}) \frac{f_{\text{px}}(\bar{x})}{\tilde{I}_{\text{px}}} \right], \tag{4.24}$$

where $\bar{x}$ is a camera path leading to the point x, $f_{\text{px}}(\bar{x})$ is the pixel contribution, and $\tilde{I}_{\text{px}}$ is the approximated ground truth value of the pixel. The resulting α does not produce the artefact in the extreme case discussed here, and performs better throughout all our test scenes.

It is important to guarantee robustness when optimizing the selection probability. The guiding distribution can cause bias or unbounded variance due to estimation errors. Therefore, it is generally a good idea to, at the very least, enforce $\alpha \geq 0.1$ [Owen and Zhou 2000]. Furthermore, the stochastic gradient descent optimiziation proposed by Müller [Vorba et al. 2019] can behave unpredictably for severe outliers. Finding more robust estimation schemes is a very important but orthogonal problem, beyond the scope of this work.

4.3 Application I: Path guiding

In this section, we discuss our primary application: designing target densities for a guided unidirectional path tracer. We apply our theory on top of the approach by Müller et al. [2017]. In the following, we first outline the mathematical formulation. Then, we present pseudo-code with the necessary changes to the implementation. Lastly, we evaluate our method on various test scenes.

4.3.1 Estimating the target density

A nested tree structure is used to represent the guiding distribution. The scene is partitioned into independent guiding caches S by a binary tree, each approximating a directional density $p(\omega_{\text{i}} \mid x \in S)$, using a quad-tree. We have derived an optimal target density for that case, (4.18). Note that our derivations locally optimize densities assuming that all other decisions are fixed. In the path guiding setting, decisions along the random walk of the nested estimator are also guided. Because we are training in iterations, this is not a problem: each iteration learns a density for the current variance of the nested estimator, which typically is close to the actual variance in the next iteration.

The remaining question is how to optimally approximate our PDF using a piecewise constant quad-tree. Each leaf node k in the quad-tree stores a weight γ_k, which determines the probability of choosing the corresponding set of directions D_k for piecewise uniform sampling. The optimal value for γ_k can be computed using the approach of Pantaleoni and Heitz [2017]. For our full target density (4.18) the result is:

$$\begin{aligned} \gamma_k &\propto \sqrt{|D_k| \int_{\omega_{\text{i}} \in D_k} p^2_{\text{full}}(\omega_{\text{i}} \mid x \in S)\, \mathrm{d}\omega_{\text{i}}} \\ &= \sqrt{|D_k| \int_{\omega_{\text{i}} \in D_k} \mathbb{E}\left[\frac{f^2_{\text{px}}(\bar{x})}{\tilde{I}^2_{\text{px}}\, p^2(\bar{x})} B^2 \langle L_{\text{i}} \rangle^2 \cos^2 \theta_{\text{i}} \right] \mathrm{d}\omega_{\text{i}}}, \end{aligned} \tag{4.25}$$

where $|D_k|$ is the size of the kth leaf, measured in solid angle. We can easily estimate the leaf node weights γ_k by accumulating the squared sample weights, multiplying once by the leaf size, and taking a square root prior to normalization.

```
  function Render():
     for i in iterations:
        RenderImage()

+       for Leaf in NextCache:
+          Leaf.Value := Sqrt(
+             Leaf.Value * Leaf.Area
+          )

        NextCache.Normalize()
        CurrentCache := NextCache
        NextCache.Reset()

  function $L_o$(x, $\omega_o$,
+                RelThroughput):

     // One sample MIS
     if Random() > $\alpha$:
        $\omega_i$ := CurrentCache(x).Sample()
     else:
        $\omega_i$ := BSDF(x, $\omega_o$).Sample()

     // MIS computations
     MisPDF := (1-$\alpha$)    * CurrentCache(x).PDF($\omega_i$)
                  +  $\alpha$    * BSDF(x, $\omega_o$).PDF($\omega_i$)
+    MisWeight := CurrentCache(x).PDF($\omega_i$) / MisPDF

     // Compute recursive estimate
     BsdfCos := BSDF(x, $\omega_o$).Eval($\omega_i$) * Cos($\theta_i$)
     $L_i$ := $L_o$(RayTrace(x, $\omega_i$), -$\omega_i$,
+             RelThroughput * BsdfCos / MisPDF)

     // Update guiding cache
     NextCache.Leaf(x, $\omega_i$) += (1 / MisPDF) *
-       $L_i$
+       (BsdfCos * $L_i$ * MisWeight * RelThroughput)^2

     // Update BSDF sampling fraction loss
     MISLoss(x).Update(
        BsdfCos * $L_i$
+       * RelThroughput
     )

     // Compute rendering equation estimate
     return $L_e$(x, $\omega_o$) + BsdfCos * $L_i$ / MisPDF
```

Figure 4.11: *Pseudo-code with the required changes to compute our full density with the algorithm by Müller et al. [2017]. Lines starting with "-", highlighted in red, denote parts that are replaced by our approach. Those starting with "+", in green, compute our proposed target density.*

4.3.2 Implementation

We have applied our theory in the Mitsuba [Jakob 2010] implementation provided by Müller et al. [2017]. The pseudo-code in Fig. 4.11 highlights the required changes. We have implemented our full target density (4.18) with the MIS compensation and selection optimiziations discussed in Section 4.2.

Müller et al. render the image in iterations (see the function `Render` in lines 1–3). Thus, the guiding cache learned in the previous iteration, `CurrentCache`, can be used for importance sampling while learning a new cache for the next iteration, `NextCache`. After each iteration is finished, we multiply the leaf values by the leaf sizes, take the square roots, and normalize (lines 5–12).

The $\mathtt{L_o}$ function (line 14) is called by `RenderImage` to recursively estimate the rendering equation. Note that our full distribution requires us to keep track of the relative pixel contribution $\mathrm{px}(\bar{x})/(p(\bar{x})\tilde{I}_{\mathrm{px}})$, see (4.18), which we do here using the parameter `RelThroughput`. When called directly from `RenderImage`, this parameter is set to the sensor response divided by the pixel estimate, i.e., $W_{\mathrm{px}}/\tilde{I}_{\mathrm{px}}$. We update `RelThroughput` for the recursive evaluation of $\mathtt{L_o}$ in lines 27–28.

Irrespective of which technique was chosen for one-sample MIS in lines 16–23, we always need to compute the `MisWeight` for the guiding strategy (line 24) for our MIS compensation (4.22). Furthermore, in line 32, we accumulate the squared sample weights, multiplied by the MIS weight, to estimate our target densities. Lastly, we apply our modified selection probability optimizer in line 36, again weighting by the relative contribution.

4.3.3 Results

We evaluated our method on a corpus of 22 scenes, all of which are rendered at a resolution of around 640×360 on an AMD Ryzen™ 9 3950X (16 cores / 32 threads @4.0 GHz) workstation with 64 GB of memory. No Russian roulette is performed to aid comparability. We compare our approach to radiance-based guiding, the target density used by previous work. Both guiding approaches make use of next event estimation. In addition, we compare to an unguided path tracer with next event estimation, and a VCM method that uses Markov chains to distribute photons [Šik et al. 2016]. For the latter, we used the authors' publicly available Mitsuba implementation.

In the following, we discuss the differences in equal time renderings on four representative examples. The full results, including convergence tests with long training times, can be found in the appendix.

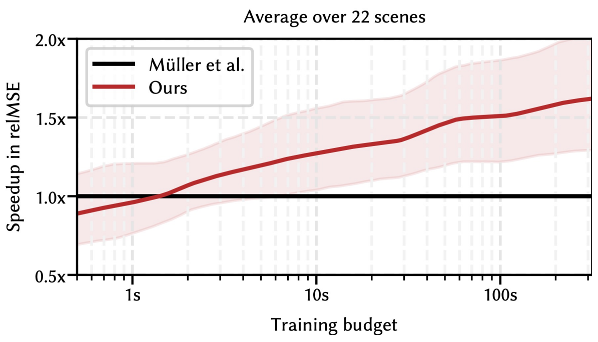


Figure 4.12: *Comparison of training cost. We plot the ratio of the relMSE after equal time (the 'speedup'), averaged over 22 scenes, using the geometric mean. The shaded region visualizes how much that ratio varies across scenes. The error is that of a 512spp rendering after different training times.*

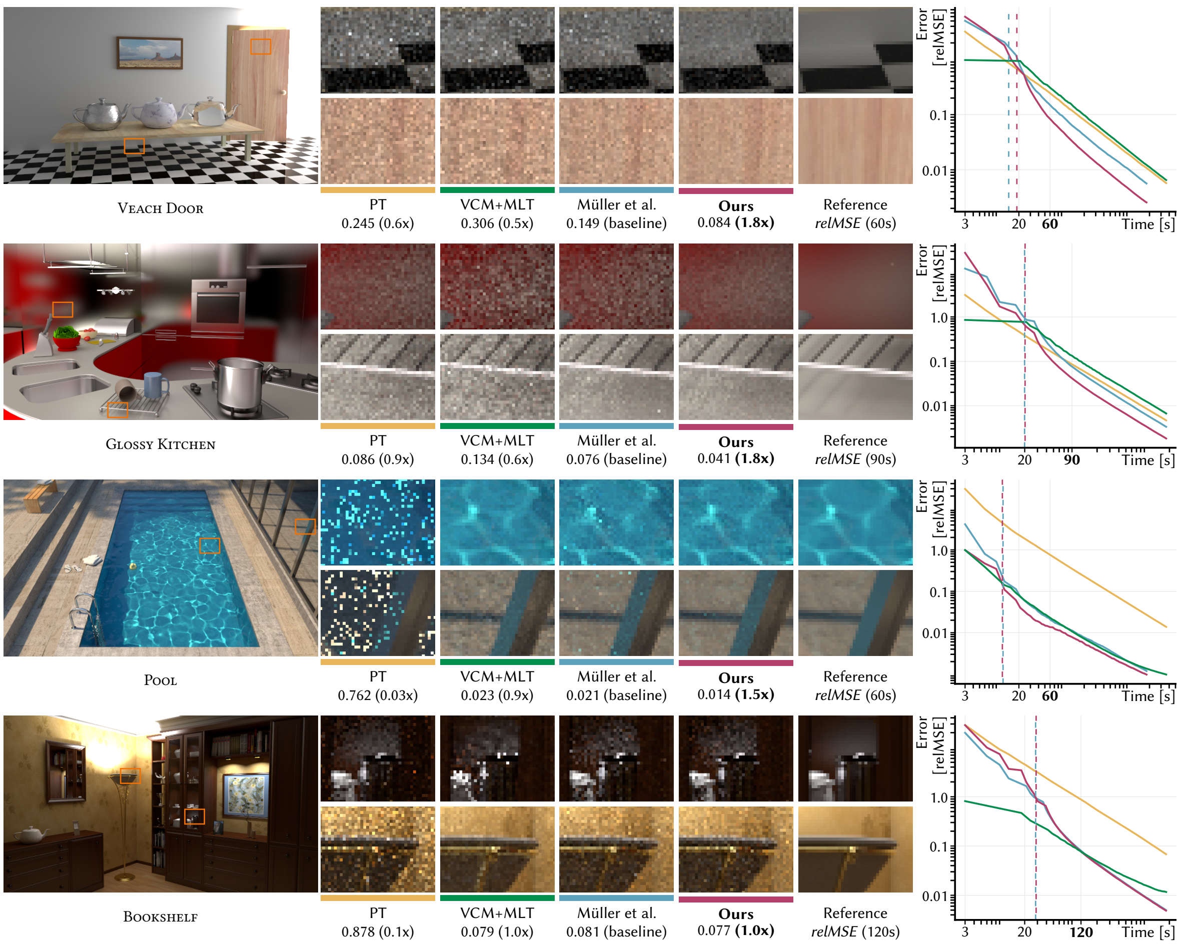


Figure 4.13: *Equal-time comparisons for four test scenes. The dashed lines in the plots mark the end of the last training iteration of the guiding methods. The rendering time of the comparison images is highlighted (60s for* Veach Door *and* Pool*, 90s for* Glossy Kitchen*, and 120s for* Bookshelf*).*

The Veach Door scene (Fig. 4.13, first row) shows how our method reduces spatial caching artefacts. Both the wall and the door are very challenging, as the surfaces on both sides end up in the same spatial cache. Even though the backside is more strongly illuminated than the side seen from the camera, its contribution to the image is less important. Our density mitigates this problem by assigning lower weight to the samples from the backside illumination, resulting in lower levels of noise overall.

The Glossy Kitchen scene (Fig. 4.13, second row) features many glossy surfaces. By incorporating the BSDF into our density (see Section 4.1.2), we are able to improve performance in regions where both radiance-based guiding and BSDF sampling perform poorly.

The Pool scene (Fig. 4.13, third row) features caustics which are challenging to render. The caustics in the pool feature a similar light transport to Fig. 4.4: sunlight is seen directly through the water surface as well as reflected by the window on the right, causing outliers on the pool floor in radiance-based approaches. Our density eliminates these outliers by taking the variance due to the unguided decision on the glass into account.

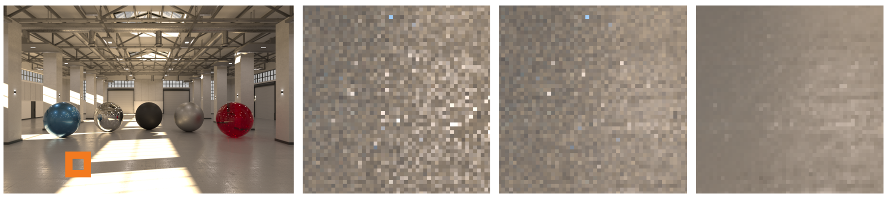
The Bookshelf scene (Fig. 4.13, fourth row) features strong indirect illumination and is thus among the most challenging scenes for our guiding density. Since our density contains the full pixel integral, opposed to just the radiance, its estimate can be noisier for longer paths. Nevertheless, our method still achieves the same performance as the original method, at least for longer renderings. This scene also shows how unidirectional path guiding is still sometimes outperformed by bidirectional methods like VCM, especially for short renderings.

Our target density only consistently outperforms the baseline with sufficient training. To measure the training cost, we computed the error (relMSE) of 512spp renderings after different training times. The ratio of that error between our method and the approach taken by previous work is shown in Fig. 4.12, averaged across all scenes. After 1.5 seconds of training, our target density outperformed the previous one on average, after 10 seconds we outperform it in every single scene. Hence, interactive renderings will not benefit from our results, as the training data is too scarce to accurately learn the density. Longer running renderings of more than a minute, however, receive consistent improvements and converge 50% faster on average.

In conclusion, our target density offers visible improvements across all scenes, at essentially no cost. While it is not suited for interactive preview renders, it is a robust alternative to radiance-based guiding for long renders, where it can noticeably accelerate convergence.

4.4 Application II: Light selection

We tested our theory in a different context and code base, by applying it to a light source selection method [Vévoda et al. 2018]. Vévoda et al. apply a Bayesian approach, where they start with a coarse, analytic approximation as a prior distribution. During rendering, they gradually learn a better posterior distribution. Their distribution already compensates for the variance of nested estimators. That is, they effectively implemented the discrete analogy of our target distribution for irradiance (4.6). We modified their method to additionally marginalize over the BSDF, i.e., compute our the locally optimal target distribution (4.17).



	(a) Vévoda et al.	**(b) Ours**	(c) Reference
relMSE whole image:	0.0070 (baseline)	0.0066 **(1.1x)**	
relMSE zoom-in:	1.2 (baseline)	0.33 **(3.5x)**	

Figure 4.14: *Results from our light selection application. The* Hall *scene features many small lights and a glossy surface. Here, even BSDF importance sampling performs poorly. Our method achieves visible noise reduction by marginalizing over the BSDF. The images were rendered with equal sample count, which is also equal time, since our modifications caused no overhead.*

4.4.1 Implementation

We implemented the original approach and our changes in a custom renderer. Computing our target distribution is also trivial in this case. In principle, only one change is required: we remove the upper bound of the cosine term that was originally used, and instead multiply the BSDF and cosine on the weight of each sample. To that end, we modify their equation (6) to now read:

$$\hat{e} = B(\omega_o, x \to y)\, \frac{L_e(y \to x)\, V(y \leftrightarrow x)\, G(y \leftrightarrow x)}{P(l \mid c)\, p(y \mid l)} = \hat{e}_x. \tag{4.26}$$

Here, y is a point on light source l in cluster c and x is the shading point. V is the visibility term and G the full geometry term, now including both cosines. $P(l \mid c)$ and $p(y \mid l)$ are the probabilities of selecting light source l in cluster c and point y on that light, respectively. The sample weights $\hat{e}$ and $\hat{e}_x$, which are now equal for our target distribution, are used to learn the posterior distribution.

We also performed another change that is not strictly required but improved the learning rate. The prior distribution for the original method was built with the original target distribution in mind, which does not contain the BSDF or cosine terms. Multiplying by the BSDF and cosine yields smaller weights, which we crudely approximated by dividing equation (10) in the original paper by a factor of eight, a number we found to perform well empirically over all our tests. Finding the optimal prior distribution is an orthogonal and application specific improvement left for future work.

4.4.2 Results

We tested our modified algorithm across a variety of test scenes. In this application, the difference was less significant than with the local path guiding method, since the nested estimator's variance was already accounted for previously. For most scenes, our result was only slightly better than the original version. However, we also did not find a single scene where our target distribution performed worse.

One specific type of scene can benefit greatly from our target distributions: Scenes with glossy surfaces and small light sources. Glossy surfaces were neglected by the previous distribution and left for the BSDF importance sampling strategy to resolve. However, if the light sources are small, BSDF importance sampling can often perform poorly, even on glossy surfaces. An example is shown in Fig. 4.14. In this modified version of the Hall scene, we reduced the size of the light sources. Neither the original target distribution nor the BSDF importance sampling strategy can resolve all of the glossy highlights. By marginalizing over the BSDF, our target distribution achieves substantial improvements in the highlights, while producing identical results everywhere else.

In conclusion, while the benefit of our target distribution is less significant in this application, we still achieved robust improvements with trivial changes and no additional overhead.

4.5 Limitations and future work

Isolated optimization We have optimized local decisions in isolation. The individual optimizations (target density, selection probability, and MIS compensation) are aware of each other only through the observed changes in the sample weights of future iterations. In our empirical tests, adding another isolated optimization resulted in consistent improvements. There is, however, no strict proof that the isolated decisions will converge to an optimal joint distribution. Further investigation in that direction is an interesting avenue for future work.

Short renderings Target densities that are theoretically optimal can still result in poor rendering performance when estimated with few training samples, i.e., for short preview renderings. In Section 4.3, we have observed this effect, where our full density sometimes only outperforms the baseline after sufficient training. Estimating a higher dimensional integral, namely the image contribution of the guiding cache, results in higher levels of noise. There are multiple possibilities to improve performance for these cases. One option is to apply reconstruction or denoising methods to the guiding caches. Another option is to design prior distributions and utilize a Bayesian approach [Vévoda et al. 2018].

Other target densities We have focused on target densities that would be optimal if they were estimated exactly. A different approach, that could also improve performance for short render times, would be to design target functions that are easier to learn. To that end, regularization could be employed [Hachisuka et al. 2012a] or a binary distribution could be learned, similar to MCMC target functions that only include visibility [Hachisuka and Jensen 2011]. Exploring such target densities is an interesting avenue for future work. It could also be interesting to find target densities that minimize different error metrics, like perceptually-based ones.

Bidirectional guiding The applications presented in this paper can easily estimate the variance due to nested estimators, as the training samples are generated from a distribution similar to the one that will be used during rendering. If that is not the case, e.g., because guiding is done bidirectionally [Vorba et al. 2014; Jensen 1995], computing the target densities is more involved, but still possible. Computing PDFs in a bidirectional setting can be tedious, a main reason why these methods are less appealing in practice. A workaround to ease implementation effort could be to only use the bidirectional samples to initialize a simpler, coarse guiding distribution. Successive iterations can then learn unidirectionally (possibly from both sides) and easily estimate our target densities.

5

Focal Path Guiding

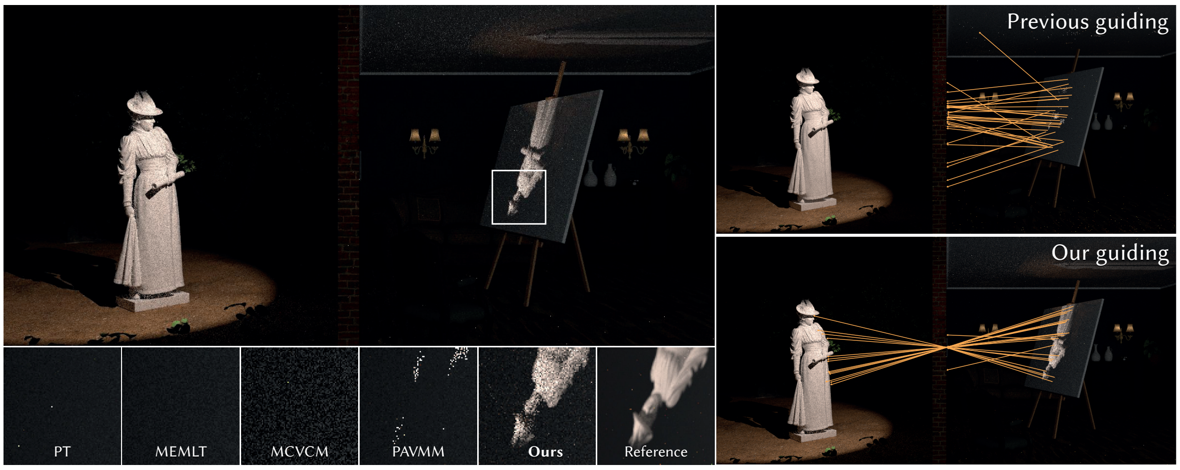


Figure 5.1: *Focal effects emerge when light is concentrated in small regions of space, for example, due to lensing effects, or narrow gaps as demonstrated in this* Camera Obscura *scene. For some of these effects, no reliable specialized technique exists, and even state-of-the-art path guiding methods fail as their representations struggle to capture the focal regions that rays must converge in. In this chapter, we introduce a novel guiding representation that is specifically tailored to handle focal effects robustly.*

The *camera obscura* (see Fig. 5.1), in which a small hole in a wall projects an inverted image of the outside world into a dark room, marks a milestone in the history of photography. Somewhat ironically, despite its pervasive use throughout computer graphics in the form of the pinhole camera, actually *rendering* a camera obscura remains almost impossible without manual intervention. In this chapter, we investigate a family of light effects that all share the same difficulty – light converging in small regions in space, either to pass narrow gaps or by being converged through lenses – and coin them "focal effects". Even sophisticated methods struggle with focal effects, as the location and extent of the focal regions in which the rays converge are not always known beforehand.

We propose a novel form of guiding representation, which is specifically tailored to handle focal effects. Instead of learning a directional or path space distribution like previous work, our method learns a spatial distribution that is not restricted to surfaces. Our method can robustly identify and sample focal regions, even when they occur in free space. We categorize

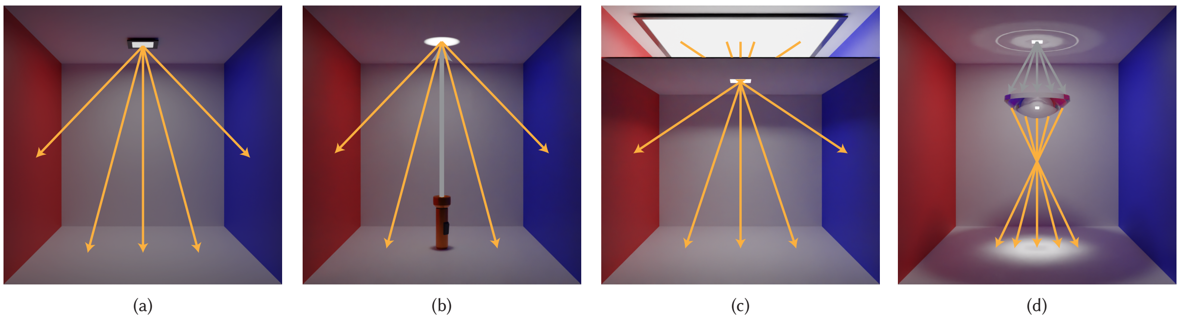


Figure 5.2: *We identify common causes of focal points. Surface-bound focal points include direct light sources (a) and indirect sources like caustics or spots (b). Focal points also occur in free space, for example when light must pass narrow gaps (c) or when lensing effects shift the apparent position of other focal points (d). Existing algorithms only handle subsets of such effects, often relying on smoothness assumptions or specific types of geometry. Our method does not rely on such assumptions and unifies all focal effects in a single framework.*

focal effects into different classes and show that our method is the first to unify all of them in a single framework. For many classes of focal effects, our algorithm achieves substantial improvements over the respective previous state-of-the-art.

We begin by identifying and classifying common sources of focal regions and survey how they are handled by existing families of algorithms (Section 5.1). We then introduce an iterative scheme that identifies relevant regions of focal light transport in a given scene (Section 5.2) and present a novel guiding approach based on sampling focal regions including its efficient implementation (Section 5.3). The method introduced in this chapter has been published before [Rath et al. 2023], of which I am the main author. Our implementation of the proposed algorithm is available at https://github.com/iRath96/focal-guiding.

5.1 Focal effects

In the following, we outline the importance of focal effects in light transport and discuss how they are handled by existing techniques. To this end, we group focal effects in different classes and introduce terminology that we will use throughout the remainder of the paper.

Focal points arise where many light paths from different directions converge in a small region of space. Common causes for focal points are illustrated in Figure 5.2. We classify focal effects in the following three categories:

- *Direct focal points* are small light sources or the camera itself (*light/camera focal points*)
- *Indirect focal points* are caused when objects interact with light, either through diffuse reflection of brightly illuminated spots (*diffusing focal points*) or by forcing light to pass through narrow gaps (*occlusion focal points*)
- *Virtual images* are the apparent position of focal points seen through (potentially multiple) reflection or refraction events

We refer to focal points as *surface-bound* when their position lies on a surface (light, camera, and diffusing focal points) and as *free space* when they occur away from surfaces (occlusion focal points and virtual images).

5.1.1 Direct focal points

These focal effects are the simplest to handle. Since their location is known beforehand, they lend themselves well to explicit sampling, either by starting random walks in their position or by trying to connect to them directly (e.g., through next event estimation). A careful sampling of light sources may be necessary if the scene contains many of them [Walter et al. 2005; Cline et al. 2006; Vévoda et al. 2018; Guo et al. 2020; Yuksel 2021].

5.1.2 Indirect focal points

More challenging are indirect sources of focal points, the location of which is not known a-priori. Diffusing focal points can be connected to, either by bidirectional methods or by learning their location through path guiding. Occlusion focal points cannot be connected to, as they do not occur at path vertices, but rather at an unknown point on the path in free space. While occlusion of direct light can be sampled explicitly through user intervention [Bitterli et al. 2015; Ogaki 2020], complex visibility of indirect light is still notoriously difficult and only explored well by specialized MCMC mutations [Otsu et al. 2018].

5.1.3 Virtual images

The complexity of sampling virtual images depends greatly on the type of the original focal point and the size and smoothness of the surface producing the virtual image.

Direct virtual images For virtual images of direct focal points, random walks initiated at the focal point work well if the surface causing the virtual image is large enough to be found by exploration (e.g., a camera pointed at a mirror is handled well by forward path tracing). Connections to virtual images of direct focal points are possible through specialized sampling techniques [Hanika et al. 2015; Zeltner et al. 2020; Jakob and Marschner 2012] (e.g., connecting to glass-enclosed light sources in forward path tracing), but require that the surface producing the virtual image is sufficiently large and smooth.

Indirect virtual images Virtual images of indirect focal points are more challenging: While diffusing focal points can be handled by some specialized techniques [Li et al. 2022; Jakob and Marschner 2012], images of occlusion focal points remain extremely difficult to find and can only be explored through MCMC methods. Apart from virtual diffusing focal points [Li et al. 2022] and reflections of surface-bound focal points in flat mirrors [Ruppert et al. 2020], virtual images are currently not well explored by path guiding methods, as they are unable to reconstruct the location of free space focal points.

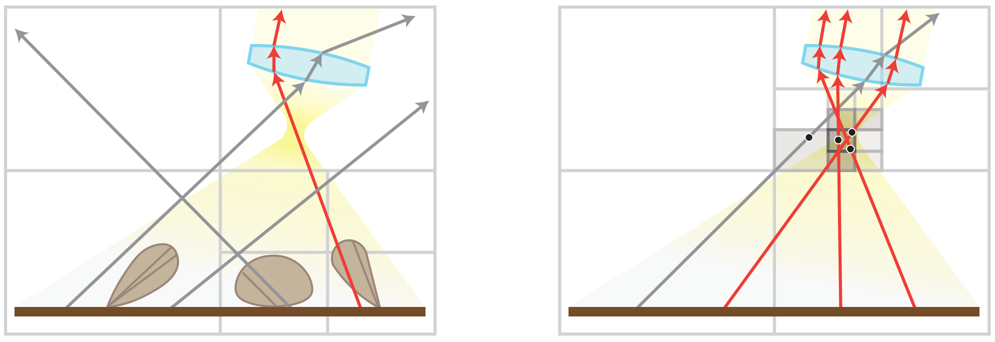

(a) *Directional distributions* **(b)** *Our spatial distribution*

Figure 5.3: *(a) Most guiding algorithms partition space and learn directional distributions per region. Due to unaccounted parallax effects, many rays miss the focal point. Some regions never find the focal point during training and cause artefacts. (b) Our method samples the spatial regions themselves, inherently compensating for parallax and sharing information globally.*

5.2 Focal guiding

We propose a novel form of path guiding, which reliably samples paths involving focal points that otherwise cause excessive variance even for sophisticated algorithms. Like many previous approaches, our guiding augments forward path tracing and iteratively refines its importance sampling density during rendering. Instead of learning local directional distributions directly, our method learns a global distribution over space and samples rays that pass through focal regions (see Figure 5.3). This has the benefit that focal points can be represented explicitly and shared globally. On the flipside, smoother and locally varying illumination cannot be captured. In the following, we detail the underlying model of our distribution and how focal regions are identified.

Spatial densities The goal of path guiding in forward path tracing is to reconstruct a desired target density $p_d(\omega_i \mid x)$ from which a new direction ω_i should be sampled at each intersection x. Like many previous works, we drop the dependency on the outgoing direction ω_o, which eases training and reduces the computational cost. Our directional distribution p_d is implicitly defined through a spatial density $p_s(y)$, where the resulting direction ω_i points from the current intersection x towards y

$$\omega_i = \frac{y - x}{\|y - x\|}. \tag{5.1}$$

The corresponding PDF can be found by integrating along all y that could have produced ω_i

$$p_d(\omega_i \mid x) = \int_0^\infty p_s(x + t\omega_i)\, t^2 \, \mathrm{d}t, \tag{5.2}$$

where the t^2 results from the change of variables from y to ω_i. Note that while y lies on the sampled ray, the next vertex of the path is still found through ray tracing and does not need to coincide with y, it can also lie before or behind y.

Discretization To represent our spatial density, we use adaptive trees inspired by Müller et al. [2017]. To this end, we use voxels v to partition the space of interest V (e.g., the scene bounding box) into disjunct regions $V_v \subset V$. Each region is characterized by constant density $p_s(x) = p_{s,v} \forall x \in V_v$, leaving us with a piece-wise constant spatial density. All points outside the region of interest have zero sampling density. To sample from this density, we first sample a voxel v with probability α_v and then sample a point uniformly within its volume $p(y \mid v) = |V_v|^{-1}$. The voxel sampling probabilities are linked to the guiding density

$$\alpha_v = |V_v|\, p_{s,v}. \tag{5.3}$$

Discretized directional density Our resulting directional density is found by summing the probabilities from all voxels

$$p_d(\omega_i \mid x) = \sum_v \alpha_v p_v(\omega_i \mid x), \tag{5.4}$$

where we analytically integrate Equation (5.2) within each voxel

$$p_v(\omega_i \mid x) = \begin{cases} \frac{t_1^3 - t_0^3}{3|V_v|} & \text{if intersected} \\ 0 & \text{else.} \end{cases} \tag{5.5}$$

Here, t_0 and t_1 denote the distances of entry and exit of the ray as it intersects the region V_v.

Our heuristic density To find how the focal points are distributed, we will make use of their definition: A continuum of paths that participate in a focal effect must all meet in the same point to make a contribution to the image. We say a path meets in a point if the point lies on any of the lines that constitute the path, even if it lies behind an intersection or before the ray origin. To characterize focal points, we introduce the point-wise contribution integral

$$\mathscr{F}(p) = \frac{1}{N_{\text{px}}} \sum_{\text{px}} \int_{\mathcal{P}(p)} f_{\text{px}}(\bar{x})\, \mathrm{d}\bar{x}, \tag{5.6}$$

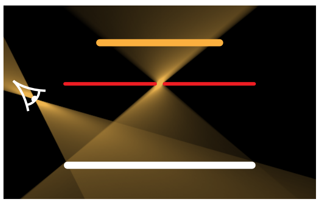
where $\mathcal{P}(p) \subseteq \mathcal{P}$ is the space of all paths that meet in p. This integral is viewpoint-aware by design – we sample focal points proportional to their image contribution instead of their raw light intensity. This prioritizes focal points that have a high impact on the image and avoids those that are not visible at all. An example of this integral is illustrated in Figure 5.4a. Since many paths meet in focal points, they cause peaks in the integral $\mathscr{F}(p)$, making it a good initial guess for our guiding density

$$\alpha_v \propto \int_{V_v} \mathscr{F}(p)\, \mathrm{d}p. \tag{5.7}$$

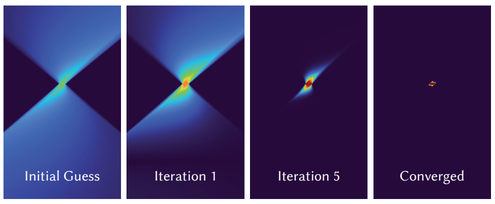
Here, we set the selection probability of a voxel v proportional to the contribution of all paths that pass through the voxel.

Estimating our density An estimate $\widetilde{\alpha}$ of our density can be obtained simply by logging the contribution of paths as they are traced:

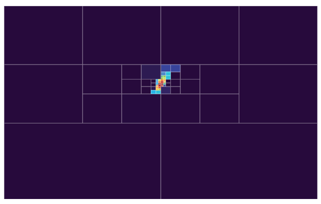
$$\widetilde{\alpha}_v \propto \sum_{\text{px}}^{N_{\text{px}}} \sum_{i=1}^{N_{\text{spp}}} \sum_{j=2}^{n(\bar{x}_i)} \begin{cases} (t_1 - t_0) \frac{f_{\text{px}}(\bar{x}_i)}{p(\bar{x}_i)} & \text{if intersected} \\ 0 & \text{else,} \end{cases} \tag{5.8}$$



(a) *Point-wise contribution integral*



(b) *Our iterative narrowing scheme*



(c) *Adaptive octrees*

Figure 5.4: *(a) We show the point-wise contribution integral of a diffuse surface (white line) illuminated by a large area light (orange line) through a narrow gap (occluder in red). The camera- and occlusion focal points cause peaks in the integral, as many paths meet in the same points. (b) Excluding segments that cannot be guided (e.g., camera segments), this serves as initial guess for our guiding density, which is then iteratively refined to avoid sampling points that are only relevant to subsets of the focal effect. (c) We represent the guiding density using adaptive octrees.*

where we collect the contribution from all path segments j that have been sampled. The values t_0 and t_1 denote the entry and exit distances of the ray $x_i^{j-1} \to x_i^j$ in voxel v, and the multiplication with $t_1 - t_0$ results from the integral over the region of the voxel.

Spurious focal points Focal effects are seldom singular points in practice, but rather small regions of finite extent. Unfortunately, this creates smearing in the point-wise contribution integral. Consider Figure 5.4a: The region within the narrow gap is sufficient to explore the full focal effect, as all paths need to cross it. But the points above and below the narrow gap also have a high value, as subsets of the paths pass them. We refer to these points as *spurious* focal points. They are detrimental to the quality of our sampling as they only benefit subsets of the focal effect, but take away probability mass that could be invested in true focal points benefiting all paths.

Iterative narrowing To narrow our distribution down to true focal points, we introduce a scheme that we refer to as *iterative narrowing*. Given a previous distribution estimate $\widetilde{\alpha}$, we obtain a narrowed distribution $\widetilde{\alpha}^N$ by weighting the contributions of paths:

$$\widetilde{\alpha}_v^N \propto \sum_{\text{px}}^{N_{\text{px}}} \sum_{i=1}^{N_{\text{spp}}} \sum_{j=2}^{n(\bar{x}_i)} w_v(x_i^{j-1}, x_i^{j-1} \to x_i^j) \frac{f_{\text{px}}(\bar{x}_i)}{p(\bar{x}_i)}, \tag{5.9}$$

where the weight w_v is given by

$$w_v(x, \omega_i) = \frac{\widetilde{\alpha}_v p_v(\omega_i \mid x)}{\sum_{v'} \widetilde{\alpha}_{v'} p_{v'}(\omega_i \mid x)}. \tag{5.10}$$

Instead of equally contributing to all intersected voxels, iterative narrowing weights the contribution to voxels by how likely the voxel samples the path segment. Since spurious focal points only capture subsets of focal effects, fewer paths contribute to them than to the true focal point, leading to lower selection probabilities. Iterative narrowing then further lowers their value by assigning lower weights to the contribution from path segments. Applying this repeatedly effectively eliminates spurious focal points (Fig. 5.4b).

Diverging focal points So far, we have glanced over the fact that focal points can also lie behind the surface we are currently sampling from. Supporting this is straightforward. We introduce additional voxels v^- that also partition the region of interest, but produce directions that point *away from* the sampled point y^-:

$$\omega_i = -\frac{y^- - x}{\|y^- - x\|}. \tag{5.11}$$

These voxels are considered intersected (for logging contributions or PDF computations) if they lie in negative ray direction.

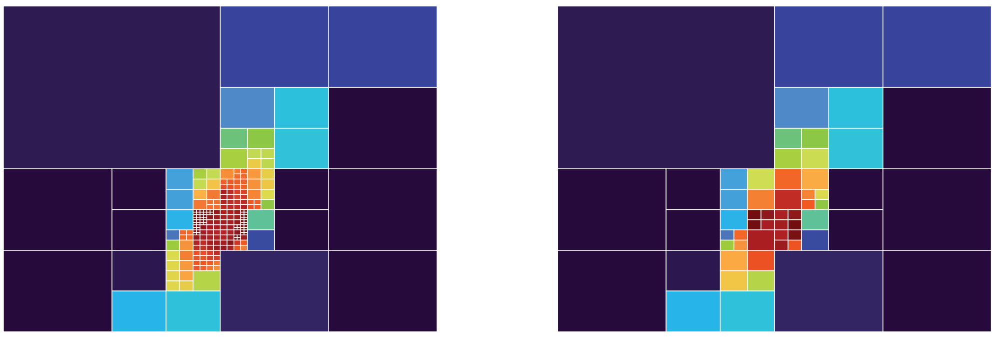
5.3 Implementation

We implement our method as a guided forward path tracer in Mitsuba [Jakob 2010]. Similar to previous guiding works, our algorithm proceeds in iterations. Each iteration refines the density from the previous iteration. The last iteration renders the final image. In the following, we detail the exact iteration schedule, data structure and parameters used by our approach.

Schedule While it is possible to continue training throughout the render, there is usually a point where the diminishing returns of additional training no longer justify the overhead of logging contributions and updating data structures. Determining the optimal amount of training is still an open problem in path guiding, hence we follow the compromise of Müller et al. [2017] and split the total budget into 50% training and 50% final render. The training phase is further subdivided into iterations, which each train an improved density by sampling from the density learned in the previous iteration. We find that iterations of equal length [Ruppert et al. 2020] perform better than iterations of increasing duration [Müller et al. 2017], as they accelerate learning of rare effects and lend themselves nicely to our iterative narrowing scheme. The training budget is split equally into n_{iter} iterations, for which we find a choice of $n_{\text{iter}} = 15$ to work well across all tested scenes.

Burn-in period Challenging focal effects can take several iterations before they are discovered. Before enabling iterative narrowing (Section 5.2), we need to be confident that all relevant focal points have been identified. Otherwise, once a valid path is finally discovered, its focal region might already be considered spurious due to its previously low value. While iterative narrowing can eventually recover from such cases, it can take several iterations for a low-scoring region to be considered focal again. To this end, we only enable iterative narrowing in the last five training iterations, which we find sufficient to dismiss most spurious focal points (see Figure 5.4b). A more sophisticated remedy to handling rare events could rely on Bayesian priors [Vévoda et al. 2018; Dodik et al. 2022], which we leave as future work.

Data structure We use an adaptive octree to represent our guiding density (see Figure 5.4c). The region of the root node is set to the bounding box of the scene. Inspired by Müller et al. [2017], a node is split when its selection probability α_v exceeds a given splitting threshold γ. This threshold is the main parameter of our technique: A lower value yields higher resolution, but requires more samples to fit and increases the cost of PDF computations. We observe that a value of $\gamma = 10^{-3}$ works well across all our test scenes (see Section 5.4.1).



(a) *Before pruning* **(b)** *After pruning*

Figure 5.5: *After the last training iteration, we collapse octree nodes that have little variation. This reduces the cost of the traversal required to compute probability densities.*

Pruning The increased resolution of splitting does not always justify the penalty on traversal cost, in particular when there is little variation within a node. To mitigate this, we use a simple pruning heuristic (see Figure 5.5): We collapse each node where the highest leaf density does not exceed twice the average density of the node. We notice that a fine subdivision is still beneficial for learning, so we prune only after the last training iteration is completed.

Sampling We obtain spatial samples $y \sim p_s$ using hierarchical sample warping [McCool and Harwood 1997]. The direction vector ω_i follows from Eq. (5.1). For the probability density (Eq. (5.4)), it suffices to sum up the voxels intersected by the ray, since all other voxels have zero probability density of producing the sample. For this, we use the traversal algorithm by Revelles et al. [2000].

Multiple importance sampling While our guiding density excels at sampling focal effects, its performance on other light transport can be poor. It is therefore advisable to combine it with other sampling strategies using multiple importance sampling [Veach and Guibas 1995b]. Like Müller et al. [2017], we perform mixture sampling of BSDF and guiding at a fixed ratio of $\lambda_B = 0.5$. Similarly, we also perform next event estimation. No guiding is performed on specular surfaces, and accordingly we do not log segments that cannot be guided.

MIS compensation Path guiding works best when it learns to augment rather than replace other sampling strategies. This is known as MIS compensation [Karlík et al. 2019]: We want guiding to only learn effects not handled well by other sampling strategies in our MIS combination. To compensate for BSDF sampling, we use the scheme proposed by Rath et al. [2020], which – similar to our iterative narrowing scheme – iteratively weights the guiding distribution by how likely it is to sample an effect. Applied to our iterative narrowing scheme (Equation (5.9)), our weights become

$$w_v(x, \omega_i) = \frac{(1 - \lambda_B)\alpha_v p_v(\omega_i \mid x)}{\lambda_B p_B(\omega_i \mid x, \omega_o) + (1 - \lambda_B) \sum_{v'} \alpha_{v'} p_{v'}(\omega_i \mid x)}, \tag{5.12}$$

where p_B is the density of BSDF sampling and λ_B its selection probability. We additionally multiply the contribution by the NEE MIS weight, as done by Ruppert et al. [2020].

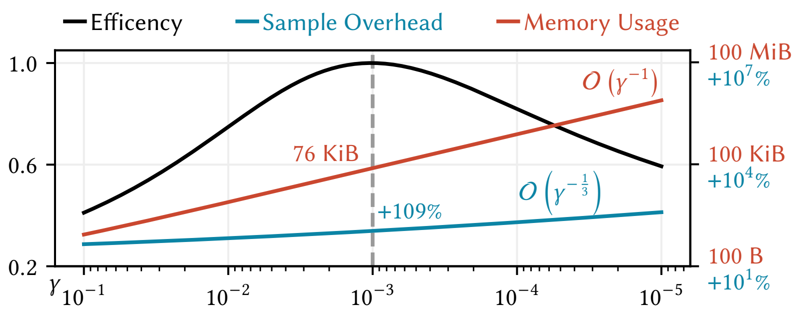


Figure 5.6: *We evaluate our splitting threshold γ. Finer resolutions (i.e., low values of γ) yield more accurate sampling, but incur higher memory usage and sample overhead relative to an unguided path tracer. The efficiency (MSE error of equal-time renders) peaks at $\gamma = 10^{-3}$.*

5.4 Evaluation

We compare the performance of our method for various focal effects against the following methods (Figure 5.7):

- PT: Forward path tracing with next event estimation [Kajiya 1986]
- MEMLT: Manifold-exploration MLT [Jakob and Marschner 2012]
- MCVCM: Metropolised vertex connection and merging [Šik et al. 2016]
- PAVMM: Parallax-aware mixtures for path guiding [Ruppert et al. 2020]

All techniques are implemented in Mitsuba [Jakob 2010]. PAVMM and our approach use half of the time budget for learning. Russian roulette is an orthogonal problem in path guiding, as guided paths often have low throughput until they reach a light source [Vorba and Křivánek 2016; Rath et al. 2022a], hence we disable Russian roulette in all techniques for comparability. In addition to our renders, we demonstrate the benefits of our approach over path space guiding [Reibold et al. 2018] in a 2-D experiment.

All evaluations were run on a 16-core AMD Ryzen™ 9 3950X processor with 64 GB of memory. All renders are rendered for approximately three minutes. We provide the mean squared error (MSE), for which we clamp outliers at the 99.9% percentile to arrive at a stable value, and ꟻLIP metric [Andersson et al. 2021].

5.4.1 Splitting threshold

We begin by analyzing the impact of the main parameter of our approach: the splitting threshold γ, which determines the resolution of our adaptive octree. We evaluate its impact on computational overhead and memory usage, averaged over three-minute renders of our five test scenes, in Figure 5.6. Finer resolutions result in more accurate sampling, at the expense of higher sample cost. The best efficiency across all scenes is achieved at a value of $\gamma = 10^{-3}$, but the performance of our algorithm is not particularly sensitive to its exact value. At this threshold, samples are roughly twice as expensive as unguided sampling, and the memory overhead of our method amounts to a mere 76 KiB.

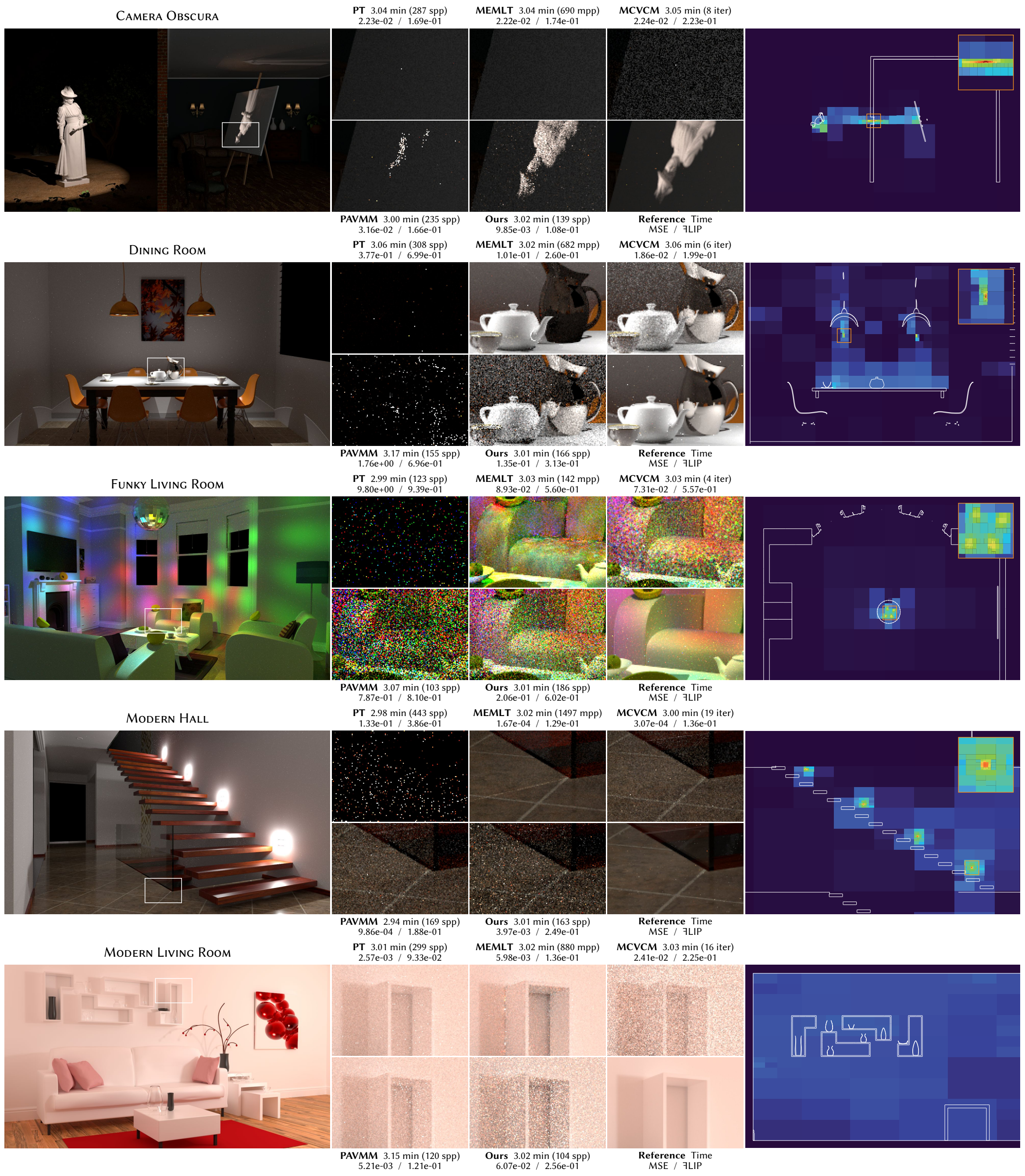

Camera Obscura
PT 3.04 min (287 spp) 2.23e-02 / 1.69e-01
MEMLT 3.04 min (690 mpp) 2.22e-02 / 1.74e-01
MCVCM 3.05 min (8 iter) 2.24e-02 / 2.23e-01
PAVMM 3.00 min (235 spp) 3.16e-02 / 1.66e-01
Ours 3.02 min (139 spp) 9.85e-03 / 1.08e-01
Reference Time MSE / ꟻLIP
Dining Room
PT 3.06 min (308 spp) 3.77e-01 / 6.99e-01
MEMLT 3.02 min (682 mpp) 1.01e-01 / 2.60e-01
MCVCM 3.06 min (6 iter) 1.86e-02 / 1.99e-01
PAVMM 3.17 min (155 spp) 1.76e+00 / 6.96e-01
Ours 3.01 min (166 spp) 1.35e-01 / 3.13e-01
Reference Time MSE / ꟻLIP
Funky Living Room
PT 2.99 min (123 spp) 9.80e+00 / 9.39e-01
MEMLT 3.03 min (142 mpp) 8.93e-02 / 5.60e-01
MCVCM 3.03 min (4 iter) 7.31e-02 / 5.57e-01
PAVMM 3.07 min (103 spp) 7.87e-01 / 8.10e-01
Ours 3.01 min (186 spp) 2.06e-01 / 6.02e-01
Reference Time MSE / ꟻLIP
Modern Hall
PT 2.98 min (443 spp) 1.33e-01 / 3.86e-01
MEMLT 3.02 min (1497 mpp) 1.67e-04 / 1.29e-01
MCVCM 3.00 min (19 iter) 3.07e-04 / 1.36e-01
PAVMM 2.94 min (169 spp) 9.86e-04 / 1.88e-01
Ours 3.01 min (163 spp) 3.97e-03 / 2.49e-01
Reference Time MSE / ꟻLIP
Modern Living Room
PT 3.01 min (299 spp) 2.57e-03 / 9.33e-02
MEMLT 3.02 min (880 mpp) 5.98e-03 / 1.36e-01
MCVCM 3.03 min (16 iter) 2.41e-02 / 2.25e-01
PAVMM 3.15 min (120 spp) 5.21e-03 / 1.21e-01
Ours 3.02 min (104 spp) 6.07e-02 / 2.56e-01
Reference Time MSE / ꟻLIP

Figure 5.7: *We compare the performance of our approach against previous works on five challenging test scenes. Each method renders for approximately three minutes. The first four scenes demonstrate various focal effects, while the last scene exhibits primarily afocal light transport. A slice through our spatial guiding density is displayed on the right side, overlaid by the corresponding slice of the scene geometry for orientation (white lines).*

5.4.2 Scenes

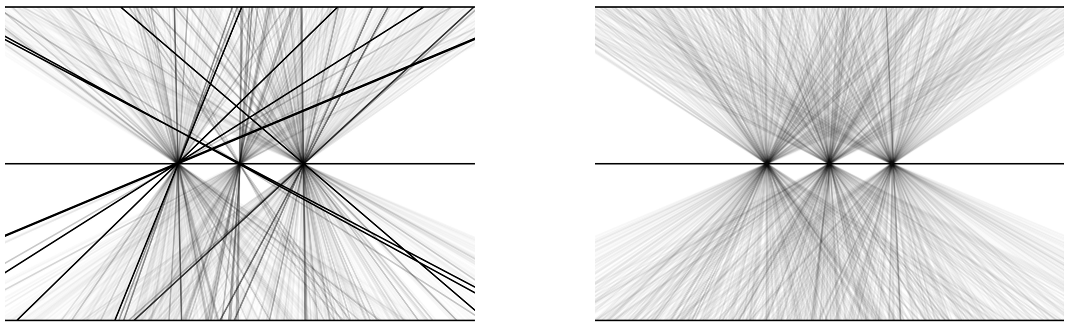
Camera Obscura The Camera Obscura scene features an occlusion focal point in form of a pinhole in the wall, which projects the strongly illuminated statue on the left onto the canvas on the right. Path tracing struggles with this scene, as uninformed sampling is unlikely to find the pinhole. MEMLT fails to find a seed path that passes the pinhole and hence completely misses the projection of the statue. The metropolised light tracing of MCVCM fails for the same reason, but as with PT, a few rare eye paths manage to pass through the pinhole. Parallax compensation (PAVMM) is of little help here, as it aims rays at surface points of the statue rather than the pinhole that is causing the focal effect. Our approach identifies the pinhole as focal region and systematically constructs paths that pass through it, resulting in significantly faster convergence than previous state-of-the-art algorithms.

Dining Room The parabolic lampshades in our version of the Dining Room scene create a virtual image of a tiny light source. Path tracing struggles to find paths that pass the focal region. Based on BDPT, both MEMLT and MCVCM successfully construct light subpaths that pass through the virtual light focal point. The specular jug on the table, however, introduces a specular-diffuse-specular (SDS) path that prohibits the connection of eye and light subpaths. MCVCM solves this issue through vertex merging, introducing bias in the process. MEMLT can reliably explore SDS paths, but again fails to find a valid subpath to initiate the exploration. Without light tracing (PT, PAVMM, and Ours), paths that pass through the virtual image need to be found explicitly, which only our method can sample systematically.

Funky Living Room The Funky Living Room scene includes a glossy disco ball illuminated by five spot lights. Path tracing struggles to find the correct facet on the disco ball connecting the path to a spot light. MEMLT produces noticeable artefacts due to uneven convergence, while MCVCM exhibits density estimation artefacts. PAVMM learns to guide towards the disco ball, but mispredicts the parallax distance of the spotlights. Our approach learns that it is best to aim rays at the interior of the disco ball, in which all spot lights converge.

Modern Hall The Modern Hall scene is illuminated by four tiny area lights enclosed in translucent boxes, which constitute diffusing focal points. This effectively prevents next event estimation and makes the technique difficult for forward unidirectional techniques. For bidirectional methods like MEMLT and MCVCM, however, this scene is fairly simple to render. As far as unidirectional methods go, PAVMM performs exceptionally well in this scene, as it can correctly compensate the parallax to the light source given the surface-bound nature of the focal point. While our method significantly improves over PT, it falls short of the performance of PAVMMs as it cannot account for occlusion or the falloff of light over distance.

Modern Living Room This scene is illuminated by a large light source just out of frame, which is handled very well by next event estimation. The path tracer remains unbeaten in this scene, as the overhead of more sophisticated methods does not pay off in this simple scene. Our approach faces an additional problem: Since it cannot represent smoothly varying illumination, our guiding degrades the quality of sampling in this scene. This is expected, as our guiding is specifically tailored to handled focal effects, which are absent from this scene.



(a) *Sampling complete light paths [Reibold et al. 2018]* **(b)** *Sampling a global spatial density (our approach)*

Figure 5.8: *We compare the performance of local versus global sampling distributions in a simplified 2-D experiment. Paths originating at the floor need to pass an obstacle with three pinholes to reach the light source at the top. The opacity of paths is proportional to the value of its estimator, i.e., darker paths represent outliers which are sampled insufficiently. While both approaches can reliably sample focal points once they are identified, local methods (a) are prone to overlooking parts of focal effects, such as the contribution from the center focal point on the right side of the floor. This results in uneven sampling, which manifests itself as islands of high variance in the rendered image. Global distributions like ours (b) do not suffer from this issue: Once focal points are identified, all regions exploit this information. This results in more even sampling, eliminating residual outliers.*

5.4.3 Comparison with path space guiding

In Figure 5.8 we compare the behavior of path space guiding [Reibold et al. 2018] and our approach in a simplified 2-D experiment. Paths originating at a plane need to pass through an obstacle with three narrow gaps to reach a large area light at the top. Both methods were trained for 6 iterations with 4096 samples each.

In path space guiding (Fig. 5.8a), directions are sampled according to statistics over earlier paths in the vicinity of the current vertex. This makes the method inherently local, as individual focal points must be re-discovered for each region, which is prone to overlooking parts of focal effects due to the sparsity of constructing valid focal paths through exploration. In our example, the right side of the floor has not found sufficient guide paths to sample the center focal point, which manifests itself in residual outliers and islands of high variance.

Our approach shares information globally (Fig. 5.8b), which allows the entire floor to benefit from robust sampling once a focal region has been identified. The price for this is that local effects are handled poorly by our approach: Occlusion, light falloff over distance, and anisotropic effects are handled much better by local methods, which can learn different distributions for different regions of space.

5.4.4 Overhead of our technique

We break down the computational overhead of our technique in Table 5.1. While sampling our density is comparatively cheap, since only a single stochastic tree traversal is required, the bottleneck lies in evaluating the probability density required for MIS. Pruning after the last training iteration (Section 5.3) reduces the costs drastically in the final rendering iteration.

Function	Computational cost	
	Training	Render
Ray tracing	51%	
Next event estimation	40%	
BSDF operations	2%	
Miscellaneous	7%	
Guiding PDF computation	+144%	+78%
Guiding learning	+34%	–
Guiding sampling	+8%	+10%
$\sum$	286%	188%

Table 5.1: *Computational cost of our method compared to unguided forward path tracing in the* Camera Obscura *scene. Our sampling is cheap, but the PDF computation is costly as it requires a ray traversal of the octree. While pruning only discards 9% of the nodes, the traversal time is halved.*

Algorithm 2: *Pseudo-code of our implementation. For brevity, we omit next event estimation.*

```
1: function Render
2:     for $i \in 1..n_{\text{iter}}$ do
3:         $\hat{\alpha} \leftarrow \mathbf{0}$                                          ← reset accumulators
4:         RenderSeconds(TrainingBudget$/n_{\text{iter}}$)
5:         RefineAndNormalizeOctree()
6:     PruneOctree()
7:     return RenderSeconds(RenderBudget)
8: function SampleGuiding($x, \omega_o$)
9:     $v \leftarrow$ SampleDistribution($\alpha$)
10:    $y \leftarrow$ SampleUniformlyInVolume($V_v$)
11:    return Normalize($y - x$)
12: function PdfGuiding($\omega_i \mid x, \omega_o$)
13:    $p \leftarrow 0$                                    ↙ Sum up all voxels in direction $\omega_i$
14:    for $(v, t_0, t_1) \in$ OctreeTraversal($x, \omega_i$) do
15:        $p \leftarrow p + \alpha_v * (t_1^3 - t_0^3)/(3 * V_v)$
16:    return $p$
17: function $L_o(x, \omega_o, T)$
18:    $\omega_i, p_{\omega_i} \leftarrow$ MixtureSample $[\alpha_{\text{BSDF}}]$ $(x, \omega_o)$
19:    $B \leftarrow \text{BSDF}(x, \omega_i, \omega_o)/p_{\omega_i}$
20:    $\langle L_i \rangle \leftarrow L_o(\text{RT}(x, \omega_i), \omega_o, T * B)$          ↙ Accumulate completed paths
21:    for $(v, t_0, t_1) \in$ OctreeTraversal($x, \omega_i$) do
22:        if BurninPeriod then
23:            $w_v \leftarrow t_1 - t_0$             ↰ We wait until $\alpha$ is reliable enough for narrowing
24:        else
25:            $w_v \leftarrow ((1 - \alpha_{\text{BSDF}}) * \alpha_v * (t_1^3 - t_0^3)/(3 * V_v))/p_{\omega_i}$
26:        $\hat{\alpha}_v \leftarrow \hat{\alpha}_v + w_v * T * B * \langle L_i \rangle$
27:    return $L_e(x, \omega_o) + B * \langle L_i \rangle$
```

5.5 Limitations and future work

We have shown a simple method to identify and sample focal points in a forward path tracer. We believe that there are many promising ways in which our work could be continued:

Augmenting other techniques While our technique excels at focal effects, its performance on general light transport is limited. We believe its best use is to augment other techniques. For instance, it could identify light portals [Bitterli et al. 2015; Ogaki 2020] or learn distances for parallax compensation [Ruppert et al. 2020].

Region of interest We currently assume that focal points lie within the bounding box of the scene, which is not necessarily true for virtual images. It could be interesting to automatically extend the region of interest when necessary. An alternative approach could be to use a representation that covers all of space, potentially with reduced precision for increasing distances.

Handling local effects Our sampling scheme assumes that focal points contribute equally throughout the scene, ignoring both visibility and the falloff of light intensity over distance. A promising remedy lending itself well to our approach is the voxel-based learning of visibility by Guo et al. [2020].

Alternative representations Our implementation relies on adaptive trees [Müller et al. 2017] to represent the focal guiding density. An interesting alternative would be to fit parametric mixture models [Vorba et al. 2014; Ruppert et al. 2020; Dodik et al. 2022] or neural networks [Müller et al. 2019], which could enable product sampling (to handle local effects) or reduce sample overhead.

Variance-aware target density Crafting densities that explicitly minimize image variance can greatly increase the rate of convergence [Rath et al. 2020]. Our guiding scheme learns selection probabilities of a mixture density, in which each voxel is a unique sampling strategy. Unfortunately, finding variance-optimal selection probabilities remains an unsolved problem [Lu et al. 2013; Sbert et al. 2018]. Stochastic gradient descent could be a promising tool to optimize our voxel selection probabilities [Müller et al. 2017].

6

Efficiency-Aware Russian Roulette and Splitting

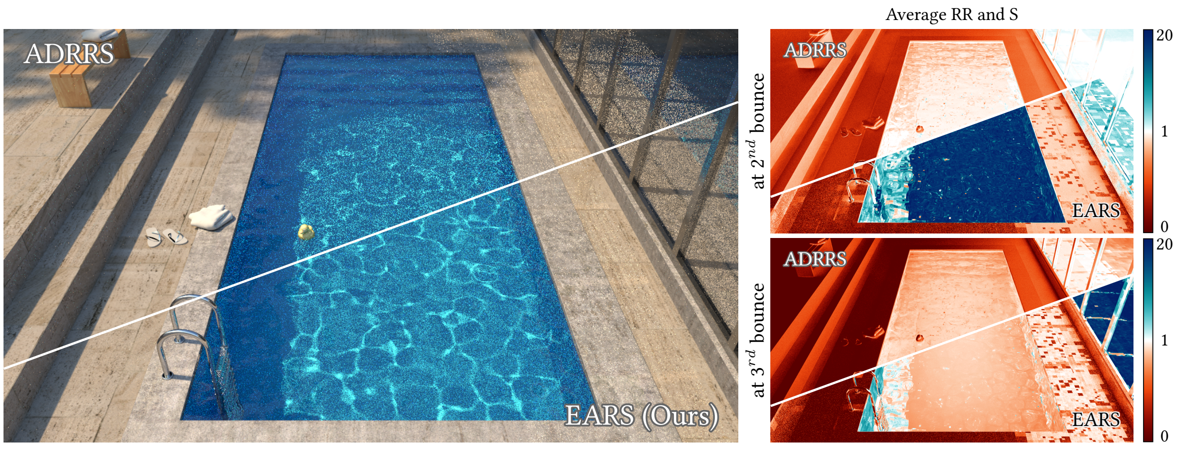


Figure 6.1: *We derive a fixed-point iteration to compute the Russian roulette and splitting (RRS) factors that maximize the rendering efficiency. Here, we compare the rendered images in equal-time (10 min) of our method and the state of the art,* adjoint-driven Russian roulette and splitting *(ADRRS) [Vorba and Křivánek 2016]. The false-color images on the right visualize the average RRS factors in each pixel at the second and third bounce. Red indicates that mostly roulette is played; blue indicates that mostly splitting is done. By directly optimizing variance and cost, our method produces more efficient RRS decisions: splitting is mostly performed at the bottom of the pool and on the diffuse surface behind the window, which dominate the image variance.*

Russian roulette and splitting are ubiquitous methods to increase the efficiency of algorithms based on path tracing. Unlike importance sampling methods such as path guiding, which attempt to increase efficiency by reducing the variance of the primary estimator, Russian roulette and splitting focus on trading off computational cost and variance across different scene regions by replacing the primary estimators with secondary estimators with carefully chosen sample counts given the path prefix at hand. For example, Russian roulette reduces costs by stochastically terminating paths that are expected to have a low contribution to the image, which, if done right, only leads to a slight overall increase in variance. Splitting, on the other hand, reduces the variance of a local estimate by continuing a prefix path with multiple suffix paths. If most of the variance is due to that local sampling decision, splitting greatly reduces the variance without incurring the cost of repeatedly sampling the prefix path.

Despite its prevalence, optimally performing Russian roulette and splitting (RRS) has received little attention. While the work of Bolin and Meyer [1997] derives RRS factors that result in optimal efficiency, their method remains unpractical and restricts RRS factors to only depend on the originating pixel and current path length. Succeeding works have shifted away from the idea of optimizing efficiency in favor of choosing RRS factors freely. For example, the current state-of-the-art, adjoint-driven RRS [Vorba and Křivánek 2016], bases decisions on estimates of the expected image contribution. While producing great results, the expected contribution does not consider variance or cost of local estimators.

We combine the benefits of previous approaches by proposing a practical method to choose RRS factors locally while optimizing efficiency globally. The difference is shown in Fig. 6.1. Our method encourages splitting when reaching the caustics at the bottom of the pool and behind the windows as they have high variance and highly benefit from splitting. On the other hand, by only considering the expected contribution of the local estimator and not its variance, ADRRS only ensures the survival of these paths but does not perform any splitting. This is visible in the false-color images on the right, comparing the average per-pixel RRS factors for both methods at different depths.

We begin by deriving a fixed-point iteration to numerically find RRS factors that maximize efficiency, and prove that it converges (Section 6.1). Applied to rendering, we introduce an online learning scheme that iteratively improves the RRS factors over time (Section 6.2). In practice, we achieve consistent speed-ups over a large variety of scenes (Section 6.3).

The method introduced in this chapter has been published before [Rath et al. 2022a], of which I am the main author. Our implementation of the proposed algorithm is available at https://github.com/iRath96/ears [Rath et al. 2022b].

6.1 Efficiency-aware RRS

We determine the optimal RRS factors that, given a prefix path $\bar{\mathbf{x}}_k$, decide if and with how many samples to continue the path in order to maximize rendering efficiency. To this end, we show how the resulting equations can be efficiently solved using a fixed-point iteration that converges to the optimal RRS factors. In the following, we first introduce the method in a simplified setting, then show how it can be applied in a rendering context.

Without loss of generality, we first consider a 2D integration problem, illustrated in Fig. 6.2. The integrand is modeled after the rendering equation and consists of the product of two functions, the prefix $g(x)$ and suffix $h(x, y)$,

$$I = \int_{\mathcal{X}} g(x) \int_{\mathcal{Y}} h(x, y) \, \mathrm{d}y \, \mathrm{d}x =: \int_{\mathcal{X}} g(x) \, H(x) \, \mathrm{d}x. \tag{6.1}$$

Given a prefix x sampled with density $p(x)$, the goal is to find the optimal number of splits, or the optimal Russian roulette (RR) probability, to continue the 'path' by sampling suffixes y_i.

We start by deriving a fixed-point iteration to compute the optimal splitting factors and prove that it converges. Then, we show that the same approach can be applied to RR, as well as a combined RRS.

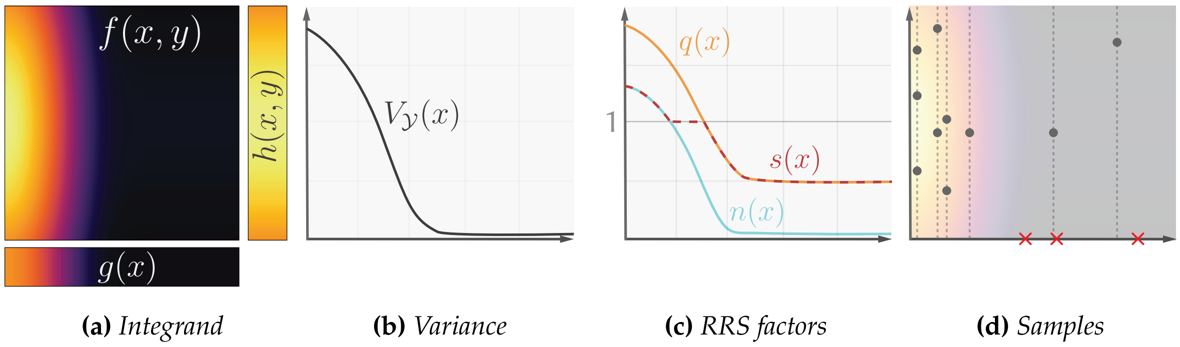


Figure 6.2: *Russian roulette and splitting in a simplified 2D example. (a) the integrand $f(x, y)$ is the product of two functions, $g(x)$ and $h(y)$. (b) the variance is highest in the high-contribution region on the left. (c) the optimal RR, splitting, or combined RRS factor for each prefix x, computed via our fixed-point iteration. (d) when using the optimized RRS factors, samples in the low-variance region are terminated (red crosses), while samples in the high-variance region are split (one batch of samples per dashed line).*

6.1.1 Optimal splitting

A nested splitting estimator $\langle I; n\rangle$ for I takes a single prefix sample $x \in \mathcal{X}$ and combines it with $n(x)$ suffix samples $y_j \in \mathcal{Y}$:

$$\langle I; n\rangle = \frac{g(x)}{p(x)} \sum_{j=1}^{n(x)} \frac{h(x, y_j)}{n(x)\, p(y_j \mid x)}. \tag{6.2}$$

For optimal splitting at x, we need the optimal *splitting function* $n(x) : \mathcal{X} \to \mathbb{N}$, which gives us a positive integer splitting count for each prefix x.

Objective

Efficiency is maximized when the product of variance and cost is minimized,

$$\arg\min_n \mathbb{V}\left[\langle I; n\rangle\right] \mathbb{C}\left[\langle I; n\rangle\right]. \tag{6.3}$$

Assuming that the suffix samples y_i are uncorrelated, we can use the law of total variance to express the variance of Eq. (6.1) as the sum of two components [Bolin and Meyer 1997]

$$\mathbb{V}\left[\langle I; n\rangle\right] = V_\mathcal{X} + \mathbb{E}\left[\frac{V_\mathcal{Y}(x)}{n(x)}\right], \tag{6.4}$$

where

$$V_\mathcal{X} := \mathbb{V}\left[\frac{g(x)}{p(x)} H(x)\right] \tag{6.5}$$

is the variance due to sampling the prefix x, if given a ground truth value $H(x)$ for the nested integral, and

$$V_\mathcal{Y}(x) := \mathbb{V}\left[\langle I\rangle \mid x\right] = \left(\frac{g(x)}{p(x)}\right)^2 \mathbb{V}\left[\langle H(x)\rangle \mid x\right] \tag{6.6}$$

is the variance, without any splitting, due to sampling a single suffix y when given a prefix x.

The cost is modeled by the cost operator $\mathbb{C}[\cdot]$,

$$\mathbb{C}[\langle I; n\rangle] = \mathbb{C}[x] + n(x)\,\mathbb{C}[\langle H(x)\rangle]\,, \tag{6.7}$$

where we assume that the cost is linear in the number of samples, i.e., it can be split into a sum of the cost $\mathbb{C}[x]$ due to sampling of the prefix, and the cost $\mathbb{C}[\langle H(x)\rangle]$ due to sampling the suffix.

Optimization

To solve Eq. (6.3), we introduce an approximation: We pretend that the splitting factors do not have to be integers. That is, we allow $n(x) : \mathcal{X} \to \mathbb{R}^+$. This allows us to compute derivatives and perform convex optimization, but it also introduces a small error, since the result has to be rounded to a positive integer, either exactly or stochastically. The result then has a slightly different variance than predicted during the optimization[1].

For a real-valued splitting function, the efficiency is a convex functional of $n(x)$, as it can be written as a sum of convex functions, which is by definition also convex. Hence, it must have a unique global minimum. The question is, how can we find it efficiently?

In principle, we can set the partial functional derivatives to zero,

$$\frac{\mathrm{d}\mathbb{V}[\langle I; n\rangle]\,\mathbb{C}[\langle I; n\rangle]}{\mathrm{d}n(x)} = 0, \tag{6.8}$$

and solve the resulting system of equations. The derivative is easy enough to compute analytically, as shown in Appendix B.1, but analytically solving the system of equations is not practical, at least not in a full rendering context. Aside from requiring some integrals that are challenging to estimate, the problem is also a recursive one, as splitting can happen at other points along the prefix and suffix.

Fixed-point iteration

Our solution is to numerically find the solution to Eq. (6.8) via a fixed-point iteration. A primer on using fixed-point iterations for root finding, and proving their convergence, can be found in Appendix B.2. The key idea is that, instead of solving (6.8) directly, we formulate a fixed-point iteration

$$n_i(x) = \gamma_{\mathrm{S}}(n_{i-1}(x)) = \sqrt{\frac{V_{\boldsymbol{y}}(x)}{\mathbb{V}[\langle I; n_{i-1}\rangle]}\,\frac{\mathbb{C}[\langle I; n_{i-1}\rangle]}{\mathbb{C}[\langle H(x)\rangle \mid x]}}, \tag{6.9}$$

where the ith iteration computes the splitting factor $n_i(x)$ based on the variances and costs of the previous iteration's configuration. In Appendix B.3, we show that the unique fixed-point $\gamma_{\mathrm{S}}(n(x)) = n(x)$ is the optimal $n(x)$ and prove that iteratively applying $n_i(x) = \gamma(n_{i-1}(x))$, starting with an arbitrary initial guess $n_0(x)$, converges to the optimum.

[1] This approximation error is also present in the previous work of Bolin and Meyer [1997].

6.1.2 Incorporating Russian roulette

A very similar fixed-point iteration can be derived to find the optimal Russian roulette (RR) probability $q(x)$. The result can be combined with the optimal splitting $n(x)$ into an optimal RRS decision $s(x)$.

Optimal RR

First, consider the case where instead of splitting, only RR is allowed. Performing RR at a prefix x increases the variance [Bolin and Meyer 1997],

$$\mathbb{V}_{\mathrm{RR}}\left[\langle I; q\rangle\right] = V_{\mathcal{X}} + \mathbb{E}\left[\frac{M_{\mathcal{Y}}(x)}{q(x)} - \left(\frac{g(x)}{p(x)} H(x)\right)^2\right], \tag{6.10}$$

where the suffix moment

$$M_{\mathcal{Y}}(x) := \left(\frac{g(x)}{p(x)}\right)^2 \mathbb{E}\left[\langle H(x)\rangle^2 \mid x\right] \tag{6.11}$$

is the expectation of the squared estimator value, given the prefix x.

The effect on our fixed-point function (6.9) is rather minor. If we re-compute the derivatives with the RR variance, the only change is that the suffix variance $V_{\mathcal{Y}}$ is replaced by the suffix moment $M_{\mathcal{Y}}$,

$$q_{i+1}(x) = \gamma_{\mathrm{RR}}(q_i(x)) = \sqrt{\frac{M_{\mathcal{Y}}(x)}{\mathbb{V}\left[\langle I; q_i\rangle\right]} \frac{\mathbb{C}\left[\langle I; q_i\rangle\right]}{\mathbb{C}\left[\langle H(x)\rangle \mid x\right]}}. \tag{6.12}$$

The objective is still convex, and the fixed-point iteration still converges to the unique optimum, since, from a mathematical point of view, we merely swapped one constant for another.

Optimal RRS

We can jointly optimize splitting and RR, by first combining them into a piece-wise variance

$$\mathbb{V}_{\mathrm{RRS}}\left[\langle I; s\rangle\right] = V_{\mathcal{X}} + \mathbb{E}\left[R(s(x))\right], \tag{6.13}$$

where we define

$$R(s(x)) := \begin{cases} \frac{V_{\mathcal{Y}}(x)}{s(x)} & \text{if } s(x) > 1 \\ \frac{M_{\mathcal{Y}}(x)}{s(x)} - \left(\frac{g(x)}{p(x)} H(x)\right)^2 & \text{else.} \end{cases} \tag{6.14}$$

The product of this piece-wise RRS variance and the cost (which remains unchanged) is still a convex functional in $s(x)$, as illustrated in Fig. 6.3. The figure shows examples for the three possible configurations of the joint objective. The minimum can be either the optimal RR value, the optimal splitting value, or $s(x) = 1$.

The minimum can be found with a similar approach to the one by Bolin and Meyer [1997]. First, we compute the splitting factor $n(x)$. If the result is greater than one, that is our optimum. Otherwise, we compute the RR factor $q(x)$ and clamp it to one, to handle the case

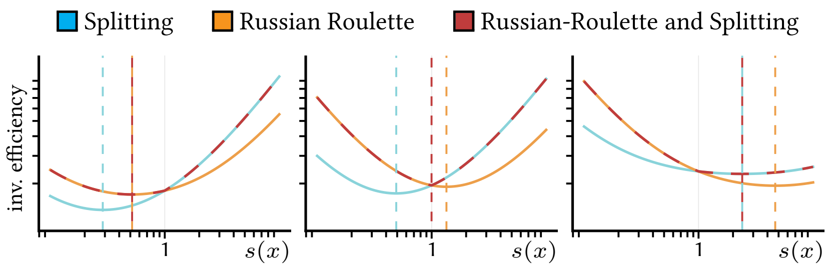


Figure 6.3: *Examples visualizing the shape of our objective function, here in 1D for a single point x with corresponding $s(x)$. The splitting (blue) and RR (orange) objectives are both convex and have a unique local minimum (vertical lines). There are three possible cases, shown from left to right: RR is optimal, doing neither is optimal, and splitting is optimal. By definition, the RR objective and the splitting objective intersect at $s(x) = 1$. The RR variance is greater than the splitting variance for $s(x) < 1$, and vice versa for $s(x) > 1$. Hence, the combined objective (dashed red curve) is also convex, and both local minima always lie in the correct portion of the domain (i.e., below or above 1). Unless 1 is the minimum, then both lie in the opposite region. These properties guarantee the convergence of the joint fixed-point iteration.*

where the optimal decision is exactly one (which is a discontinuity in our objective function). This can be written as a joint fixed-point function:

$$\gamma_{\mathrm{RRS}}(s(x)) = \begin{cases} \gamma_{\mathrm{S}}(s(x)) & \text{if } \gamma_{\mathrm{S}}(s(x)) > 1 \\ \min\{\gamma_{\mathrm{RR}}(s(x)), 1\} & \text{otherwise.} \end{cases} \tag{6.15}$$

The convexity of the joint objective, in combination with the convergence of the individual components, guarantees that the fixed-point iteration converges.

6.1.3 Application to rendering

The theory discussed so far can be directly applied to rendering. In the following, we discuss how to do so in the context of forward path tracing. Compared to the simplified setting, there are two major differences: (1) instead of a single integral, there is one per pixel, and (2) RRS occurs multiple times along a path.

Objective

A rendered image consists of multiple integrals, one for each pixel; the goal is to maximize the efficiency across all these integrals. That is, the goal is to obtain local RRS factors that maximize the total efficiency over the entire image. To this end, we extend our prior definition of efficiency to the multi-integral case by using the mean pixel variance and expected pixel render time,

$$\epsilon^{-1} = \left(\frac{1}{N_{\mathrm{px}}} \sum_{\mathrm{px}} \mathbb{V}\left[\langle I_{\mathrm{px}} \rangle\right]\right) \left(\frac{1}{N_{\mathrm{px}}} \sum_{\mathrm{px}} \mathbb{C}\left[\langle I_{\mathrm{px}} \rangle\right]\right). \tag{6.16}$$

Fortunately, this is still a convex objective (being a sum of convex functions), and the derivatives have the exact same form as the single integral case discussed so far.

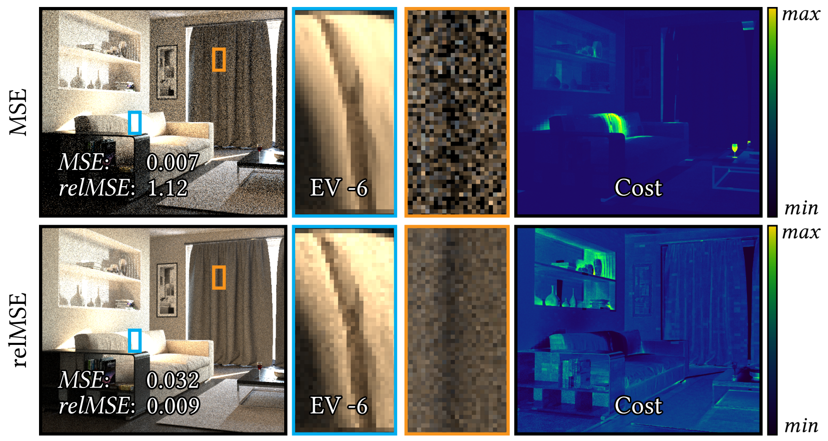


Figure 6.4: *Minimizing the mean-squared error (MSE) or relative MSE (relMSE) in EARS. Using the relMSE performs significantly better in scenes with high contrast. By decreasing the exposure (EV -6) of a crop we can see that using the MSE oversamples very bright regions. Using the relMSE, on the other hand, yields better convergence across the entire dynamic range of the image, as we can see when looking at the average per-pixel cost, shown in false-color. With the MSE, most computation time is invested in the bright pixels. With the relMSE, computation time is spread more evenly.*

There is, however, one more consideration to make. Using the absolute variance will overfit on bright pixels. Instead, we minimize the product of average relative variance and average per-pixel cost:

$$\bar{V}(n)\bar{C}(n) := \left(\frac{1}{N_{\text{px}}} \sum_{\text{px}}^{N} \frac{\mathbb{V}\left[\langle I_{\text{px}}; n\rangle\right]}{I_{\text{px}}^2} \right) \left(\frac{1}{N_{\text{px}}} \sum_{\text{px}}^{N} \mathbb{C}\left[\langle I_{\text{px}}; n\rangle\right] \right). \tag{6.17}$$

Using the relative variance, i.e., dividing by the squared ground truth I_{px}^2, prevents an oversampling of bright regions. In other words, instead of aiming for the lowest mean squared error (MSE), we aim for the lowest *relative* mean squared error (relMSE). Fig. 6.4 compares the difference between the two objectives. The scene has high variance everywhere, but the directly illuminated region on the couch is very bright and completely dominates the MSE. Hence, minimizing the MSE results in RRS factors that focus most of the samples on that region. Minimizing the relMSE instead produces a much more uniform distribution of sampling cost and hence a more perceptually uniform noise.

A problem with this objective is that the ground truth pixel value is unknown in practice. We will later show that a coarse approximation of the ground truth pixel value (e.g., using denoised intermediate renders) is a sufficient surrogate of this value.

Fixed-point function

Following the same steps as the simplified 2D example discussed before, we can formulate the RRS fixed-point functions for the ith fixed-point iteration

$$\gamma(n_i(\bar{\mathbf{x}}_k)) = \underbrace{\frac{T(\bar{\mathbf{x}}_k)}{I_{\mathrm{px}}(\bar{\mathbf{x}}_k)}}_{\text{prefix}} \underbrace{\sqrt{\frac{R(\langle L_{\mathrm{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); n_i \rangle)}{\mathbb{C}\left[\langle L_{\mathrm{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); n_i \rangle\right]}}}_{\text{local}} \underbrace{\sqrt{\frac{\bar{C}(n_i)}{\bar{V}(n_i)}}}_{\text{global}} \tag{6.18}$$

where

$$R(\langle L_{\mathrm{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); n_i \rangle) = \begin{cases} \mathbb{V}\left[\langle L_{\mathrm{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); n_i \rangle\right] & \text{if } n_i(x) > 1 \\ \mathbb{E}\left[\langle L_{\mathrm{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); n_i \rangle^2\right] & \text{else} \end{cases} \tag{6.19}$$

is the piece-wise moment or variance function, computing the *primary*, i.e., single sample, variance or second moment of the reflected radiance estimator at point $\mathbf{x}_k$.

The fixed-point function consists of three components: the relative prefix weight $T(\bar{\mathbf{x}}_k)/I_{\mathrm{px}}$ which is readily available, the local ratio of nested variance and cost, which can be cached in a 5D data structure, and the global variance and cost of the entire image.

The beauty of this approach is that we can perform a fixed-point update of a *continuous* RRS function $n(\bar{\mathbf{x}}_k)$ without actually storing the full continuous representation of all exact values $n(\bar{\mathbf{x}}_k)$ for all possible prefixes $\bar{\mathbf{x}}_k$, which would be prohibitive in practice. Instead, we store only the dependent quantities (i.e., variances and costs) that are used by the next iteration. Each fixed-point iteration stochastically updates some $n(\bar{\mathbf{x}}_k)$ for a set of random $\bar{\mathbf{x}}_k$. Since the probability of each $n(\bar{\mathbf{x}}_k)$ being updated is non-zero (otherwise $\bar{\mathbf{x}}_k$ is never sampled and hence irrelevant), this stochastic fixed-point iteration converges.

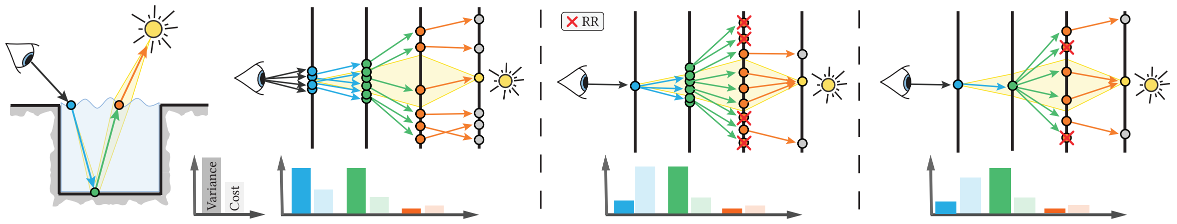
Unlike previous works, which rely on reducing the dimensionality of the problem to be computationally feasible (e.g., Bolin and Meyer [1997] only use the length of a prefix path), our fixed-point scheme does not suffer from the curse of dimensionality and hence enables us to solve the RRS factors without dimensionality reduction.

Convergence with repeated RRS

In rendering practice, RRS is performed at multiple points along a path. Fortunately, this has no effect on the convergence of our fixed-point iteration. An RRS factor $n(\bar{\mathbf{x}}_j)$ occurring at a point before or after $\mathbf{x}_k$ has a similar effect on the derivatives of the fixed-point function as the other RRS factors of unrelated $n(\bar{\mathbf{x}}'_k)$. The criteria used to prove convergence in Appendix B.3 are unaffected and convergence is still guaranteed independent of path length.

Example

Fig. 6.5 illustrates how our fixed-point iteration behaves when applied to forward path tracing rendering a complex caustic (a). The initial training iteration (b) uses classic albedo-based RR and estimates the corresponding variance and cost of the nested $\langle L_{\mathrm{r}} \rangle$ estimators at the positions blue (specular water), green (diffuse floor), and orange (specular water). The cost in the initial pass is given by the lengths of the suffix paths (e.g., blue = 3, green = 2, and orange = 1), as no splitting is done. The variance propagates backwards along the path, and



(a) *Pool scene* **(b)** *Initial data / statistic pass* **(c)** *Fixed-point iteration 1* **(d)** *Fixed-point iteration 2*

Figure 6.5: *Illustration of the fixed-point update behavior for a set of nested estimators (blue, green, and orange) along a caustic path. (a) shows the scene setup, which is similar to the* Pool *scene (Fig. 6.1). The yellow region marks the subset of paths that constitute the caustic. (b), (c), and (d) show the sampling behavior (top) and the estimated variances and costs (bottom) of the nested estimators at different stages: the initial training iteration (b) as well as the first (c), and second (d) fixed-point iteration.*

is dominated by the diffuse surface at green, so blue and green have the same local variance estimates, while orange, being a specular surface directly reflecting the sun, has low variance.

In the first fixed-point iteration (c), the high variance at blue and green results in splitting being done at both. Paths generated at green that do not find the sun (i.e., outside the yellow region) are terminated via RR. After the iteration, the variance estimate at blue decreases, due to the splitting at green, and its cost increases. The variance at green remains the same, while the cost is marginally reduced by the RR done at orange.

In the second fixed-point iteration no splitting is performed at blue, due to the now low variance and high cost. When the path then arrives at green, which has a high variance and low cost, a lot of splitting is done to reduce the overall variance of the pixel estimator. Again, suffixes sampled at green that do not find the sun are terminated by RR at orange.

6.2 Implementation

We implement our method in the Mitsuba renderer [Jakob 2010], using a recursive path tracer as the basis. Our fixed-point iteration requires an iterative rendering process, where each iteration creates some number of samples per pixel and estimates the global and local variances. Initially, we start with classic prefix weight-based RR, and then apply our fixed-point iteration (6.18) to iteratively refine the RRS factors. The required local estimates are stored in a simple 5D data structure. To alleviate the computational overhead of our method, we increase the duration of each iteration over time, thereby reducing the number of updates.

Because our RRS decisions improve over time, the rendered images of early iterations have much higher noise than later ones. Thus, we weight each iteration's rendered image with its inverse variance, which, in theory, yields the optimal combination of images [Hammersley and Handscomb 1968]. However, the variance estimates are based on the same samples as the images themselves, which introduces bias [Kirk and Arvo 1991]. But that bias is typically negligible, and it vanishes with growing iteration times.

The pseudocode in Alg. 3 provides an overview of the process. The individual steps are explained in more detail in the following.

6.2.1 Adapting the theory

In the following, we summarize modifications to the theory that are required for our implementation.

Next event estimation The pixel estimator $\langle I_{\text{px}}; n\rangle$ is given by a recursive path tracer with Russian roulette and splitting, additionally performing next event estimation at each intersection. Following the approach of Vorba and Křivánek [2016], we consider next event estimation and path continuation via BSDF sampling as an atomic operation. That is, a RRS factor of 2 implies that 2 BSDF samples are traced to continue the path, and 2 shadow rays are traced for next event estimation. This integrates nicely into our theory: The computed RRS factors are optimal under the constraint that exactly as many shadow rays need to be traced as BSDF samples. Note that in a more general context, this is not optimal. There can, e.g., be regions of high variance that only benefit from BSDF sampling. Balancing the number of samples between multiple techniques in an MIS combination [Veach and Guibas 1995b] is an orthogonal problem [Sbert et al. 2019].

Handling colors The derivations so far have glanced over the fact that the (reflected) radiance is vector-valued (RGB triplets or spectral samples). The best local RRS factor depends on spectral contributions of the prefix and the local and global estimates. There is, e.g., no point in splitting a 'red' prefix if the local reflected radiance has no red contribution. We can extend our efficiency formula Eq. (6.16) to a multichannel renderer by averaging the variances of individual color channels λ:

$$\epsilon^{-1} = \left(\frac{1}{N_{\text{px}} N_\lambda} \sum_{\text{px}} \sum_{\lambda} \mathbb{V}\left[\langle I_{\text{px}}\rangle_\lambda\right]\right)\left(\frac{1}{N_{\text{px}}} \sum_{\text{px}} \mathbb{E}\left[c\left(\langle I_{\text{px}}\rangle\right)\right]\right). \tag{6.20}$$

Using this as starting point, we find the RRS factors,

$$n = \sqrt{\frac{\sum_\lambda T_\lambda^2(\bar{\mathbf{x}}_k)\, \tilde{I}_{\text{px},\lambda}^{-2}\, R_\lambda(\langle L_{\text{r}}; n_i\rangle)}{\sum_\lambda \bar{V}_\lambda(n_i)}} \sqrt{\frac{\bar{C}(n_i)}{\mathbb{E}\left[c(\langle L_{\text{r}}; n_i\rangle)\right]}}, \tag{6.21}$$

which differ from our monochromatic RRS factors (Eq. (6.18)) only in that the product of local variance estimate with the prefix weight is now performed component-wise, and that we sum up the variances over their color channels λ. Similar derivations could be carried out using other metrics as starting point, e.g., using luminance or the maximum component of the variance spectrum if desired.

Cost heuristic To quantify the cost, we use the same simple heuristic as previous work [Bolin and Meyer 1997; Szirmay-Kalos 2005], i.e., we count the number of rays that are traced by the estimator. For simplicity, we assume that shadow rays, primary rays from the camera, and BSDF samples have the same cost.

Clamping In practice, it is beneficial to limit the allowed range of the RRS factors. On the one hand, the theoretically optimal RRS factor can in principle be arbitrarily large. On the other hand, error in our estimates, due to noise and approximations, can also produce much too large or much too small RRS factors. Thus, we clamp each RRS factor to the interval $(0.05, 20)$ to avoid bias, which can happen if $n(\bar{\mathbf{x}}) = 0$, and to prevent excessive splitting.

Algorithm 3: *Overview of our main rendering loop. The image is rendered in iterations, each taking multiple samples per pixel. Each iteration updates the global and local variance and cost estimates.*

```
1:  function Render
2:    for $i \in 1..n_{\text{iterations}}$ do
3:      $\widetilde{V}^{(i)}, \widetilde{C}^{(i)} = 0$                    ← initialize global statistics to zero
4:      $\widetilde{C}_b^{(i)}, \widetilde{E}_b^{(i)}, \widetilde{M}_b^{(i)}, n_B = 0$                    ← initialize all local statistics
5:      while time budget of iteration not exhausted do
6:        for px in image do                    ← render one sample per pixel
7:          $\bar{\mathbf{x}}_1 = \textsc{SampleCamera}(\text{px})$                    ← start a path from the camera
8:          $c, \langle L_r \rangle = \textsc{LrEstimate}(\bar{\mathbf{x}}_1, \tilde{I}_{\text{px}})$                    ← Alg. 4
9:          $\widetilde{C}^{(i)} \mathrel{+}= 1 + c$                    ← update cost, Eq. (6.22)
10:         $\widetilde{V}^{(i)} \mathrel{+}= \left(T(\bar{\mathbf{x}}_1) \cdot \langle L_r \rangle - \tilde{I}_{\text{px}}\right)^2 / \tilde{I}_{\text{px}}^2$
11:         $\langle I_{\text{px}} \rangle^{(i)} \mathrel{+}= T(\bar{\mathbf{x}}_1) \cdot \langle L_r \rangle$                    ↰ update relative variance (6.23)
12:       $N_{\text{spp}} \mathrel{+}= 1$
13:     $\widetilde{C}^{(i)}, \widetilde{V}^{(i)} \mathrel{/}= N_{\text{px}} \cdot N_{\text{spp}}$                    ← normalize estimates, Eqs. (6.22) and (6.23)
14:     $\langle I \rangle = \textsc{MergeFramesByVariance}(\langle I \rangle, \langle I \rangle^{(i)}, \bar{V}^{(i)})$
15:     $\tilde{I} = \textsc{Denoise}(\langle I \rangle)$
16:     for $B \in$ SpatialCache do                    ← normalize local statistics
17:       $\widetilde{C}_b^{(i)}, \widetilde{E}_b^{(i)}, \widetilde{M}_b^{(i)} \mathrel{/}= n_B$                    ← Eqs. (6.24), (6.25) and (6.27)
18:       $\widetilde{V}_b^{(i)} = \widetilde{M}_b^{(i)} - \left(\widetilde{E}_b^{(i)}\right)^2$                    ← compute variance Eq. (6.26)
19:   return $\langle I \rangle$
```

6.2.2 Global estimates

When a pixel estimate is completed, we record its cost

$$\widetilde{C} = \frac{1}{N_{\text{spp}}} \sum_{s=1}^{N_{\text{spp}}} \frac{1}{N_{\text{px}}} \sum_{\text{px}} c(\langle I_{\text{px}} \rangle_s) \tag{6.22}$$

and approximate the variance using a denoised image [Áfra 2019] in lieu of the ground truth

$$\widetilde{V} = \frac{1}{N_{\text{spp}}} \sum_{s=1}^{N_{\text{spp}}} \frac{1}{N_{\text{px}}} \sum_{\text{px}} \left(\frac{\langle I_{\text{px}}; n \rangle_s - \tilde{I}_{\text{px}}}{\tilde{I}_{\text{px}}} \right)^2 . \tag{6.23}$$

It is important that $\langle I_{\text{px}}; n \rangle$ is the estimator *including splitting*. That is, every path tree generated per sample is considered as a whole.

Outlier removal Even a single outlier in a single pixel can severely distort the estimate of the average image variance. We apply a simple workaround and ignore the 0.001% of all pixels that have the highest variance when computing the average image variance required by our method.

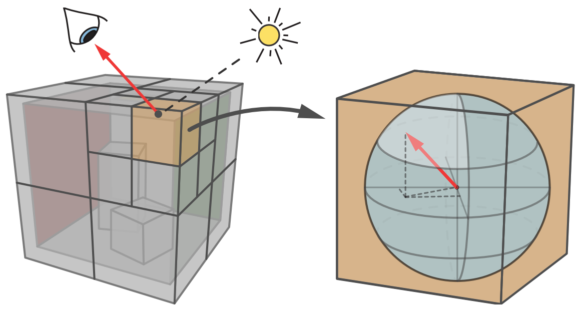


Figure 6.6: *The data structure used by our implementation. The scene is divided by an octree (left). Each cell of which stores variance estimates for the reflected radiance estimator. The directional dependency on ω_o is handled via a simple histogram (right).*

6.2.3 Local estimates

The local variance and cost estimates (see Section 6.1.3) are stored in a 5D data structure, illustrated in Fig. 6.6. The scene is partitioned by an octree, each cell of which contains a histogram over outgoing directions of fixed 4×4 resolution. In our experiments, higher resolutions showed only small improvements in the quality of RRS factors, which were offset by the disadvantage of requiring more training samples. We apply a similar approach to Müller et al. [2017], subdividing leaves of the octree after more than 40,000 samples have been accumulated.

Building the estimates Each directional histogram bin b stores approximations of variance $\widetilde{V}_b$, second moment $\widetilde{M}_b$ and cost $\widetilde{C}_b$ of the reflected radiance estimator $\langle L_\mathrm{r} \rangle$. The second moment is approximated as the average second moment over all points and directions in the bin b,

$$\mathbb{E}\left[\langle L_\mathrm{r}; n_i \rangle^2\right] \approx \widetilde{M}_b = \frac{\sum_{s=1}^{n_b} \langle L_\mathrm{r} \rangle_s^2}{n_b}, \tag{6.24}$$

where n_b is the number of samples within bin B. The variance $\widetilde{V}_b$ is approximated by additionally computing the average reflected radiance in the bin

$$\widetilde{E}_b = \frac{\sum_s^{n_b} \langle L_\mathrm{r} \rangle_s}{n_b}, \tag{6.25}$$

which we can square and subtract from the second moment to approximate the variance

$$\mathbb{V}\left[\langle L_\mathrm{r}; n_i \rangle^2\right] \leq \widetilde{V}_b = \widetilde{M}_b - \widetilde{E}_b^2. \tag{6.26}$$

Note that this approximation replaces the integral of squared L_r terms by the square of the integral. Hence, if the reflected radiance fluctuates strongly within b, the variance is overestimated and the resulting RRS factors will be too large. We discuss this approximation in the next section. The expected cost $\mathbb{E}\left[c(\langle L_\mathrm{r}; n \rangle)\right]$ within a bin is approximated by averaging the cost of all samples from the bin,

$$\mathbb{E}\left[c(\langle L_\mathrm{r}; n \rangle)\right] \approx \widetilde{C}_b = \frac{\sum_s^{n_b} c(\langle L_\mathrm{r} \rangle_s)}{n_b}. \tag{6.27}$$

Algorithm 4: *Pseudocode for the reflected radiance estimation, given a prefix path $\bar{\mathbf{x}}_k$ originating in pixel px. First, we query the corresponding bin in our data structure. Then, we apply our fixed-point function to update the RRS factor. The sample weights and costs are logged in the data structure for each sampled direction from BSDF or next event (the latter was omitted for brevity). For simplicity, we assume monochromatic rendering, but the algorithm can easily be extended to multichannel rendering as discussed in sous-section 6.2.1.*

```
 1: function LrEstimate($\bar{\mathbf{x}}_k, \tilde{I}_{\text{px}}$)
 2:     $b = \text{SpatialCacheBin}(\bar{\mathbf{x}}_k)$                                   ← find responsible bin
 3:     $n = \frac{T(\bar{\mathbf{x}}_k)}{\tilde{I}_{\text{px}}} \sqrt{\frac{\tilde{C}^{(i-1)}}{\tilde{V}^{(i-1)}}} \sqrt{\frac{\tilde{V}_b^{(i-1)}}{\tilde{C}_b^{(i-1)}}}$       ← splitting case (6.18)
 4:     if n < 1 then
 5:         $n = \min\left(1, \frac{T(\bar{\mathbf{x}}_n)}{\tilde{I}_{\text{px}}} \sqrt{\frac{\tilde{C}^{(i-1)}}{\tilde{V}^{(i-1)}}} \sqrt{\frac{\tilde{M}_b^{(i-1)}}{\tilde{C}_b^{(i-1)}}}\right)$   ← compute Russian roulette objective
 6:     $n = \text{clamp}(n, 0.05, 20)$                                   ← clamp to a reasonable range
 7:     $\Sigma c, \Sigma w = 0, 0$                                       ← initialize total cost and contribution
 8:     for $i = 1..\text{StochasticRounding}(n)$ do
 9:         $\bar{\mathbf{x}}_{k+1} = \text{SampleBsdf}(\bar{\mathbf{x}}_k)$                    ← (next event omitted for brevity)
10:         $c, \langle L_\text{r} \rangle = \text{LrEstimate}(\bar{\mathbf{x}}_{k+1}, \tilde{I}_{\text{px}})$
11:         $w = \frac{B(\omega_\text{i}, \mathbf{x}_k, \omega_\text{o})}{p(\omega_\text{i})} \langle L_\text{r} \rangle$                  ← local weight times nested estimate
12:         $\tilde{C}_b^{(i)} \mathrel{+}= c$     $\tilde{E}_b^{(i)} \mathrel{+}= w$                 ← Eqs. (6.25) and (6.27)
13:         $\tilde{M}_b^{(i)} \mathrel{+}= w^2$     $n_b \mathrel{+}= 1$                   ← Eq. (6.24)
14:         $\Sigma c \mathrel{+}= 2 + c$     $\Sigma w \mathrel{+}= w$                ← accumulate cost and contribution
15:     return $\Sigma c, \Sigma w$
```

Incremental learning According to our fixed-point scheme (Section 6.1.3), an iteration i should use the variances and costs derived from the RRS factors of the previous iteration $i-1$. For unbiased estimation, each iteration should hence start computing new variance and cost estimates from scratch. Doing so would require very long iterations for sufficiently converged estimates. To reduce the noise, we include samples from earlier iterations in our estimates. In practice, this slightly reduces the convergence speed of our iterative scheme but yields much more reliable estimates which improve the performance of our method.

6.3 Evaluation

We compare our method and ADRRS [Vorba and Křivánek 2016] for two cases: when applied solely to Russian roulette (RR) and when applied to combined RR and splitting (RRS). As a baseline, we include classic prefix weight-based RR (starting at the 5th bounce), of which we also include an adaptive sampling variant. Unlike ADRRS and our method, which learn their statistics on-line during rendering, the adaptive sampler is provided with a ground truth relative variance image that was computed in a pre-process not contained in the reported render time. All images were renderer for ten minutes with a maximum path length of 40. Our method achieves a speed-up of 1.52× over ADRRS and 5.87× over classic RR (averaged over 19 different scenes), with the poorest performing scene (Glossy Bathroom) being only 9% slower than ADRRS.

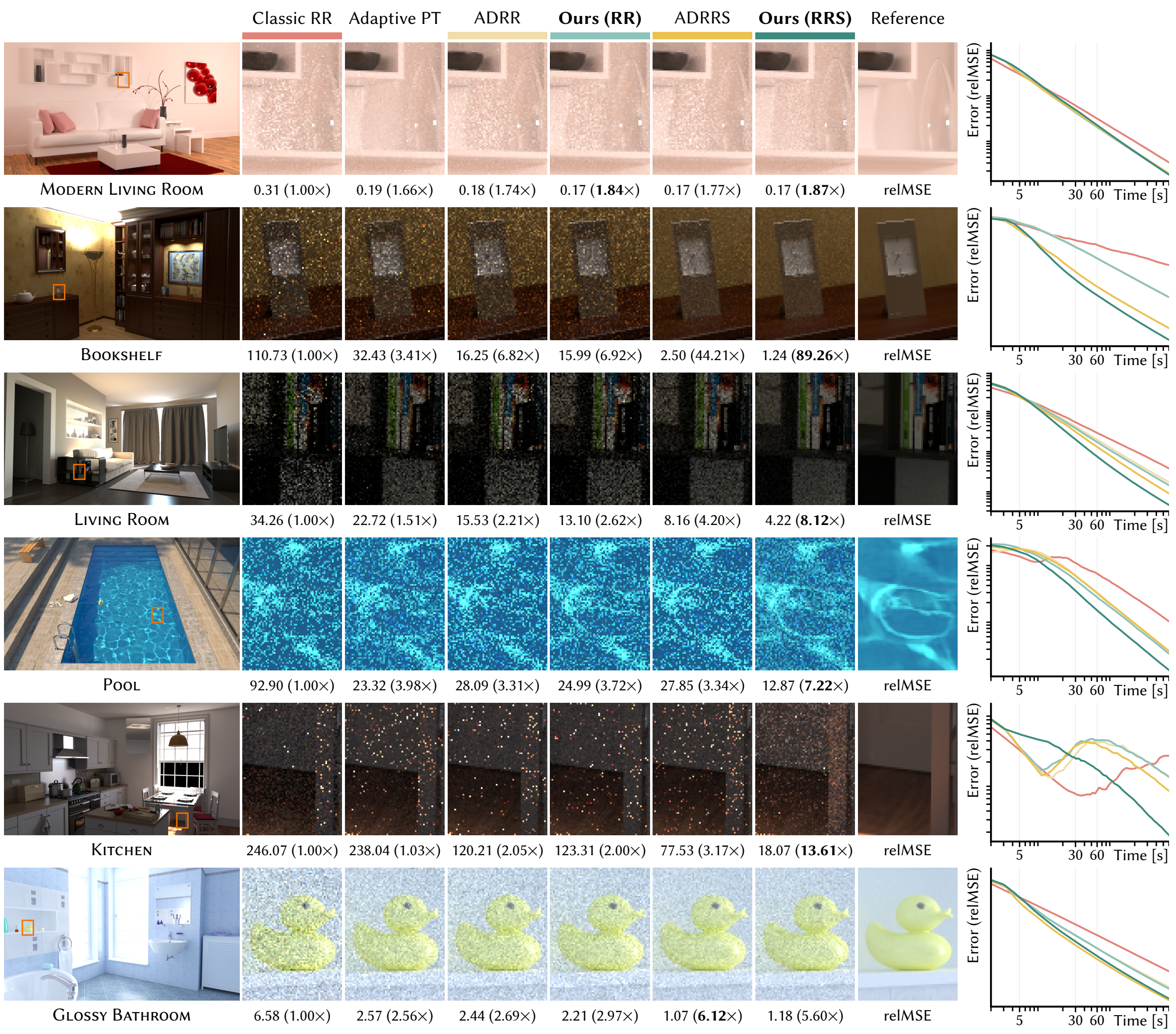

Classic RR
Adaptive PT
ADRR
Ours (RR)
ADRRS
Ours (RRS)
Reference
Modern Living Room
0.31 (1.00×) 0.19 (1.66×) 0.18 (1.74×) 0.17 (1.84×) 0.17 (1.77×) 0.17 (1.87×) relMSE
Bookshelf
110.73 (1.00×) 32.43 (3.41×) 16.25 (6.82×) 15.99 (6.92×) 2.50 (44.21×) 1.24 (89.26×) relMSE
Living Room
34.26 (1.00×) 22.72 (1.51×) 15.53 (2.21×) 13.10 (2.62×) 8.16 (4.20×) 4.22 (8.12×) relMSE
Pool
92.90 (1.00×) 23.32 (3.98×) 28.09 (3.31×) 24.99 (3.72×) 27.85 (3.34×) 12.87 (7.22×) relMSE
Kitchen
246.07 (1.00×) 238.04 (1.03×) 120.21 (2.05×) 123.31 (2.00×) 77.53 (3.17×) 18.07 (13.61×) relMSE
Glossy Bathroom
6.58 (1.00×) 2.57 (2.56×) 2.44 (2.69×) 2.21 (2.97×) 1.07 (6.12×) 1.18 (5.60×) relMSE
Error (relMSE)
5
30
60
Time [s]

Figure 6.7: *We render five scenes with different Russian roulette and splitting strategies for 10 minutes each. The numbers below the crops are the relative mean-squared error (relMSE, lower is better), with the speed-up compared to classic RR in parentheses (higher is better).*

Table 6.1: *Performance statistics of all RR and RRS methods. The samples per pixel is the number of frames rendered at an equal time (i.e., 10 min). The average path length values provide insights into the different termination behaviors of each method. For a better understanding of the splitting behavior of ADRRS and Ours(RRS), additional information is provided: first, the average number of paths generated per primary ray using splitting, and second, the average RRS factors at the first intersection of the primary ray.*

	Samples per pixel					Average path length					Average paths per sample		Average primary splits	
Scene	Classic	ADRR	Ours(RR)	ADRRS	Ours(RRS)	Classic	ADRR	Ours(RR)	ADRRS	Ours(RRS)	ADRRS	Ours(RRS)	ADRRS	Ours(RRS)
M. Living Room	1800	1213	1469	1147	274	5.61	6.81	5.75	6.99	6.90	1.08	7.28	1	5.25
Bookshelf	2370	3099	3196	2699	208	4.59	3.11	3.12	3.45	3.88	1.28	26.50	1	10.62
Living Room	1805	2526	3255	2242	219	4.95	3.19	2.67	3.33	3.13	1.20	19.99	1	13.33
Pool	2814	2234	3042	2208	421	3.17	3.35	2.90	3.37	4.54	1.01	8.69	1	1.82
Kitchen	2400	2258	2438	1967	365	4.78	4.33	4.15	4.59	4.28	1.21	19.98	1	9.42
G. Bathroom	1522	834	1367	698	51	6.92	9.86	6.35	9.62	8.85	1.87	113.55	1	18.45

Fig. 6.7 shows the results for some of our test scenes. The numbers below each crop are the equal-time relative mean-squared error (relMSE), lower is better, for which we discard 0.01% of the pixels with the highest error to increase robustness towards outliers. Note this is distinct from the outlier removal discussed in Section 6.2.2, which discards fewer outliers (0.001%) and uses denoised intermediate renders (in lieu of a reference image) to estimate the relative image variance required by our method. The numbers in parentheses are the speed-up w.r.t. classic RR.

The Modern Living Room is mostly diffuse and illuminated by a large spherical light, making it simple to render with forward path tracing. Due to mostly short paths, RR has little impact here. While our method does not noticeably improve the mean error compared to ADRRS, it still achieves a more uniform noise distribution across the image as shown in Fig. 6.8. The figure shows false-color images of the per-pixel error. Similar to adaptive sampling, our method greatly reduces the error in the caustics of the glass vases, by slightly increasing the error everywhere else. In such simple scenes, the overhead of complex RRS methods like ADRRS and ours does not pay off for very short renders. As can be seen in the error over time plots, in some cases it takes at least 20 seconds to out-perform classic RR.

The Bookshelf is an example of a scene with strong, difficult diffuse indirect illumination. Due to many dark surfaces, classic RR prematurely terminates paths. When applied only to RR, our method achieves a 30% speed-up compared to ADRR, by heeding variance. Splitting can drastically increase the performance here: if an unlikely path finds the directly illuminated regions, splitting within the illuminated region increases the odds of forming a full valid path to the light. This is an important advantage over adaptive sampling, which can only split a pixel as a whole, thus wasting time on paths that do not land in the illuminated region. For full RRS, our method performs twice as fast as ADRRS. The Living Room scene is another example with similar light transport and similar results.

The Pool is an example that shows more clearly the benefit of basing splitting decisions on variance and cost (as done by our method), rather than expected contributions (as done by ADRRS). By directly considering variance, our method performs splitting on the diffuse bottom of the pool, unlike ADRRS (see Fig. 6.1).

The Kitchen combines the effects from Bookshelf and Pool: This scene features strong indirect illumination through a high-variance caustic underneath the table. While both ADRR and our method (RR) similarly increase performance by not killing paths that land in the caustic, even stronger speed-ups can be gained with splitting. By considering variance, our method (RRS) again performs significantly more splitting than ADRRS on the caustic, resulting in less noise in the indirect illumination and a 4.3× speed-up over ADRRS. Similar to the Bookshelf scene, adaptive sampling achieves little improvement in this scene, as noise is mostly caused by indirect illumination from a small brightly lit area.

The Glossy Bathroom is a failure case of our method. The homogeneous illumination makes the scene relatively simple to render, while the many glossy surfaces are problematic for our approach. The over-approximation of the local variance results in excessive splitting and hence in 9% worse performance than ADRRS in this scene. Still, our result outperforms classic RR.

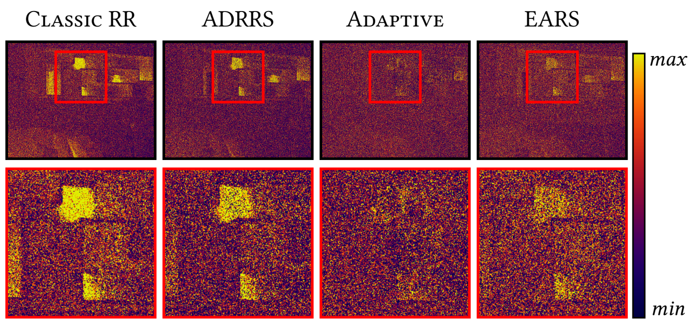


Figure 6.8: *Relative mean-squared error for* Classic *(prefix weight-based) Russian roulette (with and without adaptive sampling), EARS and ADRRS. While both ADRRS and our method approximately achieve the same mean error in this scene, EARS benefits from more uniform noise across the entire image. Classic Russian roulette with adaptive sampling achieves an even more uniform noise distribution, but higher mean error.*

6.3.1 Sampling statistics

To better understand the sampling behavior, additional statistics are shown in Table 6.1: the samples per pixel (SPP), the average path length, and, for the splitting methods, the average number of generated paths per primary ray and the average RRS factor at the first bounce.

Among RR, both ADRR and Ours (RR) perform more aggressive path terminations compared to classic RR, typically resulting in higher sample counts per pixel. Modern Living Room shows an interesting case where classic RR achieves higher SPP, but still performs poorly compared to more sophisticated approaches.

With splitting, the number of SPP naturally decreases. Our optimization objective is to maximize the full efficiency of the rendering process (6.17). Hence, if the variance due to aliasing and sampling the lens is low, as is in most of our scenes, our method tends to perform a lot of splitting at the primary hit. Thereby, variance is reduced as if additional SPP were generated, but without the cost of the primary ray. Dividing the number of paths per sample by the number of primary splits, we see that our method usually does not perform much more splitting than ADRRS at later bounces. A notable exception is the Pool scene, where splitting happens for specular paths that arrive at a caustic.

6.3.2 Overhead

To build and store the required estimates, both ADRRS and EARS incur computational overhead. In our evaluation, we have limited the memory footprint of the data structure to 24 MiB, limiting the spatial partitioning to roughly 22,000 regions. In our experiments, the benefits of higher resolution estimates were outweighed by higher computational costs of updating and traversing the data structure. In terms of computation time, the overhead of ADRRS and EARS results in an average of 15% fewer rays being traced when compared to classic RR. The primary causes of this overhead are the traversal of the tree structure (8.7% of render time) and splatting of the required statistics (5.2%). Denoising the pixel estimate and adapting the data structure each amount to less than 1% of the render time. Note that our method adds no noticeable overhead over ADRRS: Since the solution to our optimization problem is found implicitly through a fixed-point scheme, only a few additional arithmetic operations when splatting samples and computing the RRS factor are carried out.

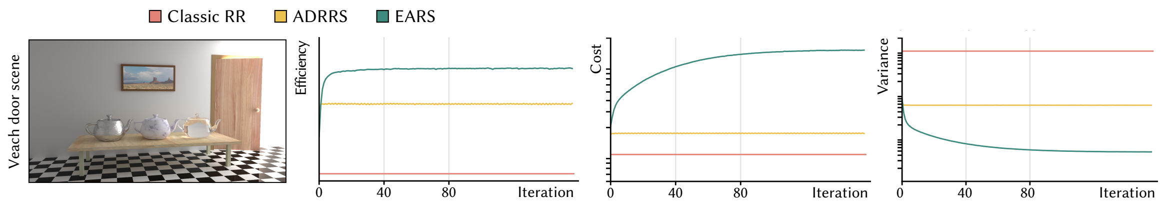


Figure 6.9: *Convergence of different Russian Roulette and splitting methods in the* Veach Door *scene. For each iteration, we measure the average time it takes to render one sample per pixel* (cost) *and the average relative variance with one sample per pixel* (variance)*. The* efficiency *is the inverse product of the two. In all our scenes, our method converges to fixed-point for cost and variance, and hence efficiency.*

6.3.3 Convergence of our fixed-point scheme

While a nice theoretical foundation, the convergence we proved for our fixed-point iteration in theory does not automatically imply convergence in practice. Approximations aside, there is also always noise in the estimates. We conducted an empirical verification of the convergence in a separate rendering setting, without incremental learning and with constant iteration times. Fig. 6.9 plots the variances and sampling costs of the image rendered in each iteration, for the Veach Door scene. Across all our scenes, the cost and variance converged nicely to a fixed-point, when given enough iterations.

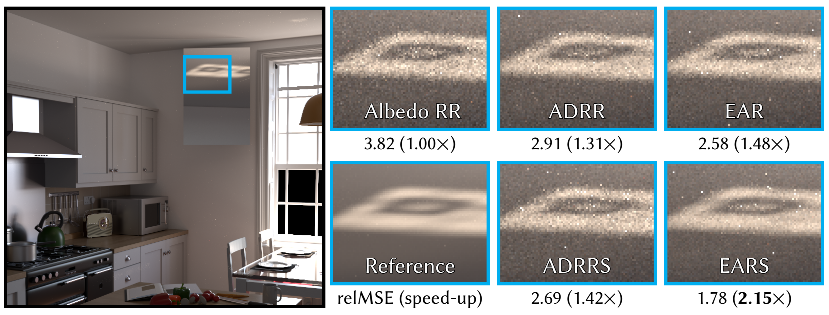


Figure 6.10: *Our method vs. previous work in a path guiding application. The numbers below the crops are the error (relMSE) and, in parentheses, the speed-up compared to the baseline, which is classic albedo-based RR.*

6.3.4 Path guiding

We have also evaluated our method in the context of path guiding, specifically the method of Müller et al. [2017]. There, our method can re-use the same data structures employed by the guiding distribution, only storing a few extra values (the cost and variances) in each spatial cell. We found that using the product of surface albedos, instead of the more common throughput weight, yielded better results for classic RR. The throughput weight can be very low if guiding has a high sampling density for a path across dark surfaces in which case classic RR can undo the benefits of guiding [Vorba and Křivánek 2016]. The product of albedos does not exhibit that problem. The results were similar to the path tracing application. An example is shown in Fig. 6.10. In the Kitchen scene, our method outperforms ADRR(S) both with and without splitting, achieving a speed-up of 1.5× compared to ADRRS.

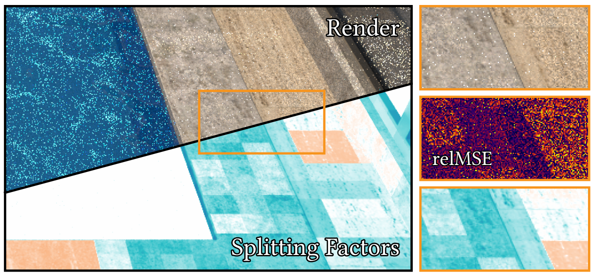


Figure 6.11: *Artefacts due to discretized estimates at a low sample count. The spatial caches used in the estimation of our RRS factors can create small visual artefacts in case the caches have not received enough samples to compute accurate variance estimates.*

6.4 Limitations and future work

In practice, our method is limited by the accuracy of the required estimates. Promising directions for future work include combinations with image space adaptive sampling and bidirectional algorithms.

Estimation error Noise in the local estimates can cause poor RRS decisions. The worst case occurs if some spatial caches contain much lower-quality estimates than their immediate neighbors. An example is shown in Fig. 6.11 for a shorter rendering of the Pool scene. The severe outliers yield poor variance estimates that differ strongly between caches. As a consequence, islands of noise with hard edges in-between are visible in the image. The problem vanishes with longer renderings, due to the increasing iteration times and the fact that we keep the variance estimates of the previous iterations. In practice, this can be alleviated by spatial filtering, i.e., by sharing estimates between caches. Similar problems are encountered in path guiding methods [Vorba et al. 2014; Müller et al. 2017].

Combining with Adaptive Sampling Our method performs Russian roulette at the primary hit point if some parts of the image have a significantly higher variance than others. An example is shown in Fig. 6.12. The figure shows a crop of the Pool scene and the corresponding RRS factor at the primary hit used by our method. The noise in this directly illuminated region is greatly increased by our method, due to aggressive RR at the primary hit point. While this increases the overall efficiency by focusing computation time on the difficult pixels, i.e., the caustics, it also wastes primary rays that could have been avoided altogether. Future work could combine our method with adaptive sampling in image space [Zwicker et al. 2015] to not sample such wasted primary rays in the first place. Doing so requires a small modification to our optimization objective: We should maximize the efficiency of each individual pixel rather than the average across the whole image, to benefit the most from adaptive sampling.

Bidirectional methods We have shown that our method can successfully increase efficiency in a unidirectional path tracer. A promising area of future work would be to apply our method to bidirectional algorithms [Veach and Guibas 1995a; Lafortune and Willems 1993; Georgiev et al. 2012a; Hachisuka et al. 2012b]. This is significantly more complicated, because bidirectional algorithms are multi-sample MIS combinations of different sampling techniques that each construct full paths between the camera and the light sources. RRS along these paths

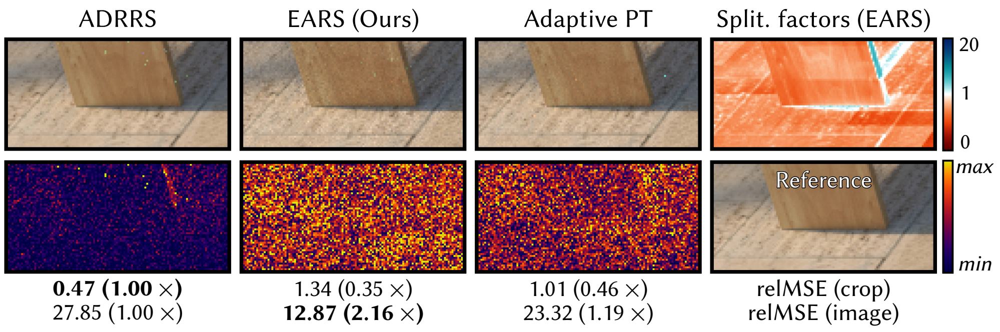


Figure 6.12: *A zoom-in of the* Pool *scene (see Figs. 6.1 and 6.7), comparing ADRRS and our method. The top-right false-color image shows the RRS factors at the primary hit used by our method. Note that aggressive RR is performed, i.e., many paths started from the camera never sample any contribution. This manifests in much higher noise in some regions of the image. The reason for this behavior is that our method focuses computation time on the much more challenging caustic. Adaptive sampling similarly suffers from increased noise, but less severely as it does not need to resort to Russian roulette to artificially lower sample counts.*

would become part of the MIS weights, hence the splitting decisions on a camera sub-path affect the splitting decisions on the light sub-paths. This produces a non-convex objective that is much harder to optimize. Further, RRS in a bidirectional context is problematic for MIS weighting, as the classic balance heuristic ignores the covariance introduced by splitting [Popov et al. 2015; Grittmann et al. 2021].

Participating media We have limited our discussion and implementation to light transport on surfaces. Volumetric transport can equally benefit from RRS [Herholz et al. 2019]. Our theory can be easily extended to the volumetric case, only the practical implementation, in particular the data structure, has to be extended.

Correlated sampling For splitting, we have assumed that variance decreases with $O(n^{-1})$. This is true for uncorrelated sampling, but not for quasi-Monte Carlo (QMC) sampling. Most likely, not all samplers will yield a necessary convex objective. However, we expect the effect of this assumption to be less significant than other sources of inaccuracies (in particular discretization and noise).

Dynamic scenes In dynamic scenes, some learned statistics could potentially be reused in later frames. Future work could look into how many iterations of our fixed-point scheme are necessary after a scene update and which parts of the data structure might need to be discarded and retrained from scratch.

7 Conclusion

"Life finds a way" – and so do learning methods in Monte Carlo rendering! The ways light interacts with our world are numerous and fascinating. Even from seemingly simple equations like the rendering equation, complex behaviors can emerge. While specialized techniques exist for certain types of light effects, handling unforeseen ones robustly requires adaptability. Thus, learning methods are indispensable tools for achieving robust and holistic rendering.

In this work, we explored what constitutes effective learning algorithms for light transport – from data representation and the targets to be learned, to the fitting process itself. By strategically optimizing these components for desirable goals, such as overall render efficiency, we have demonstrated significant improvements over approaches that rely on heuristics.

Variance-Aware Path Guiding

Previous guiding approaches pursue the dream of zero-variance sampling: If only we could make every local sampling decision perfect, the whole estimator would have zero variance. In reality, numerous limitations currently prevent this dream from becoming a reality. Some decisions cannot be made perfect, for example, because guiding them requires too long training times. We present a general approach to deal with these constraints and design target densities for path guiding that are optimal if zero-variance sampling is not feasible. The trivial modifications necessary to compute our target densities yield significant gains in efficiency and robustness.

In our primary application, shaping directional distributions in path guiding, we achieve more than 50% average speed-up over a vast corpus of scenes. Especially glossy effects are handled much better by our approach, as they are optimally marginalized over. Applying our theory to one-sample MIS optimization, we achieve similar speed-ups and greatly reduce artefacts present in previous approaches. Finally, we demonstrate that our method can also yield speed-ups in learning light selection for the many lights problem. In all applications, we require only minimal changes to the code base, as representation and scheme to learn it are unaltered.

Focal Path Guiding

Despite its pervasive use throughout computer graphics in the form of the pinhole camera, rendering a camera obscura remains almost impossible without manual intervention. We identify and analyze a family of light effects that all share the same difficulty – light converging in small regions in space, either to pass narrow gaps or by being converged through lenses – and coin them "focal effects". With this work, we focus on the representation used by path guiding and show that optimizing it to model these effects allows us to handle focal effects robustly. Our technique unifies all types of focal effects in a single framework and can render effects that previous state-of-the-art techniques are unable to handle.

Efficiency-Aware RRS

Russian roulette and splitting are common techniques that can be found in nearly every renderer. However, existing approaches are either based on zero-variance assumptions or suffer from over-simplifications, which result in suboptimal performance in practice. We derive optimal decisions and demonstrate how they can be learned using a simple yet effective fixed-point scheme. Assuming perfect knowledge of variances and expected costs, the fixed-point iteration is proven to converge to the optimal RRS factors that maximize the rendering efficiency. In our rendering application, we iteratively improve the RRS factors used by a forward path tracer; our implementation employs a simple 5D data structure to track variances and costs throughout the scene. Despite the simplicity, we achieve consistent speed-ups over the state of the art of $1.6\times$ on average over a vast corpus of scenes. Especially scenes with challenging indirect illumination benefit from the fact that our method, unlike previous work, directly minimizes variance and cost.

Closing words

While this thesis ends here, our work is far from over. Numerous other decisions in light transport simulation are still based on heuristics that are likely to benefit from learning. We hope the insights offered by this thesis inspire others to look at learning from a new perspective, with an emphasis on goal-driven optimization of the individual components of learning algorithms.

Bibliography

Attila T. Áfra. 2019. Intel® Open Image Denoise. https://www.openimagedenoise.org/

Pontus Andersson, Jim Nilsson, Peter Shirley, and Tomas Akenine-Möller. 2021. Visualizing the Error in Rendered High Dynamic Range Images. In *Eurographics Short Papers*.

James Arvo. 1986. Backward ray tracing. In *Developments in Ray Tracing, Computer Graphics, Proc. of ACM SIGGRAPH 86 Course Notes*. 259–263.

James Arvo and David Kirk. 1990. Particle transport and image synthesis. *SIGGRAPH '90*, 63–66.

Steve Bako, Mark Meyer, Tony DeRose, and Pradeep Sen. 2019. Offline Deep Importance Sampling for Monte Carlo Path Tracing. *Computer Graphics Forum (Proceedings of Pacific Graphics 2019)* 38, 7 (10 2019), 527–542.

Stefan Banach. 1922. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math* 3, 1 (1922), 133–181.

Thomas Bashford-Rogers, Kurt Debattista, and Alan Chalmers. 2012. A significance cache for accelerating global illumination. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1837–1851.

Richard Bellman. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

Vasile Berinde. 2007. *Iterative approximation of fixed points*. Vol. 1912. Springer.

Benedikt Bitterli, Jan Novák, and Wojciech Jarosz. 2015. Portal-masked environment map sampling. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 13–19.

Mark R. Bolin and Gary W. Meyer. 1997. An error metric for Monte Carlo ray tracing. In *Eurographics Workshop on Rendering Techniques*. Springer, 57–68.

Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Daniel Teece. 2018. The design and evolution of Disney's Hyperion renderer. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 1–22.

Norbert Bus and Tamy Boubekeur. 2017. Double Hierarchies for Directional Importance Sampling in Monte Carlo Rendering. *Journal of Computer Graphics Techniques (JCGT)* 6, 3 (28 August 2017), 25–37.

David Cline, Parris K. Egbert, Justin F. Talbot, and David L. Cardon. 2006. Two Stage Importance Sampling for Direct Lighting. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (Nicosia, Cyprus) *(EGSR '06)*. Eurographics Association, Goslar, DEU, 103–113.

Hong Deng, Beibei Wang, Rui Wang, and Nicolas Holzschuch. 2020. A practical path guiding method for participating media. *Computational Visual Media* 6 (2020), 37–51.

Stavros Diolatzis, Adrien Gruson, Wenzel Jakob, Derek Nowrouzezahrai, and George Drettakis. 2020. Practical product path guiding using linearly transformed cosines. In

Computer Graphics Forum, Vol. 39. Wiley Online Library, 23–33.

Ana Dodik, Marios Papas, Cengiz Öztireli, and Thomas Müller. 2022. Path Guiding Using Spatio-Directional Mixture Models. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 172–189.

Luca Fascione, Johannes Hanika, Mark Leone, Marc Droske, Jorge Schwarzhaupt, Tomáš Davidovič, Andrea Weidlich, and Johannes Meng. 2018a. Manuka: A batch-shading architecture for spectral path tracing in movie production. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 1–18.

Luca Fascione, Johannes Hanika, Rob Pieké, Ryusuke Villemin, Christophe Hery, Manuel Gamito, Luke Emrose, and André Mazzone. 2018b. Path tracing in production. In *ACM SIGGRAPH 2018 Courses*. 1–79.

Hans Fischer. 2011. *A history of the central limit theorem: from classical to modern probability theory*. Vol. 4. Springer.

Iliyan Georgiev, Thiago Ize, Mike Farnsworth, Ramón Montoya-Vozmediano, Alan King, Brecht Van Lommel, Angel Jimenez, Oscar Anson, Shinji Ogaki, Eric Johnston, et al. 2018. Arnold: A brute-force production path tracer. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 1–12.

Iliyan Georgiev, Jaroslav Krivánek, Tomas Davidovic, and Philipp Slusallek. 2012a. Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics* 31, 6 (2012), 192–1.

Iliyan Georgiev, Jaroslav Krivanek, Toshiya Hachisuka, Derek Nowrouzezahrai, and Wojciech Jarosz. 2013. Joint importance sampling of low-order volumetric scattering. *ACM Transactions on Graphics* 32, 6 (2013), 164–1.

Iliyan Georgiev, Jaroslav Křivánek, Stefan Popov, and Philipp Slusallek. 2012b. Importance caching for complex illumination. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 701–710.

Pascal Grittmann, Iliyan Georgiev, and Philipp Slusallek. 2021. Correlation-Aware Multiple Importance Sampling for Bidirectional Rendering Algorithms. *Computer Graphics Forum (EG 2021)* 40, 2 (2021), 231–238.

Pascal Grittmann, Arsène Pérard-Gayot, Philipp Slusallek, and Jaroslav Křivánek. 2018. Efficient caustic rendering with lightweight photon mapping. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 133–142.

Adrien Gruson, Mickaël Ribardière, Martin Šik, Jiří Vorba, Rémi Cozot, Kadi Bouatouch, and Jaroslav Křivánek. 2017. A spatial target function for metropolis photon tracing. *ACM Transactions on Graphics (TOG)* 36, 1 (2017), 4.

Jerry Jinfeng Guo, Pablo Bauszat, Jacco Bikker, and Elmar Eisemann. 2018. Primary sample space path guiding. In *Eurographics Symposium on Rendering*, Vol. 2018. The Eurographics Association, 73–82.

Jerry Jinfeng Guo, Martin Eisemann, and Elmar Eisemann. 2020. Next Event Estimation++: Visibility Mapping for Efficient Light Transport Simulation. *Computer Graphics Forum* 39, 7 (2020), 205–217.

Toshiya Hachisuka, Wojciech Jarosz, Guillaume Bouchard, Per Christensen, Jeppe Revall Frisvad, Wenzel Jakob, Henrik Wann Jensen, Michael Kaschalk, Claude Knaus, Andrew Selle, et al. 2012a. State of the art in photon density estimation. *ACM SIGGRAPH 2012 Courses* (2012), 1–469.

Toshiya Hachisuka and Henrik Wann Jensen. 2011. Robust adaptive photon tracing using photon path visibility. *ACM Transactions on Graphics (TOG)* 30, 5 (2011), 114–1.

Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. 2012b. A path space extension for robust light transport simulation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–10.

J.M. Hammersley and D.C. Handscomb. 1968. *Monte Carlo Methods*. Springer, Dordrecht.

Johannes Hanika, Marc Droske, and Luca Fascione. 2015. Manifold Next Event Estimation. *Computer Graphics Forum* 34, 4 (jul 2015), 87–97.

Johannes Hanika, Andrea Weidlich, and Marc Droske. 2022. Once-more scattered next event estimation for volume rendering. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 17–28.

Vlastimil Havran and Mateu Sbert. 2014. Optimal Combination of Techniques in Multiple Importance Sampling. In *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (Shenzhen, China). ACM, New York, NY, 141–150.

Paul S Heckbert. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. 145–154.

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik Lensch. 2018. A Unified Manifold Framework for Efficient BRDF Sampling based on Parametric Mixture Models. In *EGSR '18 EI&I (EGSR '18)*. Eurographics Association, 41–52.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik P. A. Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* 35 (2016), 67–77.

Sebastian Herholz, Yangyang Zhao, Oskar Elek, Derek Nowrouzezahrai, Hendrik PA Lensch, and Jaroslav Křivánek. 2019. Volume path guiding based on zero-variance random walk theory. *ACM Transactions on Graphics (TOG)* 38, 3 (2019), 1–19.

Heinrich Hey and Werner Purgathofer. 2002. Importance Sampling with Hemispherical Particle Footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics* (Budmerice, Slovakia) *(SCCG '02)*. ACM, New York, NY, USA, 107–114.

Yuchi Huo and Sung-eui Yoon. 2021. A survey on deep learning-based Monte Carlo denoising. *Computational visual media* 7 (2021), 169–185.

Wenzel Jakob. 2010. *Mitsuba renderer*. https://www.mitsuba-renderer.org

Wenzel Jakob and Steve Marschner. 2012. Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–13.

Henrik Wann Jensen. 1995. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques '95*, Patrick M. Hanrahan and Werner Purgathofer (Eds.). Springer Vienna, Vienna, 326–335.

Henrik Wann Jensen. 1996. Global Illumination using Photon Maps. In *Eurographics workshop on Rendering techniques*, Xavier Pueyo and Peter Schröder (Eds.). Springer, Springer Vienna, Vienna, 21–30.

James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 143–150.

Ondřej Karlík, Martin Šik, Petr Vévoda, Tomáš Skřivan, and Jaroslav Křivánek. 2019. MIS Compensation: Optimizing Sampling Techniques in Multiple Importance Sampling. *ACM*

Transactions on Graphics (SIGGRAPH Asia '19) 38, 6 (2019), 1–12.

Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. 2002. A simple and robust mutation strategy for the metropolis light transport algorithm. In *Computer Graphics Forum*, Vol. 21. Wiley Online Library, 531–540.

Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, Per H Christensen, Johannes Hanika, Christian Eisenacher, and Gregory Nichols. 2015. The path tracing revolution in the movie industry.. In *SIGGRAPH Courses*. 24–1.

David Kirk and James Arvo. 1991. Unbiased Sampling Techniques for Image Synthesis. *ACM Transactions on Graphics (SIGGRAPH '91)* 25, 4 (jul 1991), 153–156.

Ivo Kondapaneni, Petr Vévoda, Pascal Grittmann, Tomáš Skřivan, Philipp Slusallek, and Jaroslav Křivánek. 2019. Optimal multiple importance sampling. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics* 22, 1 (March 1951), 79–86.

Eric P. Lafortune and Yves D. Willems. 1993. Bi-directional Path Tracing. *Compugraphics '93*, 145–153.

Eric P. Lafortune and Yves D. Willems. 1995. A 5D tree to reduce the variance of Monto Carlo ray tracing. In *Sixth Eurographics Workshop on Rendering, Sixth Eurographics Workshop on Rendering*. 11–20.

He Li, Beibei Wang, Changhe Tu, Kun Xu, Nicolas Holzschuch, and Ling-Qi Yan. 2022. Unbiased caustics rendering guided by representative specular paths. In *SIGGRAPH Asia 2022 Conference Papers*. 1–8.

Heqi Lu, Romain Pacanowski, and Xavier Granier. 2013. Second-Order Approximation for Variance Reduction in Multiple Importance Sampling. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 131–136.

Michael D McCool and Peter K Harwood. 1997. Probability trees. In *Graphics Interface*, Vol. 97. 37–46.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics* 21, 6 (1953), 1087–1092.

Thomas Müller. 2016. Real-Time Transient Rendering. https://tom94.net/pages/projects/femto

Thomas Müller, Markus H. Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum* 36 (2017), 91–100.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural importance sampling. *ACM Transactions on Graphics (TOG)* 38, 5 (2019), 1–19.

David Murray, Sofiane Benzait, Romain Pacanowski, and Xavier Granier. 2020. On Learning the Best Balancing Strategy. In *Eurographics 2020*, Vol. 20. 1–4.

Johannes Nilsson. 2023. *Hierarchical Reconstruction of Quadtree-Based Approximations of Incident Radiance*. Master's thesis.

Shinji Ogaki. 2020. Generalized Light Portals. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2 (2020), 1–19.

Hisanari Otsu, Johannes Hanika, Toshiya Hachisuka, and Carsten Dachsbacher. 2018. Geometry-aware metropolis light transport. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–11.

Art Owen and Yi Zhou. 2000. Safe and Effective Importance Sampling. *J. Amer. Statist. Assoc.* 95, 449 (2000), 135–143.

Jacopo Pantaleoni. 2019. Importance Sampling of Many Lights with Reinforcement Lightcuts Learning. *arXiv preprint arXiv:1911.10217* (2019).

Jacopo Pantaleoni and Eric Heitz. 2017. Notes on optimal approximations for importance sampling. *arXiv preprint arXiv:1707.08358* (2017).

Vincent Pegoraro, Carson Brownlee, Peter S Shirley, and Steven G Parker. 2008. Towards interactive global illumination effects via sequential Monte Carlo adaptation. In *2008 IEEE Symposium on Interactive Ray Tracing*. IEEE, 107–114.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

Stefan Popov, Ravi Ramamoorthi, Fredo Durand, and George Drettakis. 2015. Probabilistic Connections for Bidirectional Path Tracing. *Computer Graphics Forum* 34, 4 (jul 2015), 75–86.

Alexander Rath. 2019. *Path Guiding with Marginalized Importance Sampling*. Master's thesis. Saarland University, Saarbrücken, Germany.

Alexander Rath, Pascal Grittmann, Sebastian Herholz, Petr Vévoda, Philipp Slusallek, and Jaroslav Křivánek. 2020. Variance-Aware Path Guiding. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 151–1.

Alexander Rath, Pascal Grittmann, Sebastian Herholz, Philippe Weier, and Philipp Slusallek. 2022a. EARS: Efficiency-Aware Russian Roulette and Splitting. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

Alexander Rath, Pascal Grittmann, Sebastian Herholz, Philippe Weier, and Philipp Slusallek. 2022b. *Implementation of EARS: Efficiency-Aware Russian Roulette and Splitting*.

Alexander Rath, Ömercan Yazici, and Philipp Slusallek. 2023. Focal Path Guiding for Light Transport Simulation. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–10.

Florian Reibold, Johannes Hanika, Alisa Jung, and Carsten Dachsbacher. 2018. Selective guided sampling with complete light transport paths. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–14.

Alfréd Rényi. 1961. On measures of entropy and information. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability, volume 1: contributions to the theory of statistics*, Vol. 4. University of California Press, 547–562.

Jorge Revelles, Carlos Urena, and Miguel Lastra. 2000. An efficient parametric algorithm for octree traversal. (2000).

R. Tyrrell Rockafellar. 1993. Lagrange Multipliers and Optimality. *SIAM Rev.* 35, 2 (1993), 183–238.

Lukas Ruppert, Sebastian Herholz, and Hendrik PA Lensch. 2020. Robust fitting of parallax-aware mixtures for path guiding. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 147–1.

Mateu Sbert, Vlastimil Havran, and Laszlo. Szirmay-Kalos. 2016. Variance Analysis of Multi-sample and One-sample Multiple Importance Sampling. *Computer Graphics Forum* 35, 7 (2016), 451–460.

Mateu Sbert, Vlastimil Havran, and Laszlo Szirmay-Kalos. 2018. Multiple importance sampling revisited: breaking the bounds. *EURASIP Journal on Advances in Signal Processing* 2018, 1 (2018), 1–15.

Mateu Sbert, Vlastimil Havran, and László Szirmay-Kalos. 2019. Optimal Deterministic Mixture Sampling.. In *Eurographics (Short Papers)*. 73–76.

Vincent Schüßler, Johannes Hanika, Alisa Jung, and Carsten Dachsbacher. 2022. Path Guiding with Vertex Triplet Distributions. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 41, 4 (2022), 1–15.

Peter Shirley, Bretton Wade, Philip M Hubbard, David Zareski, Bruce Walter, and Donald P Greenberg. 1995. Global illumination via density-estimation. In *Eurographics Workshop on Rendering Techniques*. Springer, 219–230.

Martin Šik and Jaroslav Křivánek. 2018. Survey of Markov chain Monte Carlo methods in light transport simulation. *IEEE transactions on visualization and computer graphics* 26, 4 (2018), 1821–1840.

Martin Šik, Hisanari Otsu, Toshiya Hachisuka, and Jaroslav Křivánek. 2016. Robust light transport simulation via metropolised bidirectional estimators. *ACM Trans. Graph* 35, 6 (2016), 245.

László Szésci, László Szirmay-Kalos, and Csaba Kelemen. 2003. Variance reduction for Russian-roulette. (2003).

László Szirmay-Kalos. 2005. Go with the winners strategy in path tracing. *Journal of WSCG, 2005, vol. 13, num. 1-3* (2005), 49–56.

Justin F Talbot. 2005. *Importance resampling for global illumination*. Brigham Young University.

Shelby E Temple, Juliette E McGregor, Camilla Miles, Laura Graham, Josie Miller, Jordan Buck, Nicholas E Scott-Samuel, and Nicholas W Roberts. 2015. Perceiving polarization with the naked eye: characterization of human polarization sensitivity. *Proceedings of the Royal Society B: Biological Sciences* 282, 1811 (2015).

Eric Veach. 1998. *Robust Monte Carlo methods for light transport simulation*. Stanford University.

Eric Veach and Leonidas Guibas. 1995a. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 145–167.

Eric Veach and Leonidas J Guibas. 1995b. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *SIGGRAPH '95*. ACM, 419–428.

Eric Veach and Leonidas J Guibas. 1997. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 65–76.

Andreas Velten, Di Wu, Adrian Jarabo, Belen Masia, Christopher Barsi, Chinmaya Joshi, Everett Lawson, Moungi Bawendi, Diego Gutierrez, and Ramesh Raskar. 2013. Femto-photography: capturing and visualizing the propagation of light. *ACM Transactions on Graphics (ToG)* 32, 4 (2013), 1–8.

Petr Vévoda, Ivo Kondapaneni, and Jaroslav Křivánek. 2018. Bayesian online regression for adaptive direct illumination sampling. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. 2019. Path guiding in production. In *ACM SIGGRAPH 2019 Courses*. 1–77.

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33, 4 (2014), 1–11.

Jiří Vorba and Jaroslav Křivánek. 2016. Adjoint-driven Russian roulette and splitting in light transport simulation. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and

Donald P Greenberg. 2005. Lightcuts: a scalable approach to illumination. In *ACM SIGGRAPH 2005 Papers*. 1098–1107.

Bruce Walter, Philip M Hubbard, Peter Shirley, and Donald P Greenberg. 1997. Global illumination using local linear density estimation. *ACM Transactions on Graphics (TOG)* 16, 3 (1997), 217–259.

Cem Yuksel. 2021. Stochastic Lightcuts for Sampling Many Lights. *IEEE Transactions on Visualization and Computer Graphics* 27, 10 (2021), 4049–4059.

Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. 2020. Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 149–1.

Quan Zheng and Matthias Zwicker. 2019. Learning to importance sample in primary sample space. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 169–179.

Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. 2021. Photon-driven neural reconstruction for path guiding. *ACM Transactions on Graphics (TOG)* 41, 1 (2021), 1–15.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 667–681.

Appendices

A

Variance-Aware Path Guiding

A.1 Target density for irradiance

Given the irradiance estimator:

$$\langle E(x)\rangle = \frac{\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle \, |\cos\theta_\mathrm{i}|}{p(\omega_\mathrm{i} \mid x)}, \tag{A.1}$$

our goal is to minimize its variance:

$$\mathbb{V}\left[\langle E(x)\rangle\right] = \mathbb{E}\left[\langle E(x)\rangle^2\right] - E^2(x). \tag{A.2}$$

The free variable is the PDF $p(\omega_\mathrm{i} \mid x)$: For path guiding, we would like to find the best such PDF and approximate it based on training samples. Looking at (A.2), we can see that only the first term, the second moment $\mathbb{E}\left[\langle E(x)\rangle^2\right]$, depends on the PDF. The squared ground truth value $E^2(x)$ is constant. The second moment is a convex functional of $p(\omega_\mathrm{i} \mid x)$:

$$\mathbb{E}\left[\langle E(x)\rangle^2\right] = \int_\Omega \frac{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right] |\cos\theta_\mathrm{i}|^2}{p(\omega_\mathrm{i} \mid x)} \,\mathrm{d}\omega_\mathrm{i}. \tag{A.3}$$

Hence, the minimizing PDF can be found via Lagrange multipliers:

$$p_E(\omega_\mathrm{i} \mid x) = \underset{p(\omega_\mathrm{i}|x)}{\arg\min}\, \mathbb{E}\left[\langle E(x)\rangle^2\right] + \lambda\left(\int p(\omega_\mathrm{i}' \mid x)\,\mathrm{d}\omega_\mathrm{i}' - 1\right), \tag{A.4}$$

where λ is the Lagrange multiplier and ensures that $p(\omega_\mathrm{i} \mid x)$ integrates to one, i.e., is a valid PDF. The minimizing PDF p_E – our target density – can be found by setting the derivatives to zero. First, the derivative with respect to the PDF:

$$\begin{aligned} 0 &= \frac{\partial}{\partial p(\omega_\mathrm{i} \mid x)} \mathbb{E}\left[\langle E(x)\rangle^2\right] + \lambda && \text{(A.5)}\\ &= \frac{\partial}{\partial p(\omega_\mathrm{i} \mid x)} \int_\Omega \frac{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}', x)\rangle^2\right] |\cos\theta_\mathrm{i}'|^2}{p(\omega_\mathrm{i}' \mid x)} \,\mathrm{d}\omega_\mathrm{i}' + \lambda && \text{(A.6)}\\ &= -\frac{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right] |\cos\theta_\mathrm{i}|^2}{p^2(\omega_\mathrm{i} \mid x)} + \lambda. && \text{(A.7)} \end{aligned}$$

Solving for $p(\omega_\mathrm{i} \mid x)$:

$$p(\omega_\mathrm{i} \mid x) = \frac{1}{\sqrt{\lambda}} \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta_\mathrm{i}|. \tag{A.8}$$

Now, taking the derivative with respect to λ and setting it to zero:

$$\int p(\omega'_\mathrm{i} \mid x)\,\mathrm{d}\omega'_\mathrm{i} - 1 = 0, \tag{A.9}$$

and substituting (A.8):

$$\int \frac{1}{\sqrt{\lambda}} \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega'_\mathrm{i}, x)\rangle^2\right]} |\cos\theta'_\mathrm{i}|\,\mathrm{d}\omega'_\mathrm{i} = 1, \tag{A.10}$$

we obtain the following equation for $\sqrt{\lambda}$:

$$\sqrt{\lambda} = \int \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega'_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta'_\mathrm{i}|\,\mathrm{d}\omega'_\mathrm{i}. \tag{A.11}$$

Substituting this back into (A.8):

$$p(\omega_\mathrm{i} \mid x) = \frac{\sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta_\mathrm{i}|}{\int \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega'_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta'_\mathrm{i}|\,\mathrm{d}\omega'_\mathrm{i}}, \tag{A.12}$$

we see that $\sqrt{\lambda}$ acts as a normalizing constant. The unnormalized target function is:

$$p_E(\omega_\mathrm{i} \mid x) \propto \sqrt{\mathbb{E}\left[\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle^2\right]}\, |\cos\theta_\mathrm{i}|. \tag{A.13}$$

That is, sampling should be proportional to the square root of the second moment of the nested radiance estimator, multiplied by the cosine term.

A.2 Target density for marginalized product sampling

The goal is to guide an estimator for the reflected radiance, with a PDF independent of the outgoing direction ω_o:

$$\langle L_\mathrm{o}(x, \omega_\mathrm{o})\rangle = \frac{B(\omega_\mathrm{i}, x, \omega_\mathrm{o})\,\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\rangle\, |\cos\theta_\mathrm{i}|}{p(\omega_\mathrm{i} \mid x)}. \tag{A.14}$$

To find a suitable target distribution, we first need to define our optimization goal. One option is to minimize the expected error under a given distribution of outgoing directions ω_o:

$$p_{L_\mathrm{o}}(\omega_\mathrm{i} \mid x) = \underset{p(\omega_\mathrm{i} \mid x)}{\arg\min}\, \mathbb{E}_{\omega_\mathrm{o}}\left[\mathbb{E}\left[\langle L_\mathrm{o}(x, \omega_\mathrm{o})\rangle^2\right]\right] + \lambda\,(\ldots)\,. \tag{A.15}$$

Again, we use the fact that only the second moment is non-constant with respect to the PDF. Setting the derivative with respect to $p(\omega_\mathrm{i} \mid x)$ to zero yields (dropping arguments for brevity):

$$\begin{aligned}
0 &= \frac{\partial}{\partial p(\omega_\mathrm{i} \mid x)} \iint_{\Omega\times\Omega} \frac{B^2\, \mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] |\cos\theta'_\mathrm{i}|^2}{p(\omega'_\mathrm{i} \mid x)} p(\omega_\mathrm{o} \mid x)\,\mathrm{d}\omega'_\mathrm{i}\,\mathrm{d}\omega_\mathrm{o} + \lambda \\
&= -\frac{1}{p^2(\omega_\mathrm{i} \mid x)} \int_\Omega \mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] |\cos\theta_\mathrm{i}|^2\, B^2\, p(\omega_\mathrm{o} \mid x)\,\mathrm{d}\omega_\mathrm{o} + \lambda \\
&= -\frac{1}{p^2(\omega_\mathrm{i} \mid x)} \mathbb{E}_{\omega_\mathrm{o}}\left[\mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right] |\cos\theta_\mathrm{i}|^2\, B^2\right] + \lambda \\
&= -\frac{1}{p^2(\omega_\mathrm{i} \mid x)} \mathbb{E}_{\omega_\mathrm{o}}\left[\mathbb{E}\left[\langle L_\mathrm{i}\rangle^2\right]\, B^2\right] |\cos\theta_\mathrm{i}|^2 + \lambda
\end{aligned}$$

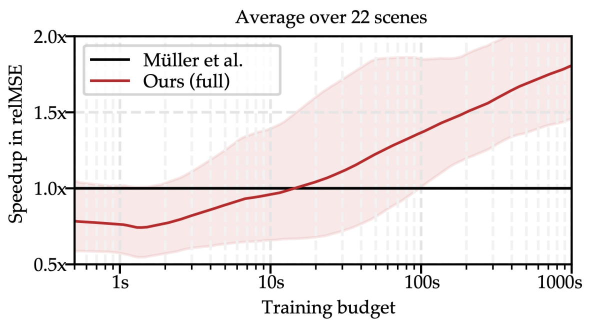


Figure A.1: *Comparison of training cost when next event estimation is disabled. We plot the ratio of the relMSE after equal time (the 'speedup'), averaged over 22 scenes, using the geometric mean. The shaded region visualizes how much that ratio varies across scenes. The error is that of a 512spp rendering after different training times.*

Again, we can substitute this into the derivative with respect to λ. Following the exact same steps as (A.8)–(A.12), we obtain:

$$p_{L_\mathrm{o}}(\omega_\mathrm{i} \mid x) \propto \sqrt{\mathbb{E}_{\omega_\mathrm{o}}\left[B^2(\omega_\mathrm{i}, x, \omega_\mathrm{o}) \left\langle L_\mathrm{i}(\omega_\mathrm{i}, x)\right\rangle^2\right]}\,|\cos\theta_\mathrm{i}|. \tag{A.16}$$

A.3 Evaluation without NEE

In addition to the evaluation in the paper, in which guiding makes use of next event estimation, we have also carried out an evaluation without NEE. As with the evaluation in the paper, we evaluated our method on a corpus of 22 scenes, all of which are rendered at a resolution of around 640×360. No Russian roulette is performed to aid comparability. However, guiding approaches do not make use of next event estimation.

A summary of our findings is given in figure A.1. Note that without next event estimation, the training time required for our density to outperform radiance-based approaches increases by a factor of ten (15 seconds as opposed to 1.5 seconds with NEE). This is due to higher levels of noise in the target density estimates. Being easier to approximate, the locally optimal density p_simple [Rath 2019] does not suffer from this problem: it consistently outperforms the baseline almost immediately, but does not reach the same performance gain that our globally optimal variant reaches for longer renders. Additionally, the locally optimal density can cause artefacts due to spatial marginalization (one example of which is the VEACH MIS scene). Due to these limitations – and since NEE is so prevalent – we generally recommend using our globally optimal density in conjunction with NEE instead.

On the next page, we discuss the differences on three representative examples.

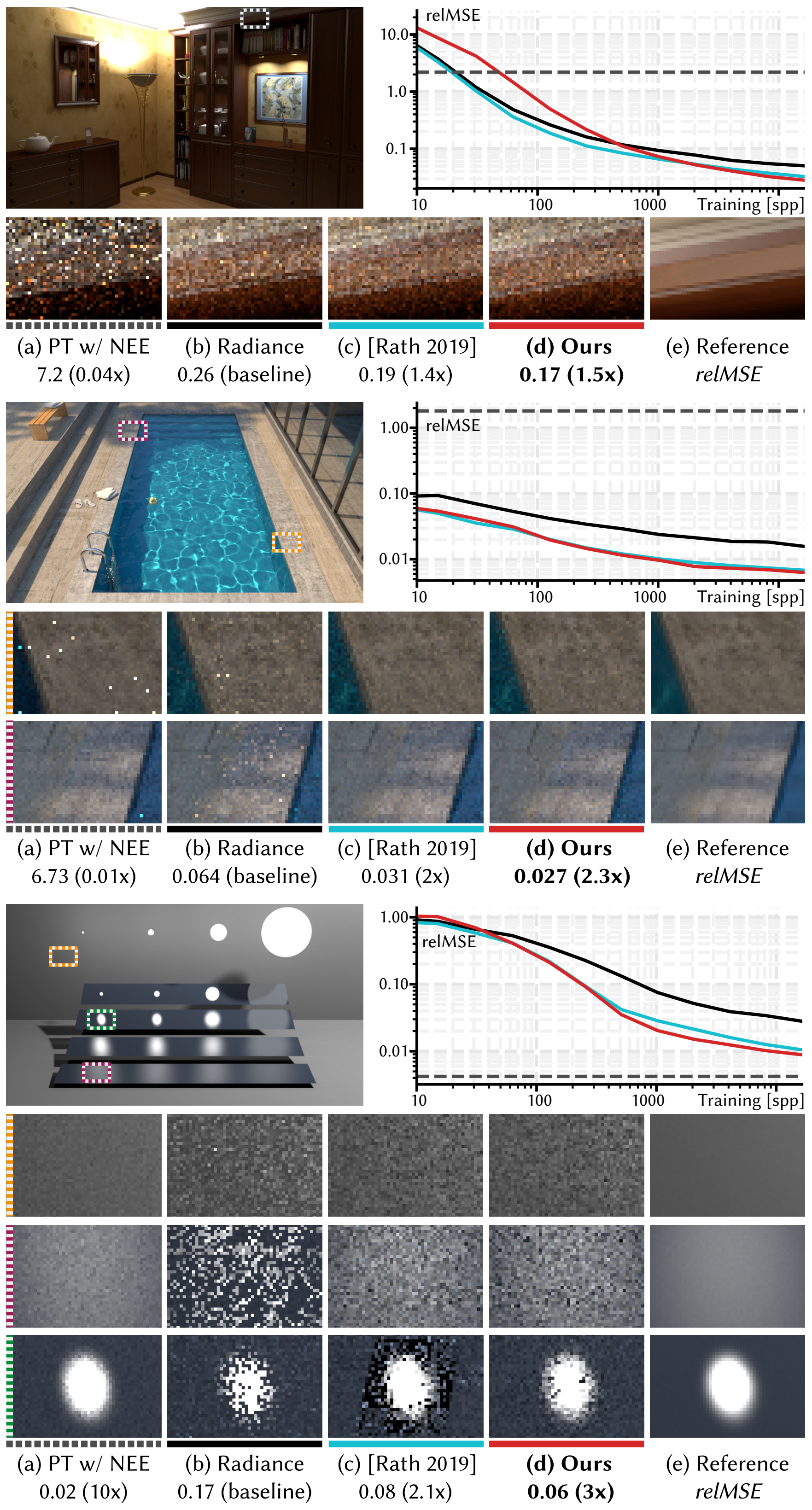


Figure A.2: *Three scenes rendered with different target densities. The guiding caches were trained for 2047spp and the images rendered at 128spp, to better compare noise patterns. To demonstrate the benefit of the guiding caches, the path tracer also only uses 128spp for rendering. Additionally, we plot the relMSE error of 512spp renders for different training budgets.*

The Bookshelf scene (figure A.2, top) features strong indirect illumination and is thus among the most challenging scenes for our density. The additional factor that our density computes, i.e., the pixel contribution, causes noisier guiding cache estimates. Therefore, our density requires at least 500 training samples per pixel (100 seconds) to outperform the baseline in this scene when no next event estimation is performed.

The Pool scene (figure A.2, center) features caustics which are challenging to render. The locally optimal density and ours behave nearly identical, since the image contributions of the guiding caches do not vary much. The right side of the pool (first row of zoom-ins) features the same light transport as discussed in the paper: sunlight is reflected by the window on the right, causing outliers on the floor in radiance-based approaches. Both of our densities eliminate these outliers by taking the variance due to the unguided decision on the glass into account. We also eliminate outliers to the left of the pool (second row of zoom-ins), which are caused by spatial marginalization: Sampling towards the sun is weighted down by the many points that lie in the shadow. Note that with NEE enabled, guiding ignores direct illumination, thereby mitigating the outliers in direct sunlight, but still producing outliers under the water surface.

The Veach MIS scene (figure A.2, bottom) shows light sources of different sizes being reflected by materials of varying roughness. With radiance-based guiding, we observe outliers on the wall due to the spatial cache boundaries, which are eliminated by both the locally and globally optimal densities (see the first row of zoom-ins). Additionally, taking the BSDF into account allows them to waste fewer samples on light sources outside of the reflection lobes, thereby improving performance in regions where both radiance-based guiding and BSDF sampling perform poorly (second row of zoom-ins). Note that the locally optimal variant sacrifices performance of the surrounding points for the bright glossy highlight (third row of zoom-ins). This shows one of the benefit of our globally optimal density, which does not favor brighter regions.

B

Efficiency-Aware Russian Roulette and Splitting

B.1 Derivatives of the objective

The functional derivatives of variance and cost, respectively, are:

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I; n\rangle\right]}{\mathrm{d}n(x)} = -p(x)\,\frac{V_{\mathcal{Y}}(x)}{n^2(x)} \tag{B.1}$$

$$\frac{\mathrm{d}\mathbb{C}\left[\langle I; n\rangle\right]}{\mathrm{d}n(x)} = p(x)\,\mathbb{C}\left[\langle H(x)\rangle \mid x\right]. \tag{B.2}$$

The derivative of their product (6.8) is obtained via the product rule,

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I; n\rangle\right]\mathbb{C}\left[\langle I; n\rangle\right]}{\mathrm{d}n(x)} = \tag{B.3}$$
$$p(x)\left(\mathbb{C}\left[\langle H(x)\rangle \mid x\right]\mathbb{V}\left[\langle I; n\rangle\right] - \frac{V_{\mathcal{Y}}(x)}{n^2(x)}\mathbb{C}\left[\langle I; n\rangle\right]\right).$$

From that, we can easily obtain our fixed-point function

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I; n\rangle\right]\mathbb{C}\left[\langle I; n\rangle\right]}{\mathrm{d}n(x)} = 0 \tag{B.4a}$$

$$\Leftrightarrow \mathbb{C}\left[\langle H(x)\rangle \mid x\right]\mathbb{V}\left[\langle I; n\rangle\right] = \frac{V_{\mathcal{Y}}(x)}{n^2(x)}\mathbb{C}\left[\langle I; n\rangle\right] \tag{B.4b}$$

$$\Leftrightarrow n(x) = \sqrt{\frac{V_{\mathcal{Y}}(x)}{\mathbb{V}\left[\langle I; n\rangle\right]}\,\frac{\mathbb{C}\left[\langle I; n\rangle\right]}{\mathbb{C}\left[\langle H(x)\rangle \mid x\right]}}, \tag{B.4c}$$

where the last equivalence holds because variance, cost, and RRS factor are always positive.

B.2 Fixed-point iterations for root finding

The following provides a very brief intro to fixed-point iteration methods applied to root finding problems. A more complete discussion can be found, e.g., in Berinde [2007], or in various textbooks on numerical analysis.

Consider the simple example of computing the root of

$$f(x) := \sqrt{x} - x = 0 \tag{B.5}$$

(which, in fact, resembles our derivative). The analytical solution of this is trivially given by $x = 1$. It can also be computed numerically, using a fixed-point iteration. For that, we rewrite the equation as

$$f(x) = 0 \Leftrightarrow g(x) - x = 0 \Leftrightarrow g(x) = x, \tag{B.6}$$

where

$$g(x) := \sqrt{x} \tag{B.7}$$

is the equivalent *fixed-point function*. By construction, every fixed-point of g, i.e., every x for which $g(x) = x$, must be a root of $f(x)$, and vice versa. This equivalence provides a way to numerically find the roots of $f(x)$ via a *fixed-point iteration*, i.e., by repeatedly updating

$$x_i = g(x_{i-1}), \tag{B.8}$$

starting with an initial guess x_0.

The convergence of this simple example is a well-known property: repeatedly applying the square root to any number will eventually converge to 1. But how can it be proven? If the fixed-point function $g : D \to D$ is a mapping from some domain D onto itself, and if g is a *contraction*, then *Banach's fixed-point theorem* [Banach 1922] guarantees that there is a unique fixed-point $x_f \in D$ and that the fixed-point iteration above will converge to that fixed point, when initialized with any $x_0 \in D$.

For $g(x)$ to be a contraction, its first derivatives need to be less than 1 over the domain D. In our example,

$$g'(x) = (2\sqrt{x})^{-1} < 1 \Leftrightarrow x > 0.25, \tag{B.9}$$

this holds for $D = (0.25, \infty)$. The domain of $g(x)$, however, is the set of all positive real numbers. So this is not a sufficient criterion for convergence. To prove convergence, we have to prove two more properties:

1. g is a mapping from D onto itself, $g(x) \in D \, \forall x \in D$
2. g moves all points x outside D closer to it, $g(x) > x \forall x \notin D$

The first property guarantees that points within the sub-domain D will not leave it. Hence, g is a contraction mapping over D and Banach's theorem states that a unique fixed-point lies within D and our iteration will converge to it. The second property ensures that all initial guesses outside D are updated such that they eventually enter D. In our simple example here, both properties are trivially verified. Thus, convergence is guaranteed for all initial guesses $x_0 \in \mathbf{R}^+$.

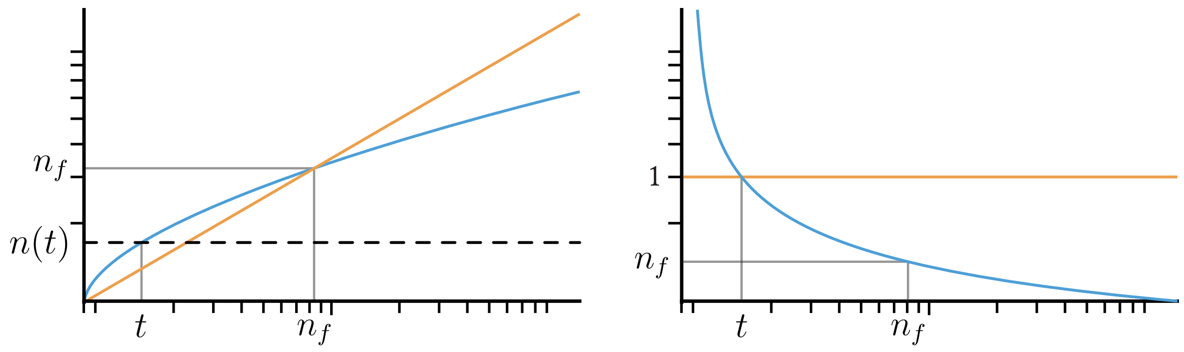


Figure B.1: *The fixed-point function is strictly monotonically increasing (left) and its derivatives have a convex hyperbolic shape (right). The fixed-point n_f lies in a sub-domain where the first derivatives are always less than one, i.e., below the orange line on the right. And, due to the monotonicity, $\gamma(n) > n \ \forall n < n_f$. Together, these two properties ensure convergence.*

B.3 Proof of convergence

The convergence of the fixed-point iterations Eqs. (6.9) and (6.12) can be proven by following the exact same steps as for the simple example in Appendix B.2. In the following, we provide the proof for the case of a single RRS factor. This readily generalizes to the multivariate / functional case, where the same conditions can be proven along each dimension separately.

To prove convergence, we have to prove three properties: There is a sub-domain (t, ∞) where the fixed-point function has bounded derivatives,

$$\exists t : \gamma'(n) < 1 \, \forall n > t, \tag{B.10}$$

the points within that sub-domain are mapped to the same domain,

$$\gamma(n) > t \, \forall n > t, \tag{B.11}$$

and the points outside the domain move closer to it,

$$\gamma(n) > n \, \forall n \leq t. \tag{B.12}$$

All three conditions follow directly from a simple analysis of the fixed-point function, as visualized in Fig. B.1. The first derivative of our fixed-point function (6.18) can be obtained easily with the chain- and quotient rules; we skip the exact equation here for brevity. It is positive everywhere and has a hyperbolic shape,

$$\gamma'(n) > 0 \qquad \gamma'(n) \in O\left(1/\sqrt{n}\right) \tag{B.13}$$

Therefore, γ is concave and strictly monotonically increasing. Further, we can trivially obtain the following limits:

$$\lim_{n\to\infty} \gamma'(n) = 0 \qquad \lim_{n\to 0^+} \gamma'(n) = \infty \qquad \lim_{n\to 0^+} \gamma(n) = 0 \tag{B.14}$$

The existance of the threshold (B.10) then follows directly from these limits and the hyperbolic shape of the first derivative.

The remaining two conditions can be proven by first asserting that the unique fixed-point is above the threshold, $n_f > t$. This can be done geometrically. By definition, the fixed-point n_f is an intersection of $\gamma(n)$ with the diagonal of the positive quadrant. As $\gamma(n)$ is a concave function, there are either zero or two such intersections. If there are two, basic geometry implies that the first occurs at a point n_a where γ approaches the diagonal from below, $\gamma'(n_a) > 1$, and the second at a point n_b where γ approaches the diagonal from above, $\gamma(n_b) < 1$. The limit $\lim_{n\to 0} \gamma = 0$ gives us the first intersection with the diagonal: it would have been at $n_a = 0$, but that is outside our domain. We know that the derivative at zero is unbounded, $\gamma'(0) = \infty$, so there must be another intersection where

$$\gamma'(n_b = n_f) < 1. \tag{B.15}$$

This intersection is our unique fixed-point in the domain of positive real numbers, and its derivative is always less than one. That, in turn, implies that $n_f > t$, i.e., the fixed-point is above the threshold.

This immediatly implies (B.12), since all $n < n_f$ must be mapped to a value greater than n, because γ is above the diagonal between 0 and n_f,

$$\gamma(n) > n \,\forall n < t < n_f. \tag{B.16}$$

Hence, the threshold itself is also mapped to a value greater than t, $\gamma(t) > t$, so every $n > t$ must also satisfy (B.11), by definition of monotonicity. $\square$

Universität des Saarlandes

How to Train Your Renderer: **Optimized Methods for Learning Path Distributions in Monte Carlo Light Transport**

Alexander Rath
Student No. 2567082

Saarbrücken, 2024